

# OFFENSIVE ROBOT CYBERSECURITY

Cyber-protecting robots by hacking-them-first

Víctor Mayoral-Vilches

Supervised by

Prof. Dr. Stefan Rass

Prof. Dr. Martin Pinzger

2018-2025

Víctor Mayoral-Vilches

# **Offensive Robot Cybersecurity**

Cyber-protecting robots by hacking-them-first  
with Game Theory and Machine Learning

**Thesis submitted for the Doctoral Programme in Computer  
Science (*Doktoratsstudium der Technischen  
Wissenschaften Informatik*)**

Department of Artificial Intelligence and Cybersecurity  
System Security Group, Alpen-Adria-Universität Klagenfurt

**2025**

© Víctor Mayoral-Vilches, 2025

*Dissertation submitted to the  
System Security Group, Alpen-Adria-Universität Klagenfurt, Klagenfurt  
University*

All rights reserved. No part of this publication may be reproduced or transmitted, in any form or by any means, without permission.

Print production: Klagenfurt University.

*Para Leo y Sam*



# Abstract

Robots, integral to modern automation and services, encounter formidable cybersecurity challenges, primarily due to their inherent complexity and the lack of vendor accountability for security, shifting the burden to end-users. *Offensive Robot Cybersecurity*, the title of this thesis, introduces a groundbreaking approach by advocating for offensive security methods empowered by automation leveraging modern Artificial Intelligence. It emphasizes the necessity of understanding attackers' tactics and identifying vulnerabilities in advance to develop effective defenses, thereby improving robots' security posture. This endeavor leverages a decade of robotics experience, employing Machine Learning and Game Theory to streamline the vulnerability identification and exploitation process. Intrinsically, the thesis uncovers a profound connection between robotic architecture and cybersecurity, highlighting that the design and creation aspect of robotics deeply intertwines with its protection against attacks. This duality—whereby the architecture that shapes robot behavior and capabilities also necessitates a defense mechanism through offensive and defensive cybersecurity strategies—creates a unique equilibrium. Approaching cybersecurity with a dual perspective of defense and attack, rooted in an understanding of systems architecture, has been pivotal in this research. This synergy affirms the adage that *the best defense is a good offense*, especially in the complex realm of robotics. Through comprehensive analysis, including ethical considerations, the development of security tools, and executing cyber attacks on robot software, hardware, and industry deployments, this thesis proposes a novel architecture for cybersecurity cognitive engines. These engines, powered by advanced game theory and machine learning, pave the way for autonomous offensive cybersecurity strategies for robots, marking a significant shift towards self-defending robotic systems. This research not only underscores the importance of offensive measures in enhancing robot cybersecurity but also sets the stage for future advancements where robots are not just resilient to cyber threats but are equipped to autonomously safeguard themselves.



# Resumen

Los robots, esenciales para la automatización y la industria moderna, enfrentan desafíos formidables en el área de la ciberseguridad, principalmente debido a su complejidad inherente –pues los robots son complejos sistemas con redes de redes– y la falta de responsabilidad por parte de los fabricantes (software y hardware) de estos robots que con frecuencia ignoran la ciberseguridad, trasladando la responsabilidad a los usuarios finales (las fábricas donde estos operan). Esta tesis titulada *Offensive Robot Cybersecurity* (Ciberseguridad Ofensiva para Robots) introduce un enfoque innovador al abogar por métodos de ciberseguridad ofensivos para proteger robots de forma económica y con recursos limitados, usando la Inteligencia Artificial como medio de automatización. Enfatiza la necesidad de comprender e imitar las tácticas de los ciber-atacantes e identificar vulnerabilidades con anticipación para desarrollar defensas efectivas, mejorando así la ciber-resiliencia de los robots.

Esta investigación construye sobre una década de experiencias en robótica, y emplea Aprendizaje Máquina (*Machine Learning*) y Teoría de Juegos (*Game Theory*) para automatizar el proceso de identificación y uso de vulnerabilidades en robots. De forma intrínseca, esta tesis estudia la profunda conexión entre la arquitectura robótica (tanto software como hardware) y la ciberseguridad, destacando que el diseño y la creación de los robots debe entrelazarse profundamente con su protección frente a ciber-ataques. Esta dualidad, donde la arquitectura que da forma al comportamiento y capacidades del robot también requiere de un mecanismo de defensa a través de estrategias de ciberseguridad, crea un equilibrio único, difícil de resolver y cambiante en el tiempo. Se aborda este reto con una perspectiva dual de ataque y posterior defensa, arraigada en un entendimiento de la arquitectura de sistemas y en el popular dicho: *la mejor defensa es un buen ataque*. A través de un análisis exhaustivo, incluyendo consideraciones éticas, el desarrollo de herramientas de ciberseguridad y la ejecución de ciberataques en software, hardware y despliegues industriales robóticos, esta tesis describe una arquitectura novedosa para motores cognitivos de ciberseguridad. Estos motores, impulsados por

teoría de juegos avanzada y aprendizaje máquina automático, permiten la implementación de estrategias de ciberseguridad ofensivas y autónomas para robots, marcando un cambio significativo hacia sistemas robóticos auto-defensivos mediante técnica ofensivas. Esta investigación no solo subraya la importancia de las medidas ofensivas en la mejora de la ciberseguridad para los robots y otros sistemas industriales, sino que también prepara el terreno para avances futuros donde los robots no solo sean resistentes a las amenazas cibernéticas, sino que también estén equipados para protegerse y responder de manera autónoma.

# Preface

My professional career in robotics began in 2013, with a first stage focused on the study of the software architecture of robots. After various international experiences, in 2014, I had the fantastic opportunity to be part of the group that started one of the most significant revolutions in robotics in recent times: the creation of the Robot Operating System 2 (ROS 2). Used today by the majority of robotics engineers around the world, ROS is the common language used in robotics to build robotic behaviors. It is a set of modular and open source software libraries that include tools, interconnection and communication elements between robotic processes. Technically, ROS provides a message-passing infrastructure between robot software components, drivers for robotic hardware, state-of-the-art robotic algorithms, and powerful development tools to simulate, debug, and visualize a robot's behaviors. Simply put, ROS allows building robot brains and my involvement in building ROS 2 allowed me to grow rapidly as an engineer, and encouraged me to create dozens of robots over the next few years.

After half a decade building robots and leading teams to do so, in 2018 I decided to embark on a journey to explore cybersecurity applied to robots. Throughout this journey, I discovered the deep connection between robotic architecture and cybersecurity in robotics, more intimate than initially perceived. Robotic architecture focuses on creation, on shaping behaviors and possibilities, while cybersecurity in robotics is oriented towards offense or protection, allowing what is built to be defended. This synergy creates a unique balance when cybersecurity is approached from a knowledge of systems architecture and with a dual perspective: defense and attack, both essential. This learning, the result of exploration and study, has been key in my research, reaffirming that in the field of robots, as in many others, the best defense is indeed a good offense.

I began this research (my PhD) formally in 2020 with a clear intention: to create the technological principles so that an artificial intelligence is capable of hacking robots better than any human. I write these lines happy to have largely satisfied this initial objective while having open sourced

much of it. During these years, I developed and participated in the creation of technology and prototypes that allow achieving superhuman capabilities in terms of offensive security in robotics. My contributions to the area of robotic cybersecurity today allow the creation of artificial systems that guide and even automate cyberattacks. No human is capable of processing so much so quickly. To respond on multiple fronts. Or training himself to exploit a new *target* or a new attack vector in a matter of a few seconds. It is surprising, exciting, but also overwhelming what these AIs can do.

I'd like to thank everyone that supported me in this journey. Thanks to Laura, my companion on this and many other trips. Her constant support, patience, and vital balance have been fundamental pillars in the most challenging moments of this research, providing not only emotional refuge, but also unique perspectives that have enriched my research. Her intelligence and empathy have been a constant source of inspiration and strength every day. Thanks to my father, whose tireless support and unconditional trust have been my beacon in the darkness of those most difficult times. His wisdom and calm approach to life's challenges have taught me to face problems with patience and determination. His example of perseverance and curiosity to "always keep learning" have been a constant guide in my life. Thanks to my mother for never giving up. Thanks to David for his drive and resilience. Always there, despite our many mistakes. Thanks to Stefan and Martin, my supervisors, for their expert guidance and unwavering scientific support throughout this process. Their ability to challenge and expand my ideas, combined with their commitment to academic excellence, has been crucial to my development as a researcher. Stefan, with his renowned expertise as a mathematician in security and game theory, has brought a unique and rigorous perspective, enriching my understanding of the theoretical aspects of cybersecurity. Martin, renowned for his deep knowledge of software and software engineering, has been instrumental in honing my hands-on approach, ensuring my applications and tools are robust and advanced. Together, they have formed an unrivaled supervisory team, offering a balanced blend of theory and practice that has been essential to the depth and success of my research.

I greatly appreciate the support received from the company Alias Robotics. This research has been partially funded by SEGRES (EXP 00131359 / MIG-20201041) grant to Alias Robotics. Its funding and trust in my work has been essential in carrying out some significant parts of my

---

research, allowing me to dive deeply into the field of robotics cybersecurity. Thanks to Endika and Unai for successfully taking the lead of the corporate daily matters, while I was focused on research. Last but not least, thanks to the rest of my family for their unwavering faith in my abilities over the past. Thanks to María for the sacrifices for a better future, and to Ramón for the stolen acorns in times of need. Now, your grandson keeps acorns on the tree in his garden, sowing perseverance and affection. Thanks also to Rosa and Adela. Their emotional support, teachings and trust in my dreams have been a constant source of motivation and courage to keep going, even in the most difficult times.

For any reader, but especially for my two boys, may this learning encourage you to approach the challenges of the future with courage, resilience and cunning.

**Víctor Mayoral-Vilches**

Zuia, June 2025



# Prefacio

Mi carrera profesional en robótica comenzó en 2013, con una primera etapa centrada en el estudio de la arquitectura software de los robots. Tras varias experiencias internacionales, en 2014, tuve la fantástica oportunidad de formar parte del grupo que inició una de las revoluciones en robótica más significativas de los últimos tiempos: la creación del Sistema Operativo de Robots 2 (ROS 2). Usado a día de hoy por la mayoría de ingenieros e ingenieras de robótica por todo el mundo, ROS es el lenguaje común que se utiliza en robótica para construir comportamientos robóticos. Es un conjunto de librerías de software modulares y de código abierto que incluyen herramientas, y elementos de interconexión y comunicación entre procesos robóticos. Técnicamente, ROS proporciona un sistema de transmisión de mensajes entre componentes de software, controladores para el hardware robótico, algoritmos robóticos de última generación y potentes herramientas de desarrollo para simular, depurar y visualizar los comportamientos de un robot. De forma sencilla, ROS permite crear cerebros robóticos y mi participación en la construcción de ROS 2 me permitió crecer rápidamente como ingeniero, y me animó a crear decenas de robots durante los años siguientes.

Tras media década construyendo robots y liderando equipos para ello, en 2018 decidí emprender un viaje para explorar la ciberseguridad aplicada a los robots. A lo largo de este viaje, he descubierto la profunda conexión entre la arquitectura robótica y la ciberseguridad en robótica, más íntima de lo que inicialmente se percibe. La arquitectura robótica se enfoca en la creación, en moldear comportamientos y posibilidades, mientras que la ciberseguridad en robótica se orienta hacia la ofensiva o la protección, permitiendo defender lo construido. Esta sinergia crea un balance único cuando se aborda la ciberseguridad desde un conocimiento de arquitectura y con una perspectiva dual: defensa y ataque, ambas esenciales. Este aprendizaje, fruto de la exploración y el estudio, ha sido clave en mi investigación, reafirmando que en el ámbito de los robots, como en otros muchos, la mejor defensa es efectivamente un buen ataque.

Comencé esta investigación (mi doctorado) formalmente en 2020 con

una intención clara: crear los principios tecnológicos para que una inteligencia artificial sea capaz de hackear robots mejor que cualquier humano, para después mitigar dichas vulnerabilidades antes de aparezcan los primeros ataques. Escribo estas líneas contento de haber satisfecho en gran medida este objetivo inicial. Durante estos años, he desarrollado y participado en la creación de tecnología y prototipos que permiten alcanzar capacidades sobre-humanas en lo que a la seguridad ofensiva en robótica se refiere. Mucho de este trabajo ha sido contribuido abiertamente (*open sourced*), con licencias permisivas. Mis contribuciones al área de la ciberseguridad robótica permiten hoy la creación de sistemas artificiales que orientan e incluso automatizan ciber-ataques. Ningún humano es capaz de procesar tanto y tan rápido. De responder a múltiples frentes. O de capacitarse para explotar un nuevo *target* o un nuevo vector de ataque en cuestión de pocos segundos. Es sorprendente, excitante pero también sobrecogedor.

Me gustaría terminar agradeciendo a todas aquellas y aquellos que me han apoyado en esta aventura. Gracias a Laura, mi compañera en este y otros muchos viajes. Su apoyo constante, su paciencia y su equilibrio vital han sido pilares fundamentales en los momentos más desafiantes de esta investigación, proporcionando no solo un refugio emocional, sino también perspectivas únicas que han enriquecido mi investigación. Su inteligencia y empatía han sido una fuente constante de inspiración y fuerza cada día. Gracias a mi padre, cuyo apoyo incansable y confianza incondicional han sido mi faro en la oscuridad de esas épocas más difíciles. Su sabiduría y su enfoque sereno ante los desafíos de la vida me siguen enseñando a enfrentar los problemas con paciencia y determinación. Su ejemplo de perseverancia y curiosidad por "siempre seguir aprendiendo" han sido una guía constante en mi vida. Gracias a mi madre por nunca darse por vencida. Gracias a David por su fuerza y resiliencia. Siempre junto a mí, a pesar de nuestras muchas equivocaciones. Gracias a Stefan y a Martin, mis supervisores, por su orientación experta y apoyo científico inquebrantable durante todo este proceso. Su capacidad para desafiar y expandir mis ideas, combinada con su compromiso con la excelencia académica, ha sido crucial para mi desarrollo como investigador. Stefan, con su renombrada experiencia como matemático en ciberseguridad y teoría de juegos, ha aportado una perspectiva única y rigurosa, enriqueciendo mi comprensión de los aspectos teóricos de la ciberseguridad. Martin, reconocido por

---

sus profundos conocimientos en software y software engineering, ha sido fundamental para pulir mi enfoque práctico, asegurando que mis aplicaciones sean robustas y avanzadas. Juntos, han formado un equipo de supervisión inigualable, ofreciendo una combinación equilibrada de teoría y práctica que ha sido esencial para la profundidad y el éxito de mi investigación.

Agradezco enormemente el apoyo recibido por parte de la empresa Alias Robotics. Esta investigación ha sido parcialmente financiada por el proyecto SEGRES (EXP 00131359 / MIG-20201041) concedido a Alias Robotics. Su financiación y confianza en mi trabajo han sido esenciales para llevar a cabo algunas partes significativas de mi investigación, permitiéndome sumergirme profundamente en el campo de la ciberseguridad robótica. Gracias a Endika y Unai por liderar con éxito el día a día corporativo, mientras yo me centraba en la investigación. Por último, pero no menos importante, gracias al resto de mi familia por su fé inquebrantable en mis capacidades a lo largo de muchos años. Gracias a María por los sacrificios por un futuro mejor y a Ramón por las bellotas robadas en tiempos de necesidad. Ahora, vuestro nieto guarda bellotas en el árbol de su jardín, sembrando perseverancia y cariño. Gracias también a Rosa y a Adela. Su apoyo emocional, sus enseñanzas y su confianza en mis sueños han sido una fuente constante de motivación y coraje para seguir adelante, incluso en los momentos más difíciles.

Para cualquier lector o lectora, pero especialmente para mis dos chicos, que este aprendizaje os impulse a abordar con valentía, resiliencia y astucia los retos del futuro.

**Víctor Mayoral-Vilches**

Zuía, June 2025



# Contents

- Abstract** **iii**
- Resumen** **v**
- Preface** **vii**
- Prefacio** **xi**
- Contents** **xv**
- List of Figures** **xxiii**
- List of Tables** **xxix**
  
- I Motivation** **1**
- 1 Introduction** **3**
  - 1.1 Why robot cybersecurity? . . . . . 3
  - 1.2 About ROS . . . . . 3
  - 1.3 Research Objectives and Approach . . . . . 6
    - 1.3.1 Ethics . . . . . 9
    - 1.3.2 Blueprints . . . . . 10
    - 1.3.3 Cyber-attacking robots . . . . . 11
    - 1.3.4 Automated offensive robot cybersecurity . . . . . 11
  
- 2 Background context** **13**
  - 2.1 Biographical cornerstones in robot cybersecurity . . . . . 13
  - 2.2 Robotics software quality, safety and security . . . . . 15
  - 2.3 Cybersecurity Across Various Domains: IT, OT, IoT, and Robotics . . . . . 19
    - 2.3.1 Definitions and Differentiations . . . . . 20
    - 2.3.2 Misunderstandings in Robot Security . . . . . 21

xv

2.4	Machine Learning and Game Theoretic approaches to offensive cybersecurity . . . . .	22
<b>II</b>	<b>Ethics</b>	<b>27</b>
<b>3</b>	<b>Ethical Landscape in Robot Cybersecurity</b>	<b>29</b>
3.1	Introduction . . . . .	29
3.2	Background . . . . .	30
3.3	Surveying security in robotics communities . . . . .	31
3.3.1	Surveying the ROS community . . . . .	31
3.3.2	Surveying the PX4 community . . . . .	32
3.3.3	Surveying the ROS-Industrial community . . . . .	34
3.3.4	Surveying the European robotics community at the European Robotics Forum (ERF) (2020) . . . . .	37
3.4	Security research results in robotics . . . . .	37
3.5	Discussion . . . . .	39
3.5.1	Ethical Considerations During Surveys . . . . .	39
3.5.2	Vulnerability Disclosure Policies . . . . .	41
3.5.3	Lack of learning resources . . . . .	47
3.6	Conclusion . . . . .	48
<b>III</b>	<b>Blueprints</b>	<b>53</b>
<b>4</b>	<b>Cybersecurity in ROS</b>	<b>55</b>
4.1	Introduction . . . . .	55
4.2	Background . . . . .	57
4.3	Approach . . . . .	59
4.3.1	Modeling . . . . .	59
4.3.2	Authentication . . . . .	61
4.3.3	Authorization . . . . .	62
4.3.4	Generation . . . . .	63
4.3.5	Deployment . . . . .	64
4.3.6	Monitoring and mitigation . . . . .	64
4.4	Application and analysis . . . . .	67
4.5	ROS 2 over VPN: Tunneling Communications . . . . .	69
4.5.1	Challenges in Securing ROS 2 Communications . . . . .	69

---

4.5.2	ROS 2 over VPN: A Layered Security Approach	69
4.5.3	Implementation . . . . .	70
4.6	Conclusion . . . . .	71
<b>5</b>	<b>The Cost of Security: Benchmarking Robot Computations</b>	<b>73</b>
5.1	Introduction . . . . .	73
5.2	Background . . . . .	78
5.3	RobotPerf: Principles & Methodology . . . . .	79
5.3.1	Non-Functional Performance Testing . . . . .	79
5.3.2	ROS 2 Integration & Adaptability . . . . .	79
5.3.3	Platform Independence & Portability . . . . .	79
5.3.4	Flexible Methodology . . . . .	80
5.3.5	Opaque Performance Tests . . . . .	81
5.3.6	Reproducibility & Consistency . . . . .	81
5.3.7	Metrics . . . . .	82
5.3.8	Current Benchmarks and Categories . . . . .	82
5.3.9	Run Rules . . . . .	83
5.4	Evaluation . . . . .	83
5.4.1	Fair and Representative Assessment of Heterogeneity . . . . .	83
5.4.2	Quantitative Approach to Hardware Selection	85
5.4.3	Rigorous Assessment of Acceleration Benefits	86
5.4.4	Benchmarking the security impact in ROS 2 communications . . . . .	87
5.5	Conclusion . . . . .	88
<b>6</b>	<b>Hardware Accelerators in Robotics</b>	<b>91</b>
6.1	Introduction . . . . .	91
6.2	Background . . . . .	95
6.2.1	ROS and ROS 2 . . . . .	95
6.2.2	Hardware Acceleration for ROS and ROS 2 . . . . .	95
6.3	An Open Architecture for Hardware Acceleration in ROS 2	96
6.3.1	Extending the ROS 2 Build System . . . . .	97
6.3.2	Extending the ROS 2 Build Tools . . . . .	98
6.3.3	Adding Firmware Extensions . . . . .	98
6.3.4	Low-Overhead Real-Time Tracing & Benchmarking . . . . .	98
6.4	Case Study: Accelerating ROS 2 Perception . . . . .	100

6.4.1	Method . . . . .	100
6.4.2	CPU-Only Tracing Results . . . . .	101
6.4.3	Accelerating and Benchmarking CPU & FPGA . . . . .	101
6.5	Conclusion . . . . .	105
6.6	Future work . . . . .	106
<b>IV</b>	<b>Cyber-attacking robots</b>	<b>109</b>
<b>7</b>	<b>Attacking Robot Software</b>	<b>111</b>
7.1	Introduction . . . . .	112
7.2	Background . . . . .	113
7.3	DDS and Real-Time Publish-Subscribe (RTPS) Packets . . . . .	115
7.4	Research Methodology and Technical Details . . . . .	118
7.4.1	A New Scapy Layer to Dissect and Forge RTPS and DDS Data . . . . .	118
7.4.2	Source-code and Binary Fuzzing . . . . .	122
7.4.3	Internet-wide Scanning for RTPS Endpoints . . . . .	133
7.5	Conclusion . . . . .	137
<b>8</b>	<b>Attacking Robot Hardware</b>	<b>139</b>
8.1	Introduction . . . . .	139
8.2	Robot teardown . . . . .	142
8.2.1	Case Study 1: Teardown of an industrial collaborative robot . . . . .	143
8.2.2	Case Study 2: Teardown of a next-gen indus- trial collaborative robot . . . . .	145
8.2.3	Case Study 3: Teardown of a mobile industrial robot . . . . .	147
8.3	Teardown-enabled security research . . . . .	149
8.4	Lessons learned . . . . .	150
8.5	Conclusions . . . . .	151
<b>9</b>	<b>Attacking Robots in Industry</b>	<b>155</b>
9.1	Introduction . . . . .	155
9.2	Background . . . . .	156
9.3	Akerbeltz . . . . .	157
9.3.1	Target selection and rationale . . . . .	157

---

9.3.2	Ransomware’s flow . . . . .	160
9.3.3	Discussion . . . . .	162
9.4	Conclusions . . . . .	163
<b>V</b>	<b>Automation</b>	<b>165</b>
<b>10</b>	<b>Determining the Attack Target with Game Theory</b>	<b>167</b>
10.1	Introduction . . . . .	168
10.2	Case Studies . . . . .	173
10.2.1	Implementation Remarks . . . . .	174
10.2.2	Robot Case Studies . . . . .	175
10.3	Results and Comparison . . . . .	176
10.3.1	Baseline Comparison: A Best-Effort defense Policy . . . . .	176
10.3.2	Overview of Experiments . . . . .	178
10.3.3	MARA: Results . . . . .	178
10.3.4	MiR100: Results . . . . .	181
10.4	Discussion . . . . .	184
10.4.1	Incomplete Attack Graphs and Zero-Day Exploits	184
10.4.2	Cutting the rope vs. Changing the Attack Graph	185
10.4.3	Further Generalizations . . . . .	185
10.4.4	Complexity and Scalability . . . . .	186
10.4.5	Including Signals about Adversarial Activities	187
10.5	Conclusions and Outlook . . . . .	187
<b>11</b>	<b>Guiding Attacks with Machine Learning</b>	<b>191</b>
11.1	Introduction . . . . .	192
11.2	Background . . . . .	198
11.2.1	Penetration Testing . . . . .	198
11.2.2	Large Language Models . . . . .	199
11.3	Penetration Testing Benchmark . . . . .	199
11.3.1	Motivation . . . . .	199
11.3.2	Benchmark Design . . . . .	200
11.4	Exploratory Study . . . . .	201
11.4.1	Testing Strategy . . . . .	202
11.4.2	Evaluation Settings . . . . .	202
11.4.3	Capability Evaluation (RQ1) . . . . .	204

11.4.4	Comparative Analysis (RQ2)	206
11.5	Methodology	208
11.5.1	Overview	208
11.5.2	Design Rationale	209
11.5.3	Reasoning Module	210
11.5.4	Generation Module	213
11.5.5	Parsing Module	215
11.5.6	Active Feedback	216
11.5.7	Discussion	216
11.6	Evaluation	217
11.6.1	Evaluation Settings	218
11.6.2	Performance Evaluation (RQ3)	218
11.6.3	Strategy Evaluation (RQ4)	219
11.6.4	Ablation Study (RQ5)	220
11.6.5	Practicality Study (RQ6)	221
11.7	Discussion	222
11.8	Conclusion	223
<b>12</b>	<b>Conclusion and future work</b>	<b>229</b>
<b>VI</b>	<b>Appendices</b>	<b>233</b>
<b>A</b>	<b>Research Results</b>	<b>235</b>
A.1	Thesis framework	235
A.2	Past work	235
A.3	Peer-reviewed articles	238
A.4	Preprint articles (including <i>under review</i> )	240
A.5	Tutorials and talks	241
A.6	Vulnerabilities found	241
<b>B</b>	<b>Game Theoretic Appendix</b>	<b>245</b>
B.1	Cut-The-Rope Background	245
B.2	The Model	248
B.2.1	Definition of Payoffs	251
B.2.2	Solution Concept	254
B.3	Movement Patterns	255
B.3.1	Periodically Active Defender	255

B.3.2	Probabilistic Success on Exploits . . . . .	256
B.3.3	Checks with Random Intervals (“Exponential Strategy”) . . . . .	257
B.3.4	Spot Checks with Random Intervals and Prob- abilistic Success on Exploit . . . . .	258
B.4	Analytic Results . . . . .	259
B.5	Application for Risk Control . . . . .	263
B.6	Derivation of the probability (B.13) . . . . .	265
<b>C</b>	<b>Machine Learning Appendix</b>	<b>267</b>
	<b>Bibliography</b>	<b>271</b>



# List of Figures

1.1	The robotics stack, including ROS 2 core layers. Reworked from [6] . . . . .	5
1.2	Thesis building blocks . . . . .	9
2.1	Software integrity as the union between security and quality	15
2.2	Security as the target of this thesis. . . . .	16
2.3	Safety, quality and security relationship . . . . .	18
2.4	Relationship between various technology domains . . . . .	19
2.5	Captures from Mobile Industrial Robots (MiR) documentation indicating a purely IT security approach. . . . .	22
2.6	Attacker model for robotics. Adapted from [67, 68, 69, 70] . . . . .	23
3.1	Surveying the ROS robotics community (2019), from [83] . . . . .	33
3.2	Surveying the PX4 robotics community (2020), adapted from [83]. . . . .	35
3.3	Surveying the ROS-Industrial robotics community (2020), adapted from [83]. . . . .	36
3.4	Surveying the European robotics community (ERF 2020), adapted from [83]. . . . .	38
3.5	Vulnerability data for various robots, adapted from [83]. . . . .	40
3.6	Summary of Vulnerability Disclosure Policies in Robotics . . . . .	42
4.1	A subset of the computational graph of the simulated robot including sensor and control topics, adapted from [102]. . . . .	66
4.2	FogROS2 proposes an additional security layer to ROS 2 communications using a VPN, reproduced from [103]. . . . .	70
5.1	A high level overview of RobotPerf, reproduced from [133] . . . . .	76
5.2	Benchmarking results on diverse hardware platforms across perception, localization, control, and manipulation workloads defined in RobotPerf beta Benchmarks. Reproduced from [133]. . . . .	84

## List of Figures

---

5.3	Benchmark comparison of perception latency (ms) on AMD's Kria KR260 with and without the ROBOTCORE Perception accelerator. Reproduced from [133]. . . . .	86
5.4	Mean latency of ROS 2 intra-network communications with various different security measures after 1000000 samples .	87
6.1	The open architecture for hardware acceleration in ROS 2, reproduced from [139]. . . . .	93
6.2	Overview of the components of the open architecture for hardware acceleration in ROS 2. Reproduced from [139]. . .	96
6.3	Computational graph of our case study perception application, reproduced from [139]. . . . .	99
6.4	Tracepoints instrumented across ROS 2 abstraction layers on CPU for case study computational graph (Figure 6.3). Breakdown summary in Figure 6.5. Reproduced from [139].	100
6.5	Breakdown of CPU runtime derived from tracing and benchmarking. Total computation time of our case study graph is dominated by message passing overheads, a bottleneck consuming over 73.3% of total runtime. Reproduced from [139].	102
6.6	Total runtime of CPU baseline and FPGA, FPGA-Integrated, and FPGA-Streaming hardware-accelerated implementations of case study application. Acceleration enables up to 26.96% speedup over CPU. Reproduced from [139]. . . . .	104
7.1	We found exposed DDS systems in 34 countries [97], including vulnerable ones, identified via distinct IPs leaking data. Color code hints about criticality by vulnerable targets: red for most, yellow/grey for least. Reproduced from [97]. . . . .	113
7.2	DDS is a standardized software library used for software-based controlled systems, directly or via ROS 2. Reproduced from [97]. . . . .	114
7.3	Simplified software-based control system with actuators, controller, sensors, communicating by exchanging data over DDS. Reproduced from [97]. . . . .	116
7.4	An example of a dissected RTPS message with a DATA submessage. Reproduced from [97]. . . . .	116

---

7.5	Diagram of the research methodology and workflow we used: The numbered items indicate the sequence of steps we followed throughout this research, beginning with the development of a packet dissector. Reproduced from [97]. . . . .	119
7.6	With the Scapy RTPS layer, a developer can create arbitrarily complex (and unexpected but valid) RTPS packets. Reproduced from [97]. . . . .	120
7.7	The UDP payload contains the RTPS header and subsequent data. Reproduced from [97]. . . . .	121
7.8	In case of UDP or TCP transport, the locator is the IP-port pair. Reproduced from [97]. . . . .	122
7.9	We found the amplification vulnerability almost by accident, by setting the PID_DEFAULT_UNICAST_LOCATOR to the first IP address that came to mind and easy to type. Reproduced from [97]. . . . .	122
7.10	Abstract representation of the data flow in a typical DDS/RTPS message exchange. From a fuzzing perspective, the deserialize() step is the fuzz target. Reproduced from [97].	124
7.11	Starting from a debug trace, we found interesting functions and explored further by manually looking into the source code with the aid of Visual Studio Code engine. Reproduced from [97]. . . . .	124
7.12	Typical switch-case control structure found similar in all DDS implementations. Each of the branches takes care of one RTPS sub-message type (e.g., DATA, INFO_DST, HEARTBEAT). Reproduced from [97]. . . . .	125
7.13	(Top screenshot) In some cases, we inserted a memcpy() to directly fuzz the input in the right place when it was not possible to further decompose the function into a smaller, self-contained fuzz target. In other cases, we used the original code to dump the data received by the de-serialization routine to verify that we found the right fuzz target (bottom screenshot). Reproduced from [97]. . . . .	126
7.14	Example harness for OpenDDS RTPS deserialization routine written for AFL++ using persistent mode. Reproduced from [97]. . . . .	127

## List of Figures

---

7.15	Cyclone DDS harness required a mock network subsystem. The actual fuzzing is happening at line 86, where we pass the configuration initializer a pointer memory-mapped XML file. Reproduced from [97]. . . . .	128
7.16	Backtrace of a crash found by libFuzzer on Cyclone DDS, which led us to CVE-2021-38441, a multi-byte heap-write primitive. Reproduced from [97]. . . . .	129
7.17	Finding fuzz targets in RTI Connex DDS, CoreDX DDS, and Gurum DDS required us to reverse engineer the binary libraries, which was easy as the vendor did not use any anti-reverse engineering measures. Reproduced from [97]. . . . .	130
7.18	We used the debug trace and the decompiled code (see Figure 7.20) to create a harness for UnicornAFL. Reproduced from [97]. . . . .	131
7.19	A simple scripting of RADAMSA can lead to surprising results. Reproduced from [97]. . . . .	132
7.20	Register state and sample crash cases in RTI Connex DDS Connector (Python bindings) found with RADAMSA from the original XML configuration file. Reproduced from [97]. . . . .	133
7.21	The dashboard of our DDS-scanning system allows analysts to explore the data. User interface is not publicly available. Reproduced from [97]. . . . .	134
7.22	We used spoofed RTPS discovery messages sent via ZMap to collect answers from valid DDS endpoints and filtered echoed and invalid responses. Reproduced from [97]. . . . .	136
7.23	Starting from a template RTPS packet, the spoofer instantiates it for a given collector (IP and port) and sends it out via ZMap. The collector uses the RTPS packets sent out to decide whether the received packets are valid and not simply duplicates of what was sent out. Reproduced from [97]. . . . .	137
8.1	Our teardown methodology for robots . . . . .	143
8.2	UR3 collaborative robot teardown. Adapted from [104]. . . . .	144
8.3	UR3e collaborative robot teardown. Adapted from [104]. . . . .	146
8.4	MiR-100 mobile industrial robot teardown. Adapted from [104].	148
8.5	Simplified electrical diagrams of Universal Robots UR3 CB-Series (8.5a) and UR3e e-Series (8.5b) collaborative robots. Reproduced from [104]. . . . .	154

---

9.1	Universal Robots UR3. Reproduced from [88]. . . . .	159
9.2	UR3 PolyScope Robot User Interface hijacked. In this case, Akerbeltz locks and disables most functions and a message is overlaid on top. Reproduced from [88]. . . . .	161
10.1	Game theoretic contributions to our architecture to develop fully automated offensive robot cybersecurity agent, a cognitive cybersecurity engine denominated Malism. The Cut-The-Rope method implemented sits between ExploitFlow and PentestGPT, receiving the exploit tree from ExploitFlow , reasoning about it to determine optimal attack vectors, passing such vectors to PentestGPT to transform them into actionable exploits and finally, feeding such exploits back to ExploitFlow for their execution and result digestion. . . . .	170
10.2	Cut-The-Rope (static game) inside the continuous process of permanent system hardening to fight against. Reproduced from [283]. . . . .	171
10.3	MARA use case results for periodic spot checks (Section B.3.1). Adapted from [283]. . . . .	179
10.4	MARA use case results for spot checks at random intervals. Adapted from [283]. . . . .	180
10.5	MiR100 use case results for periodic spot checks (Section B.3.1). Adapted from [283]. . . . .	182
10.6	MiR100 use case results for spot checks at random intervals (Section B.3.4). Adapted from [283]. . . . .	183
11.1	Machine Learning contributions to our architecture to develop fully automated offensive robot cybersecurity agent, a cognitive cybersecurity engine denominated Malism. . . . .	193
11.2	Overview of strategy to use LLMs for penetration testing. Reproduced from [292]. . . . .	203
11.3	Overview of PentestGPT. Reproduced from [292]. . . . .	209
11.4	Pentesting Task Tree in a) visualized tree format, and b) natural language format encoded in LLM. Reproduced from [292]. . . . .	212
11.5	A demonstration of the task-tree update process on the testing target <i>HTB-Carrier</i> . Reproduced from [292]. . . . .	213

## List of Figures

---

11.6	The performance of GPT-3.5, GPT-4, PentestGPT-GPT-3.5, and PentestGPT-GPT-4 on overall target completion and sub-task completion. Reproduced from [292]. . . . .	225
11.7	Penetration testing strategy comparison between GPT-4 and PentestGPT on <i>VulnHub-Hackable II</i> . Reproduced from [292].	226
11.8	The performance of PentestGPT, PentestGPT-No-Annotation, PentestGPT-Operation-Only, and PentestGPT-Parameter-Only on both normalized average code coverage ( $\mu LOC$ ) and bug detection. Reproduced from [292]. . . . .	227
12.1	Our architecture and framework to develop fully automated offensive robot cybersecurity agents, which name <i>cybersecurity cognitive engines</i> in Chapter 10: Malism. . . . .	230
B.1	Basic Gameplay of Cut-The-Rope. Reproduced from [283]. .	248
B.2	Cut-The-Rope (static game) inside the continuous process of permanent system hardening (dynamic game). Reproduced from [283]. . . . .	264

# List of Tables

- 2.1 Summary of security-related terms . . . . . 18
- 2.2 Comparative analysis of cybersecurity across different technologies . . . . . 21
  
- 3.1 Robot Hacks: Notable Cybersecurity Incidents in Robotics . . 48
- 3.2 Summary of Observations from Robotics Security Surveys and Research . . . . . 50
  
- 5.1 Comparative evaluation of representative existing robotics benchmarks with RobotPerf across essential characteristics for robotic systems. . . . . 77
- 5.2 Grey-box vs. black-box benchmarking trade-offs. . . . . 80
  
- 7.1 A list of DDS implementations that we analyzed in this research. Reproduced from [97]. . . . . 117
- 7.2 Assuming up to one DDS domain and at least one participant, we tested the open UDP ports of each of the six reference implementations. All those marked with checks could detect any of the identified ports just by scanning for four ports listed on the last row of the table. Reproduced from [97]. . . . . 135
  
- 8.1 The 17 novel (new CVE IDs) vulnerabilities encountered during a period of two years in the robots of Teradyne and as a result of an initial hardware teardown. . . . . 153
  
- 10.1 Overview of Experiments . . . . . 178
  
- 11.1 Overall performance of LLMs on Penetration Testing Benchmark. . . . . 204
- 11.2 Top 10 Types of Sub-tasks completed by each tool. . . . . 205
- 11.3 Top Unnecessary Operations Prompted by LLMs on the Benchmark Targets. Experiments conducted on June 2023. . . 206
- 11.4 Top causes for failed penetration testing trials . . . . . 207

## List of Tables

---

11.5	PentestGPT performance over the active HackTheBox Challenges. . . . .	222
A.1	Past work group 1 . . . . .	236
A.2	Past work group 2 . . . . .	237
A.3	Peer-reviewed articles group 1. . . . .	238
A.4	Peer-reviewed articles group 2. . . . .	239
A.5	Preprints, this includes <i>under review</i> documents . . . . .	240
A.6	Workshops, tutorials and talks in peer-reviewed conferences. . . . .	242
A.7	Some of the most relevant robot vulnerabilities found during research. . . . .	243
C.1	Summarized 26 types of sub-tasks in the proposed penetration testing benchmark. . . . .	270

Part I

# **Motivation**



# Chapter 1

## Introduction

### 1.1 Why robot cybersecurity?

For the last fifty years, we have been witnessing the dawn of the robotics industry, but robots are not being created with security as a concern. Security in robotics is often mistaken with safety. From industrial to consumer robots, going through professional ones, most of these machines are not prepared for cyber-threats and resilient to security vulnerabilities. Manufacturers' concerns, as well as existing standards, focus mainly on safety. Security in robotics is still not being considered as a relevant matter. The integration between these two areas from a risk assessment perspective was studied in [1, 2] which resulted in a unified security and safety risk framework. Commonly, robotics safety is understood as developing protective mechanisms against accidents or malfunctions, whilst security is aimed to protect systems against risks posed by malicious actors [3]. A slightly alternative view is the one that considers safety as protecting the environment from a given robot, whereas security is about protecting the robot from a given environment. In this proposal we adopt the later and tackle the emerging need to address security in robotics.

### 1.2 About ROS

Robots are networks of networks, with sensors capturing data, passing to compute technologies, and then on to actuators and back again in a deterministic manner. These networks can be understood as the nervous system of the robot, passing across compute *Nodes*, that represent neurons. Like the human nervous systems, real-time information across all these computational *Nodes* is fundamental for the robot to behave coherently.

"*Robot brains*" are built with this same philosophy. Behaviors take the form of computational graphs, with data flowing between *Nodes* operating intra-process, inter-process and across physical networks (communication

## 1. Introduction

---

buses), while mapping to underlying sensors, compute technologies and actuators. The Robot Operating System (ROS) [4] is a robotics framework for robot application development. ROS enables a robotics developer to build these computational graphs and create robot behaviors by providing libraries, a communication infrastructure, drivers and tools to put it all together. It provides an open source codebase with a commercially friendly license that helps roboticists reduce the effort to bring up robot behaviors. Widely used to govern interactions across robots, at the time of writing, the original ROS article [4] has been cited about 12000 times, which shows its wide acceptance for research and academic purposes. ROS was born in this environment: its primary goal was to provide the software tools and libraries that users would need to employ to undertake novel robotics research and development. Adoption in industry has also been rapidly increasing over the last few years. According to the latest ROS community metrics [5] sampled every year in July, the number of ROS downloads has increased by over 50%, with about 600 million downloads between July of 2020 and July of 2021 <sup>1</sup>. Moreover, based on the download percentages reported from packages.ros.org, we observe a significant increase in adopting ROS 2, which suggests that within 2024 there will be more users using ROS 2 than its predecessor <sup>2</sup>.

Following from the success of ROS, and redesigned for production environments, the second generation of ROS, ROS 2, is rapidly becoming a standard in the robotics industry. Built upon the Data Distribution Service (DDS [7]) as its default communication middleware (used in safety-critical scenarios), ROS 2 aspiration was to offer a more robust framework that could be integrated into products and industry. Figure 1.1 depicts the ROS 2 robotics stack, relative to the OSI model. From a technical viewpoint, it provides three main components:

1. **ROS Plumbing:** ROS provides an efficient message-passing infrastructure facilitating seamless communication between different

---

<sup>1</sup>There's a slowdown identified in the year 2022 according to the data. This is justified mostly due to two reasons: 1) COVID influence and 2) slowdown of ROS contributors and interest due to Intrinsic's acquisition of Open Robotics

<sup>2</sup>We also note that past studies estimated that by 2024, 55% of the total commercial robots shipped that year would include at least one ROS package. For more details, refer to <https://www.businesswire.com/news/home/20190516005135/en/Rise-ROS-55-total-commercial-robots-shipped>.

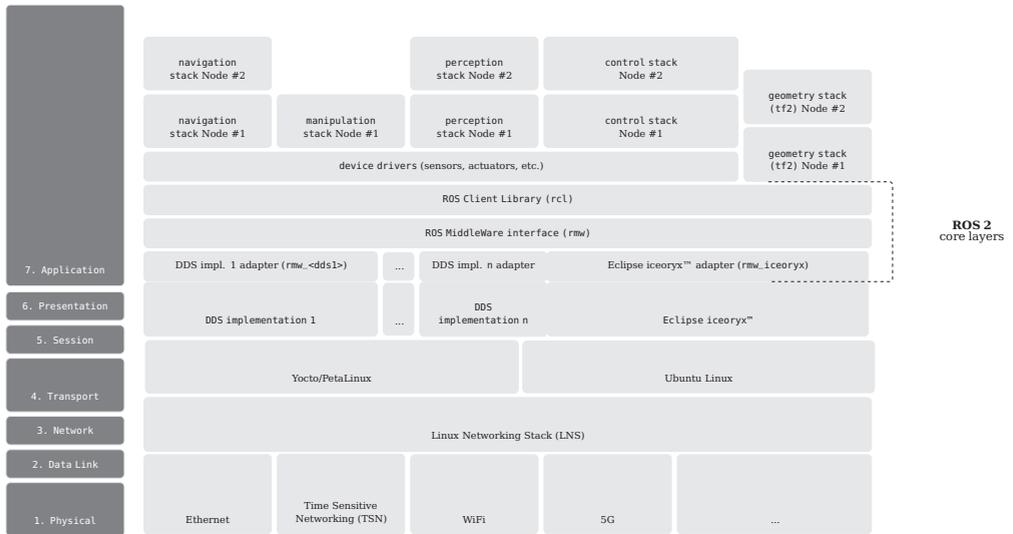


Figure 1.1: The robotics stack, including ROS 2 core layers. Reworked from [6]

robotic system components. This includes topics for asynchronous message exchange, services for synchronous requests, and actions for handling long-duration tasks. ROS’s process management, or node management, ensures efficient handling of various processes, promoting a modular and scalable design. Additionally, ROS device drivers are integral for interfacing with the robot’s hardware, ensuring effective control and data acquisition from sensors and actuators.

2. **ROS Tools:** ROS boasts a suite of visualization tools like RViz and rqt, essential for real-time monitoring and understanding of a robot’s operation. CLI tools such as rostopic, rosnodet, and rosservice offer convenient system interaction and control, crucial for development and troubleshooting. The simulation environment, provided by tools like Gazebo [8], enables safe and cost-effective testing in varied scenarios. Moreover, ROS’s data logging capabilities, particularly with rosbag, are vital for data analysis, debugging, and scenario replication.
3. **ROS Capabilities:** ROS’s control capabilities ensure precise robot

## 1. Introduction

---

movements and actions, encompassing real-time control loops and motion planning for complex task execution. Its perception capabilities, processing data from various sensors, are critical for tasks like navigation and environment mapping. ROS's manipulation features enable physical interaction with the environment, essential in applications such as assembly and logistics. Furthermore, coordinate transformations, managed by the tf library, maintain spatial relationships crucial for path planning and object manipulation.

ROS 2 stands as a foundational element in the realm of modern robotics, playing a pivotal role in this thesis focused on offensive robot cybersecurity. As the *de facto* standard for modern robotic software development, ROS offers a comprehensive framework that not only streamlines the creation of complex robotic applications but also inherently shapes the cybersecurity landscape of robotic systems. In the context of offensive cybersecurity, the ubiquity and modular architecture of ROS present both opportunities and challenges. Its widespread adoption makes it a critical target for cybersecurity research, as vulnerabilities within ROS (including any of its higher level abstraction, or underlayers) could potentially impact a vast array of robotic systems. This thesis, therefore, places significant emphasis on ROS, examining its architectures, communication protocols, and tools through the lens of offensive cybersecurity. By doing so, it aims to uncover and address potential security weaknesses, contributing to the robustness and resilience of robotics systems developed with ROS. This exploration not only aids in fortifying ROS-based robots against cyber threats but also provides valuable insights into the broader field of robot cybersecurity.

### 1.3 Research Objectives and Approach

Security is not a product, but a process that needs to be continuously assessed in a periodic manner, as systems evolve and new cyber-threats are discovered. This becomes specially relevant with the increasing complexity of such systems [9]. Current robotic systems are of high complexity, a condition that in most cases leads to wide attack surfaces and a variety of potential attack vectors which makes difficult the use of traditional approaches. Altogether, this leads to the following research question: given the increasing complexity of robotic systems, how can we protect them from cyber-threats? Inspired by the popular adage "*the best defense is*

a *good offense*", the present work attempts to answer this question by focusing on offensive practices yet, given the time-consuming requirements of traditional offensive security approaches [10, 11, 12], the answer might not be straightforward and require further automation.

**Research hypothesis 1.** *The inherent complexity of robotic systems, which leads to wide attack surfaces and a variety of potential attack vectors, will be more effectively tackled with an offensive security approach rather than a defensive one.*

Scalability is one of the core problems of offensive security assessments (red teaming, penetration testing, etc.) as of now [13, 14]. This thesis dives into robot cyber security and study how traditional and modern offensive methods apply to robotics, delivering a new angle to protect such systems. To this research question, the investigation herein proposed reviews and creates new mechanisms and techniques that leverage Machine Learning and Game Theory allowing for the full automation of offensive security assessments.

**Research hypothesis 2.** *An autonomous, end-to-end offensive security assessment algorithm that leverages Machine Learning and Game Theory will address scalability problems in offensive security.*

This research also argues and challenges the underlying ethics of the security community when applied to robotics. Particularly, *de facto* standards and conventions on security outcomes and disclosures are assessed and challenged, with a critical non-vendor-centered attitude. Ultimately, given the novelty of the topic itself, a relevant number of contributions are expected in the form of blueprints that will empower future research.

In the remainder of this chapter, the research objectives and approach are discussed in more detail. The overall research framework is introduced in Section 1.3, followed by an explanation of each one of the building blocks.

### Security, cybersecurity and cyber security

Within this thesis, the terms security, cybersecurity and cyber security are used interchangeably.

## 1. Introduction

---

Although security has gained attention over the past years, our computational systems are increasingly vulnerable against cyber attacks. This includes of course all robots. As indicated by Bozic [9], one reason behind is definitely the system’s complexity, comprising interacting heterogeneous systems. Complexity is the enemy of security. It is very difficult to assure no vulnerabilities in a system that an attacker could exploit. In robotics, the reality is even more concerning, as there is no culture of security<sup>3</sup>. As studied in previous work [2, 15], except few vendors and selected actions, no real concern has been shown. Offensive security methods (such as pentesting or red teaming) play an important part in the security lifecycle. In these engagements, as introduced by Applebaum [14], (offensive) security teams try to break into an organization’s assets, identifying vulnerabilities along the way. Red teams take this concept even further, trying to fully emulate what real adversaries do: instead of just compromising the network and identifying vulnerabilities, they have a larger goal that requires significant post-compromise work. Offensive security methods are necessary to understand attackers’ behavior, to train defensive mechanisms and ultimately, to help protect existing systems by discovering flaws first. The latter is specially important in highly complex systems, such as robots.

In an attempt to raise awareness and create some base formal studies and tools, this research objective is to study how offensive cyber security methods applied to robotics allow to protect such systems. In particular, this thesis aims to study how traditional methods in offensive security are applied to robotics. Such methods include penetration testing, red teaming, automated attack planning or advanced persistent threats amongst others. The expected outcome is twofold: a) to validate that these practices are applicable to robotics and b) that the inherent complexity of robotics systems, often systems of system at multiple levels, make the manual application of these methods unfeasible in most robotic setups. In addition, this study covers ethics and defensive security mechanisms that protect systems, networks and operations. Such is necessary for implementing offensive methods and understanding the underlying ethical and legal boundaries. Ultimately, this study looks into how to automate

---

<sup>3</sup>Early robot languages such as ABB’s *Rapid*, KUKA’s *KRL* or Fanuc’s *Karel* did not consider security primitives. Similarly, later robotic frameworks born from academic efforts (e.g. ROS[4]) decided to ignore and forward security to third parties.

and optimize offensive cyber security attacks targeting robots. Preliminary ideas on this direction include the use and extension of prior work (as covered in 2.4) on leveraging Machine Learning for the semi-automation of offensive security assessment. Such machine learning techniques could be trained using popular and widely available Capture The Flag (CTF) environments adapted for robotics. In addition, to optimize the use of resources and define proper learning and exploitation strategies, Game Theoretic approaches are considered.

For coherence, this proposal has been organized in a framework consisting on four building blocks. As depicted in Figure 1.2, from bottom to top, (0) Ethics, (1) Blueprints for researching offensive security for robots, as well as robotics in general, (2) Cyber-attacking robots and (3) Cyber-attacking robots with Machine Learning and Game Theory. Generally, upper layers require of primitives from the underlying ones. The following subsections briefly describe each block.

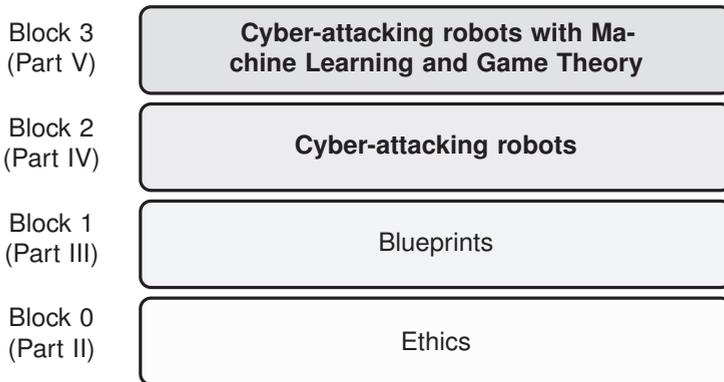


Figure 1.2: Thesis building blocks

### 1.3.1 Ethics

This block treated in Part II aims to study the ethical landscape of cybersecurity in robotics. Particularly, it aims to build some basic knowledge on the ethically accepted situation for robotics in Europe, in main areas of application. Beyond analysing existing vulnerabilities and manufacturer responses to them, this layer also studies robotic users'

perception of cybersecurity by conducting various surveys in popular robotic communities. In addition, this block challenges some of the *de facto* standards in security when applied to robotics and discuss them with a critical attitude. Including topics such as disclosure policies, the requirement from vendors to freely notify research results and vulnerabilities (so called *responsible disclosure*) or even, the relevance of disclosing flaws at all, inspired by previous criticism [16]. This block is discussed in more detail in Chapter 3.

### 1.3.2 Blueprints

Building upon the previous ethical and legal baselines, this block is covered in Part III and develops the tools, libraries and frameworks necessary to perform robotics security research and offensive attacks in robots and their components. Each one of these research elements will be considered as "*blueprints*". Often considered part of a research continuum, the blueprints produced in this phase include:

- Building upon the DevSecOps in Robotics principles researched in the non peer-reviewed published article [17], **SROS2** proposes a series of developer tools, meant to be usable and that facilitate adding security capabilities to ROS 2 computational graphs (Chapter 4). On top of SROS2, we also discuss the shortcomings of the DDS Security extensions [18] which ROS 2 embraces and how additional layers of security via VPN could help mitigate them.
- Security measures such as those proposed by SROS2 in Chapter 4 add a computational overhead to robotic systems. In order to accurately estimate the performance impact, **RobotPerf** proposes an open-source and community-driven benchmarking tool designed to assess the performance of robotic computing systems in a standardized, architecture-neutral, and reproducible way, including the security associated aspects of them (Chapter 5).
- Chapter 5 elucidates the computational overhead of integrating security into robotic communications, highlighting the resultant time bottlenecks that diminish system capabilities. To address this, **RobotCore** is introduced. It aims to empower the development of hardware accelerators that expedite security computations in

robotics, thereby enabling the incorporation of security in robotic interactions without compromising capabilities. RobotCore introduces a framework and an *open architecture for hardware acceleration* that extends ROS to easily develop hardware accelerators in an accelerator-agnostic manner, across GPUs and/or FPGAs (Chapter 6).

### 1.3.3 Cyber-attacking robots

Using previous blueprint security primitives, tools developed (such as Alurity [19] or ExploitFlow [20]) and know-how, research produced within this block aims at building basic attacks on robots and robot components targeting both, software and hardware. These attacks' objectives are threefold: first, they aim to contribute building some basic research ground on offensive cyber security robot practices. Second, these attacks will help raise the awareness of the current practices most vendors are applying. Third and ultimately, these actions help building expertise and develop an intuition on common offensive cyber-behavior, which will later be necessary when automating attacks using Machine Learning and Game Theory.

In Part IV, we present our contributions while cyber-attacking robots. Chapter 7 cares about attacking robot software and investigates the Data Distribution Service (DDS) standard, crucial for robotic operation, uncovering numerous vulnerabilities as detailed in [21]. We highlight the discovery of publicly accessible DDS implementations, stressing the importance of persistent security evaluation and robust integration practices. Chapter 8 explores attacking robot hardware through teardowns, emphasizing the significance of such methods in identifying security flaws. We also address the effects of planned obsolescence in robotics, advocating for a 'Right to Repair' stance. The last chapter, 9 focuses on attacking robots in industry and introduces Akerbeltz ransomware, the first instance of ransomware targeting industrial collaborative robots. Akerbeltz demonstrates its impact on industrial robots by encrypting and locking robotic systems, which underscores the critical need for heightened security against potential cyber-attacks.

### 1.3.4 Automated offensive robot cybersecurity

Bridging the gap between the intricate world of robotic security vulnerabilities and proactive countermeasures, this final block shifts the focus

from identification to automated action. It lays the groundwork for systems designed to anticipate and combat threats autonomously, fortifying defenses in an ever-evolving digital battleground where automation play an increasingly pivotal role. This transition encapsulates the move from understanding vulnerabilities and its mitigations (covered in parts III and IV), to implementing advanced strategies for robust and resilient cybersecurity cognitive engines. Constructing upon all previous blocks, in Part V we attempt to design and deploy an autonomous system for cyber-attacking robots using Machine Learning and Game Theory.

Building on the strategic insights provided by game theory, we advance into the domain of proactive cyber defense and offense within automated systems in Chapter 10. This first chapter in Part V tackles the challenge of *Determining the attack target with Game Theory*, equipping us with a strategic toolset to predict the most vulnerable points in our robot threat landscape and to fortify them accordingly. This chapter describes how to prioritize our defensive efforts and to anticipate the moves of our adversaries, aligning our future actions with the most likely paths they might traverse in the adversary's illicit endeavors.

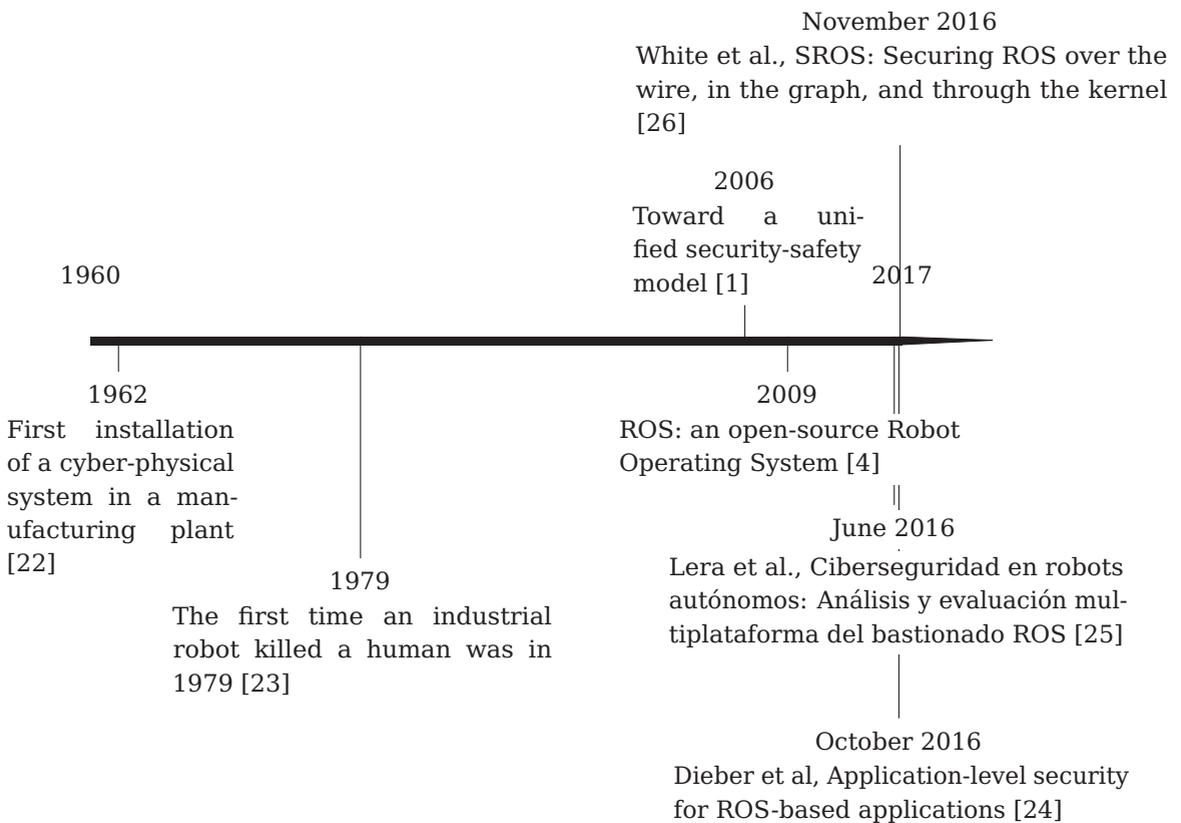
Following this strategic planning phase, *Guiding attacks with Machine Learning* presented in Chapter 11, immerses us into the dynamic process of directing our cyber agents. Here, we leverage the advanced capabilities of modern Machine Learning, particularly the generative power of Large Language Models, to automate the decision-making processes in cyber-attacks. These models act as multi-faceted instruments, capable of parsing vast amounts of data, reasoning through complex scenarios, and generating adaptive attack and defense strategies. With the prototype PentestGPT, we have a glimpse into a future where cybersecurity operations are autonomous, intelligent, and ever-evolving, reflecting the intricate nature of the digital battleground where they operate.

Part V as a whole forms a comprehensive narrative that transitions from a state of awareness and understanding of vulnerabilities to an active, autonomous and intelligent stance against them. They pave the way for automated systems that do not merely react to threats but anticipate and adapt to them, marking a significant leap in the field of offensive robot cybersecurity.

# Chapter 2

## Background context

### 2.1 Biographical cornerstones in robot cybersecurity



## 2. Background context

---

Arguably, the first installation of a cyber-physical system in a manufacturing plant was back in 1962 [22]. The first human death caused by a robotic system is traced back to 1979 [23] and the causes were safety-related according to the reports. From this point on, a series of actions involving agencies and corporations triggered to protect humans and environments from this machines, leading into safety standards.

Security however hasn't started being addressed scientifically in robotics until recently. One of the first peer-reviewed published articles on the topic goes back to 2016 [25] which already warns about the security dangers of the Robot Operating System (ROS) [4]. Following this first assessment, the same group in Spain authored a series of articles touching into robot security [27, 28, 29, 30, 31]. Contemporary to Lera's work, White et al. propose an addition to ROS API to support modern cryptography and security measures [26]. This work led by White (UC San Diego, Spain) and Caiazza (Ca' Foscari University, Italy) delivered a series of articles [32, 33, 34, 35, 36, 37] which focused on proposing defensive blueprints for robot cybersecurity. Starting also in 2016 [24], Dieber (Joanneum Research, Austria) led a series of publications that researched robot cybersecurity focusing around ROS [38, 39, 40, 41, 42].

A careful review of the prior art in the last paragraph indicates that there're three groups, in different points of the world, that almost simultaneously started working in robot cybersecurity by tackling the lack of security primitives and proposing studies and blueprints for securitization. Most of this existing work focuses on defensive approaches and mechanisms. Very few publications touched into offensive approaches to secure robotic systems being [43] one of the first. Some pioneering work on the offensive side focused onto generating primary concerns and social awareness on robot security faults [44, 45] but from this literature review, very few instances of offensive security have been seen so far in robotics.

The following sections shed some additional light into the foundations of this research.

## 2.2 Robotics software quality, safety and security

**Quality** (Quality Assurance or QA for short) and **Security** are often misunderstood when it comes to software. Ivers argues [46] that quality "essentially means that the software will execute according to its design and purpose" while "security means that the software will not put data or computing systems at risk of unauthorized access". Within [46] one relevant question that arises is whether the quality problems are also security issues or vice versa. Ivers indicates that quality bugs can turn into security ones provided they're exploitable, and addresses the question by remarking that quality and security are critical components to a broader notion: software integrity. Depicted in Figure 2.1, software integrity can be represented the union of both software security and quality (**Software Integrity** = Security  $\cup$  Quality).

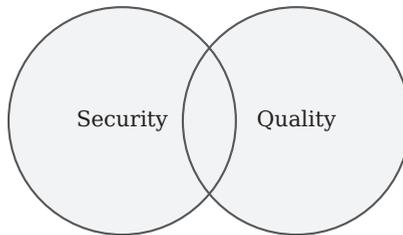


Figure 2.1: Software integrity as the union between security and quality

Coming from the same group, Vamosi [47] argues that "quality code may not always be secure, but secure code must always be quality code". This somehow conflicts with the previous view and leads one to think that secure software is a subset of quality. The author of this thesis rejects this view and argues instead that Quality and Security remain two separate properties of software that may intersect on certain aspects (e.g. testing) as depicted in Figure 2.2.

While the target of this thesis is Security, Quality is also studied given its intersection. Often, both secure and quality code share several requirements and mechanisms to assess them. This includes testing approaches such as static code testing, dynamic testing, fuzz testing or software component analysis (SCA) among others.

## 2. Background context

---

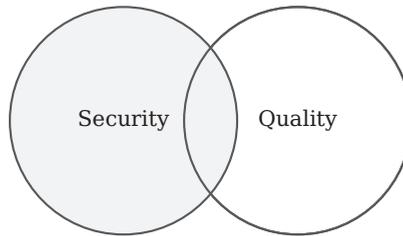


Figure 2.2: Security as the target of this thesis.

In robotics there is a clear separation between Security and Quality that is best understood with scenarios involving robotic software components. For example, if one was building an industrial Autonomous Guided Vehicle (AGV) or a self-driving car, often, she/he would need to comply with coding standards (e.g. MISRA [48] for developing safety-critical systems). The same system's communications, however, regardless of its compliance with the coding standards, might rely on a channel that does not provide encryption or authentication and is thereby subject to eavesdropping and man-in-the-middle attacks. Security would be a strong driver in here and as remarked by Vamosi [46], "neither security nor quality would be mutually exclusive, there will be elements of both".

Quality in robotics, still on its early stages [49], is often viewed as a pre-condition for **Safety**-critical systems. Similarly, as argued by several, safety can't be guaranteed without security [50, 51]. Coding standards such as MISRA C have been extended [52, 53] to become the C coding standard of choice for the automotive industry and for all industries developing embedded systems that are safety-critical and/or security-critical [51]. As introduced by ISO/IEC TS 17961:2013 "in practice, security-critical and safety-critical code have the same requirements". This statement is somehow supported by Goertzel [50] but emphasized the importance of software remaining dependable under extraordinary conditions and the interconnection between safety and security in software. This same argument was later extended by Bagnara [51] who acknowledges that having embedded systems non-isolated anymore plays a key role in the relationship between safety and security. According to Bagnara, "while safety and security are distinct concepts, when it comes to connected

software" (non-isolated) "not having one implies not having the other", referring to integrity.

In the opinion of the author of this dissertation, coding standards such as MISRA or ISO/IEC TS 17961:2013 for safety-critical and security-critical software components do not guarantee that the final robotic system will be secure and thereby, safe. As illustrated in the example above, robotics involves a relevant degree of system integration and inter-connectivity (non-isolated embedded systems connected together internally and potentially, externally as well). As such, both secure and ultimately safe robotics systems do not only need to ensure quality by complying against coding standards but also guarantee that they aren't exploitable by malicious attackers.

In the traditional view of *system security*, safety is often understood as "nothing bad happens naturally" while security intuitively indicates that "nothing bad happens intentionally". Acknowledging the acceptance of this view in the security community, this thesis puts special focus in the context of robotics. To further understand terminology and prior art in a robotics context, Table 2.1 presents a summary of the concepts discussed with their interpretation applied to robotics and the corresponding sources used:

Security, as understood in Table 2.1 shares Integrity with Safety. As discussed in [50, 51], "the only thing that distinguishes the role of integrity in safety and security is the notion of *exceptional condition*. This reflects the fact that exception conditions are perceived as accidental (safety hazards) or intentional (security threats)". The latter, security threats, are always connected to vulnerabilities. A vulnerability is a mistake in software or hardware that can be directly used by an arbitrary malicious actor or actress to gain access to a system or network, operating it into an undesirable manner [54]. In robotics, security flaws such as vulnerabilities are of special relevance given the physical connection to the world that these systems imply. As discussed in [2], "*Safety cares about the possible damage a robot may cause in its environment, whilst security aims at ensuring that the environment does not disturb the robot operation. Safety and security are connected matters. A security-first approach is now considered as a prerequisite to ensure safe operations*". Figure 2.3 depicts the concepts of Safety, Quality and Security representing their relationships. In particular,

## 2. Background context

---

Concept	Interpretation	Reference/s
Safety	Safety cares about the possible damage a robot may cause in its environment. Commonly used taxonomies define it as the union of integrity and the absence of hazards (Safety = Integrity + Absence of catastrophic consequences)	[2, 50, 51]
Security	Security aims at ensuring that the environment does not disturb the robot operation, also understood as that the robot will not put its data, actuators or computing systems at risk of unauthorized access. This is often summarized as Security = Confidentiality + Integrity + Availability.	[2, 46, 50, 51]
Quality	Quality means that the robot's software will execute according to its design and purpose	[46]
Integrity	Integrity can be described as the absence of improper (i.e., out-of-spec) system (or data) alterations under normal and exceptional conditions	[51]

Table 2.1: Summary of security-related terms

safety as a super-set of security and quality.

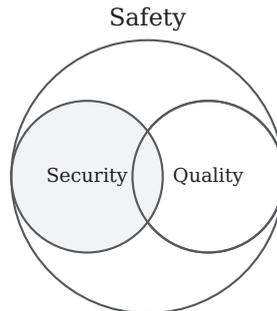


Figure 2.3: Safety, quality and security relationship

This thesis focuses on security, however, its relationship with quality and safety must be noted. Security intersects quality in the sense that some methods are shared between both (e.g. testing). Moreover, as discussed, a safe system demands first security and quality.

Robot security vulnerabilities are potential attack points in robotic systems that can lead not only to considerable losses of data but also to safety

incidents involving humans. Some authors [55] claim that unresolved vulnerabilities are the main cause of loss in cyber incidents. The mitigation and patching of vulnerabilities has been an active area of research [56, 57, 58, 59, 60, 61] in computer science and other technological domains. Unfortunately, even with robotics being an interdisciplinary field composed from a set of heterogeneous disciplines (including computer science), to the best of the knowledge of this thesis' author and his literature review, not much vulnerability mitigation research related to robotics has been presented so far.

## 2.3 Cybersecurity Across Various Domains: IT, OT, IoT, and Robotics

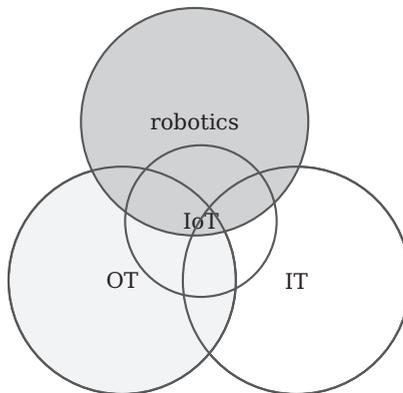


Figure 2.4: Relationship between various technology domains

Cybersecurity is fundamentally about ensuring systems are protected from digital dangers and threats. This concept, while universally applicable, varies in interpretation and implementation across different technological domains such as Information Technology (IT), Operational Technology (OT), Internet of Things (IoT), and robotics. Each domain, due to its unique technological architecture, necessitates a distinct approach to cybersecurity.

### 2.3.1 Definitions and Differentiations

In the realm of digital security, various terms have evolved to denote domain-specific aspects:

**Information Technology (IT):** Involves the use of computers for data management and is typically employed organization-wide [62].

**Operational Technology (OT):** Focuses on managing industrial operations through monitoring and control of devices and processes [63].

**Industrial Control Systems (ICS):** A crucial segment of OT that includes systems used to oversee industrial processes.

**Internet of Things (IoT):** Represents the extension of internet connectivity to various sensors and devices.

**Industrial IoT (IIoT):** The application of IoT technologies in industrial settings.

**Robotics:** An interdisciplinary domain that integrates sensors, actuators, and computation to create autonomous systems.

The application of cybersecurity varies across IT, OT, IoT, IIoT, and robotics. Each technology, while subject to the overarching goal of operational security, requires a tailored approach. For instance, ICS, evolving from isolated systems to more IT-like structures, have increased vulnerability to cyber threats due to increased connectivity [64]. The confluence of IT and OT in IoT necessitates a unique set of security skills, distinct from those required in traditional IT or OT environments [65]. Robotics, due to its inherent complexity and the integration of various systems, demands domain-specific knowledge and a specialized approach to cybersecurity.

Table 2.2 presents a comparative analysis (extended from [63, 64]) which reveals distinct requirements and challenges in implementing cybersecurity across these technologies. Notably, robotics stands out due to its critical real-time requirements and safety implications, underscoring the need for a comprehensive and dedicated cybersecurity strategy tailored to robotic systems. Such approach should emphasize the relevance of isochronous and deterministic communications, crucial in robotic systems.

Security topic	IT	OT (ICS)	I(D)oT	Robotics
<b>Antivirus</b>	Widely used, easily updated	Complicated, network detection and prevention solutions mostly	Complicated, technology fragmentation	Complicated, complex, few solutions, network monitoring insufficient
<b>Life cycle</b>	3-5 years	10-20 years	5-10 years	10+ years
<b>Awareness</b>	Decent	Poor	Poor	Very poor
<b>Patch management</b>	Often	Rare, manufacturer approval needed	Rare, end-user permission/action often needed	Very rare, complex setups
<b>Change Management</b>	Regular and scheduled	Rare	Rare	Very rare, specialized technicians
<b>Evaluation of log files</b>	Established practice	Unusual	Unusual	Non-established
<b>Time dependency</b>	Delays accepted	Critical	Some delays accepted	Critical
<b>Availability</b>	Not always, failures accepted	24*7	Some failures accepted	24*7 available
<b>Integrity</b>	Failures accepted	Critical	Some failures accepted	Critical
<b>Confidentiality</b>	Critical	Relevant	Important	Important
<b>Safety</b>	Not relevant	Relevant	Not relevant	Critical
<b>Security tests</b>	Widespread	Rare and problematic	Rare	Mostly not present
<b>Testing environment</b>	Available	Rarely available	Rarely available	Rare, difficult to reproduce
<b>Determinism requirements</b>	Non-real-time, high delay/jitter acceptable	Hard real-time, high delay/jitter not acceptable	Often non-real-time, soft/firm real-time in some environments	Hard real-time for safety critical, firm/soft for others

Table 2.2: Comparative analysis of cybersecurity across different technologies

### 2.3.2 Misunderstandings in Robot Security

A prevalent issue in the robotics industry is the conflation of IT security with comprehensive robotics security. Many manufacturers, such as Mobile Industrial Robots (MiR) as depicted in Figure 2.5, focus predominantly on IT security, neglecting the broader spectrum of threats specific to robotics [66]. This oversight can lead to severe safety hazards, as compromised robots pose risks not just to data privacy but also to human safety and environmental integrity. Therefore, it is imperative for manufacturers to adopt a holistic view of security that encompasses all aspects unique to robotic systems.

Given the safety implications of autonomous systems, and given the

## 2. Background context

---

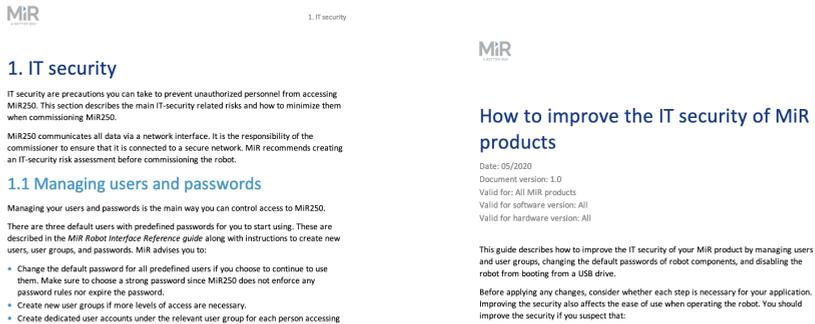


Figure 2.5: Captures from Mobile Industrial Robots (MiR) documentation indicating a purely IT security approach.

security precondition of safety, the need for robust cybersecurity measures in robotics surpasses that in IT, OT, or IoT. The complexity and integration of systems in robotics demand a security approach that is both comprehensive and specific to the domain, with careful consideration for robot's real-time nature. This thesis advocates for the implementation of such dedicated security measures in the field of robotics.

## 2.4 Machine Learning and Game Theoretic approaches to offensive cybersecurity

Offensive security typically involves a malicious attacker performing a series of steps, often connected, and usually requiring a high degree of expertise or knowledge on the target system. Within offensive security practices, Penetration Testing (*pentesting*) is an active method for assessing and evaluating the security of digital assets by planning, generating and executing all possible attacks that can exploit existing vulnerabilities. Figure 2.6 pictures the pentesting pipeline for a robotics system inspired on prior work [67, 68, 69, 70].

As described by several authors [10, 11, 12], current pentesting practices are becoming repetitive, complex and resource consuming despite the use of automated tools. Moreover, they generally require a high level of expertise in those systems targeted. Applebaum et al. [13, 14] also underline the

## Machine Learning and Game Theoretic approaches to offensive cybersecurity

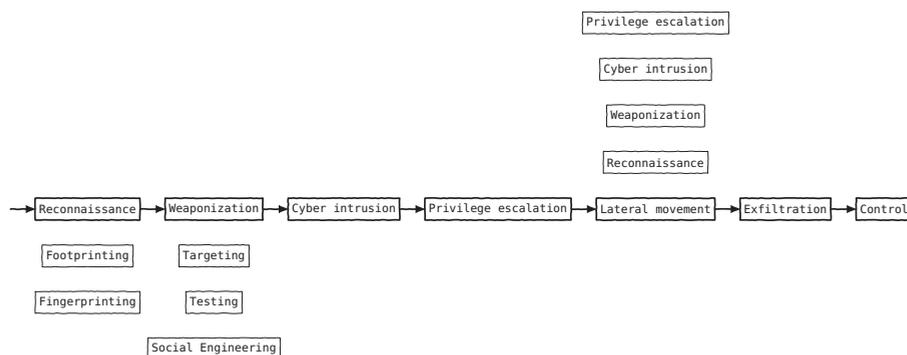


Figure 2.6: Attacker model for robotics. Adapted from [67, 68, 69, 70]

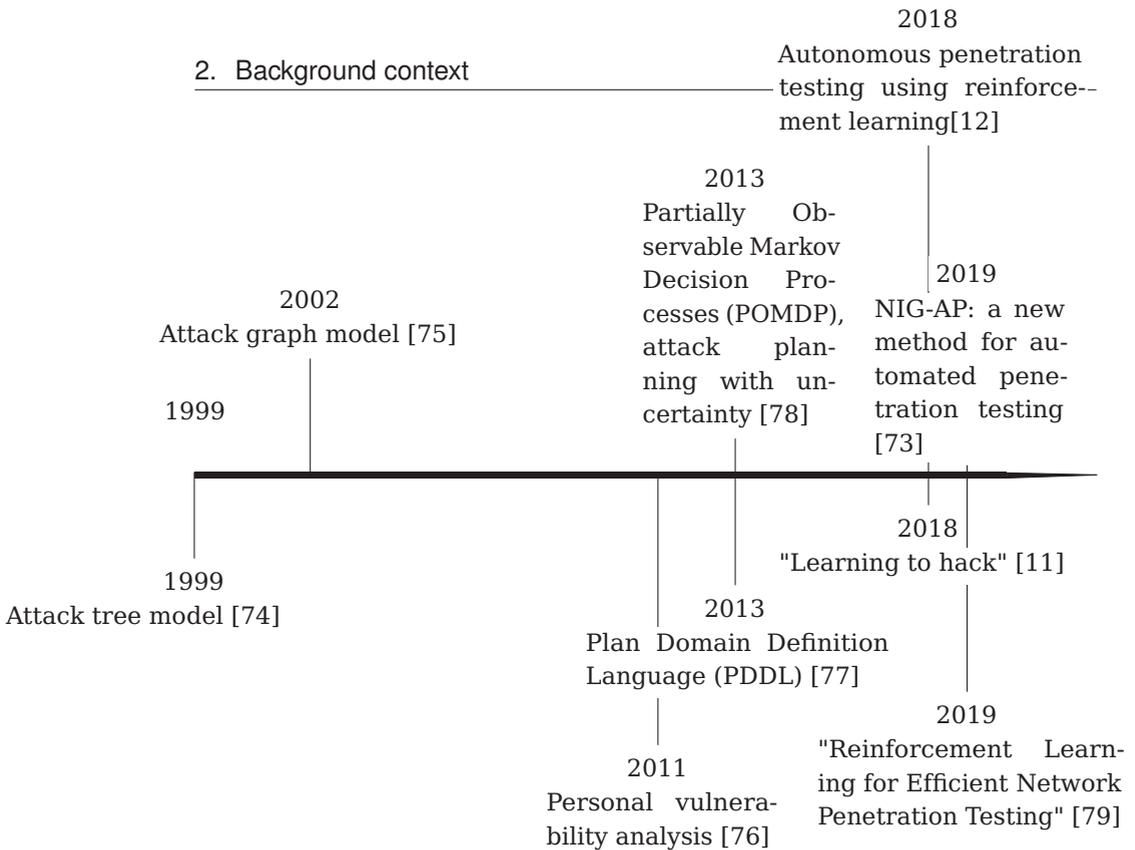
importance of pentesting in the security lifecycle and the shortcomings associated with manual execution. These challenges have made several researchers to turn into Machine Learning (ML) to try and automate the process of pentesting or other security assessments <sup>1</sup>.

While several articles describe the growing popularity of the use of machine learning in defensive approaches for security, [11, 71, 72] already report that very few sources cover the use of Machine Learning for offensive approaches to security. While reviewing literature, the author of this thesis found it extremely helpful to step back a few decades to review previous work that attempted to automate pentesting activities using Artificial Intelligence (AI) approaches. A complete and detailed walkthrough on these past efforts is beyond the scope of this thesis but nicely compiled at [73]. For completeness and reader's convenience, the most relevant events on this regard prior to the start of this thesis (2020) have been pictured in the diagram below:

---

<sup>1</sup>Security assessments are often misunderstood and pentesting is used to group them all. For the context of this thesis, penetration testing assessment refers to the activity that aims to find as many vulnerabilities and configuration issues as possible in the time allotted, and exploiting those vulnerabilities to determine the risk of the vulnerability. Red team assessment aims to test the organization's detection and response capabilities against a targeted objective.

## 2. Background context



Moving past AI and diving more contemporary ML approaches <sup>2</sup>, in 2018 Niculae [11] presents one of the first pieces available where ML, in particular, Reinforcement Learning (RL) is used to tackle the pentesting assessment task while modeling it as a game. Niculae evaluates different RL techniques, namely Q-Learning (QL), Extended Classifier Systems (XCS) or Deep Q-Networks (DQN) while comparing it with Human performance and multiple fixed-strategies including Random, Greedy and a Finite State Machine. During his work and for his particular simplified model, Niculae reports building an agent that can learn to penetrate a network comparable in strength to a human, paving the way for future work in the application of RL in pentesting.

---

<sup>2</sup>The author of this thesis assumes the reader has some basic understanding of the differences and relationships between "artificial intelligence", "machine learning" and "deep learning".

Also in 2018 (though published later), Schwartz [12] argues that current approaches to automated pentesting have relied on methods which require a model of the exploit outcomes, however the cyber security landscape is rapidly changing as new software and attack vectors are developed which makes producing and maintaining up-to-date models a challenge. Instead and similar to Niculae, Schwartz advocates for the application of RL and evaluates QL and DQL obtaining positive results. Interestingly however, Schwartz remarks that the implemented algorithms in the simulated scenarios were only practical for smaller networks and numbers of actions and would not be able to scale to truly large networks.

The most recent studies available on this domain are [79] and [73]. The first one explores the design and development of an ML-based pentesting system while supported by human experts. The article itself provides an interesting motivation for robotics, but does not provide a clear description of the underlying RL techniques, the reward function, or a detailed reasoning on how the state space and the action spaces of the pentesting activity were captured. Moreover, the authors conclude their paper highlighting that the main drawback of their contribution is the need of a high-caliber human expert supervising the initial training. Zhou et al [73] present a new method for pentesting using RL. Particularly, they propose an attack planning algorithm based on RL to automatically discover attack paths without prior knowledge of the scenario network. Opposed to [79], this last article provides a good review of the pentesting foundations, underlying math formalizing pentesting as a Markov decision process and a detailed description of the assumptions made for representing the problem in reinforcement learning.

Despite the advances on ML, scalability remains a problem for automated pentesting across network layers, even when powered with modern techniques such as the ones described above. Scalability can be understood easily by switching across (attack) subjects and/or determining next steps or actions during a pentesting offensive effort. It is unclear whether the current trend of ML research (mostly driven by increasingly complex RL constructs) can help lead towards an *end-to-end ML-driven penetration testing*. The research conducted herein should tap into this problem and propose novel approaches. Beyond the use of ML for addressing the scalability problem in pentesting, another interesting issue is selecting the right attack targets. More specifically, selecting which attack vectors (corresponding to individual targets) should be exploited first. When conducting

## 2. Background context

---

an offensive pentesting exercise, commonly, multiple targets appear over the exercise. For each attack target, multiple attack vectors are often identified. Selecting which attack vector to prioritize first can be somewhat derived by using scoring metrics, like the Common Vulnerability Scoring System (CVSS) [80]. However, in practice, pentesters use this metric as yet one more indicator to build an intuition on *where to attack*. Developing this intuition is complex and not fully understood. This thesis will also look at this problem by looking at Game Theory (GT). Computational game theory [81, 82] can also help tackle the scalability problem and schedule resources while taking into account the importance of different targets, the responses of the adversaries to the security posture and the potential uncertainties in adversary payoffs and observations. The combination of both, machine learning and game theoretic approaches might eventually lead to results that empower autonomous and versatile red team assessments, first envisioned by Applebaum [13] back in 2016.

A deeper study of the biographical cornerstones for AI in pentesting will be conducted in Chapter 11. To the best of this author's knowledge and extend of literature review, at the time of the start of this research (2020), no single article covered the use of ML and GT methods to secure robots via an offensive approach<sup>3</sup>. This presents an interesting opportunity which is explored throughout this research.

---

<sup>3</sup>Which means, employing a 'Hack-First' approach that is effective and scales by leveraging ML and GT

Part II

# **Ethics**



## Chapter 3

# Ethical Landscape in Robot Cybersecurity

**Contributions:** *Sole author and contributor of [83]. All results are publicly available.*

---

### 3.1 Introduction

This chapter, lying within the larger framework of ethical considerations in cybersecurity for robotics, delves into the complex landscape of this evolving field. The objective is multifaceted: to map out the current ethical standards within the robot cybersecurity domain, dissecting how these standards manifest in key application areas. This exploration goes beyond merely cataloging vulnerabilities and manufacturers' responses; it also seeks to understand the perceptions and attitudes of robotic users and vendors toward cybersecurity. This understanding is garnered through surveys distributed across popular robotics communities, thus capturing a wide range of insights and sentiments.

An important aspect of this chapter is its critical examination of the established norms and standards in robot cybersecurity. It challenges and scrutinizes these norms, particularly focusing on the ethics of disclosure policies. This includes the expectations placed on vendors regarding the notification of research findings and vulnerabilities—a practice commonly known as *responsible disclosure*. Furthermore, the chapter probes into the more controversial facets of this topic, such as the ethical implications of disclosing security flaws at all, drawing inspiration from previous critiques like those of Rescorla (2005) [16].

To achieve a comprehensive understanding of the state of robot cybersecurity, this chapter employs a systematic review approach that

### 3. Ethical Landscape in Robot Cybersecurity

---

unfolds in three distinct phases. Initially, it delves into the existing body of literature, painting a picture of the current cybersecurity landscape in robotics. Following this, the chapter presents findings from surveys conducted within various robotics groups and communities, offering a grassroots perspective on the state of cybersecurity in this field. The final phase of this exploration is anchored in empirical data, drawn from three years of proactive security research in robotics. This phase not only discusses identified vulnerabilities but also reflects on the offensive security exercises undertaken during this period.

The culmination of this chapter is a synthesis of insights and conclusions drawn from these diverse sources, including a discussion of the current ethical standards within the robot cybersecurity domain. It aims to provide actionable guidance on securing robotic systems, understanding their vulnerability to various attack vectors, and strategies to minimize their exposure to potential threats. This chapter thus serves as a critical piece in understanding the ethical underpinnings of cybersecurity in robotics, laying the groundwork for responsible and secure advancements in this dynamic and impactful field.

## 3.2 Background

As introduced before, arguably, the first installation of a cyber-physical system in a manufacturing plant dates back to 1962 [22]. The first human death attributed to a robotic system occurred in 1979 [23], with safety-related issues cited as the cause. This incident sparked a series of actions by agencies and corporations aimed at protecting humans and environments from these machines, leading to the development of safety standards. However, security in robotics has only started to be addressed more recently. Following after McClean's (2013) early assessment, one of the initial published articles on the topic by Lera et al. (2016) warned about the security dangers of the Robot Operating System (ROS) [4, 25]. Subsequently, the same group in Spain authored a series of articles focusing on robot cybersecurity [27, 28, 29, 30, 31]. Around the same time, Dieber et al. (2016) led research into cybersecurity in robotics, proposing defensive blueprints for ROS-based robots [38, 39, 40, 41, 42]. Their work included enhancements to ROS APIs to incorporate modern cryptography and security measures. Concurrent with Dieber et al.'s work, White et

al. (2016) also began producing a series of articles proposing defensive mechanisms for ROS [32, 33, 34, 35, 36, 37].

A little over a year later, starting in 2018, more groups began to contribute to the field. Mayoral-Vilches et al. (2018) initiated a series of security research efforts aimed at defining offensive security blueprints and methodologies in robotics, resulting in various contributions [2, 17, 19, 84, 85, 86, 87, 88]. Notably, this group released a framework for conducting security assessments in robotics [89], a vulnerability scoring mechanism for robots [90], a CTF environment for robotics to train cybersecurity engineers [91], and a robot-specific vulnerability database [15], among other contributions. In 2021, Zhu et al. published a comprehensive introduction to this emerging topic, aiming to foster a sub-community in robotics for cybersecurity [92]. These contributions correspond to the bibliographical milestones in robot cybersecurity presented in Chapter 2.

A careful review of the prior art described in the preceding paragraphs leads to the following observation: based on the literature, robot cybersecurity is still a nascent field that requires further attention, tools, and educational material to train new engineers in security practices for robotics.

### 3.3 Surveying security in robotics communities

During a period of three years (2019 - 2021) various security surveys were conducted in top robotic communities through their corresponding conferences and forums. The following subsections discuss each one of them while attempting to draw some observations:

#### 3.3.1 Surveying the ROS community

Figure 3.1 presents a summarized result of the survey conducted in the ROS community during a period of several months<sup>1</sup>. The survey received a total of 52 responses, which represented the small interest in security at the time. The largest groups of participants are depicted in Figure 3.1b. The most represented group comes from Universities (30%), followed by Software vendors (18%) and Robot manufacturers (14%)<sup>2</sup>. The majority

---

<sup>1</sup>Security survey launched within the ROS Discourse community (announcement, announcement 2, preliminary results).

<sup>2</sup>Others comprise various subgroups, all with less representation than the ones mentioned

### 3. Ethical Landscape in Robot Cybersecurity

---

of the respondents have at last 2 years of experience with ROS and half of them at least 5 (3.1c), most coming from Europe (3.1d). Figure 3.1e present data on security considerations. The data indicates that 73% of the participants think that they have not invested enough to protect their robots from cyber-threats. Coincidentally, the same number of participants indicated that their organizations are open to invest however only 26% acknowledge to actually have invested. This data leads to the following observation:

**Observation 3.3.1.** *There seems to be a gap between the expectations and the actual investment, which suggests that cybersecurity actions in robotics will grow in the future for the ROS community.*

When considering the mitigation strategies applied by respondents as depicted in Figure 3.1f, it's important to highlight that most efforts concentrate on perimeter actions (i.e. firewalls, segmentation and segregation) whereas robot-specific defensive solutions are only applied in a 36% of the cases. Similarly, network assessments and security audits are conducted only in one fourth of the cases (26%) which conflicts with the *de facto* security practices in other industries, wherein assessments are critical to evaluate the resilience of technology.

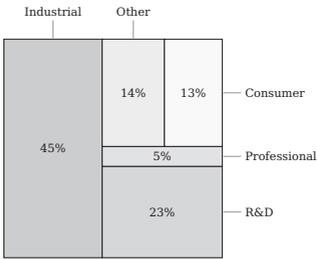
**Observation 3.3.2.** *The lack of robot-specific security measures (36%) and offensive assessments (26%) can be interpreted as an indicator of the maturity level of the technology when compared to other sectors (e.g. IT or OT) where these practices are common and specialized.*

#### 3.3.2 Surveying the PX4 community

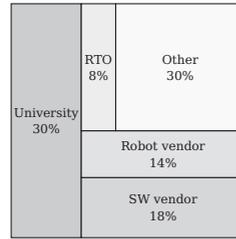
PX4 [93] is an open source flight control software for drones and other unmanned vehicles. Similar to ROS, its community represents another relevant group in robotics. A security survey was conducted in 2020 and the results are summarized in Figure 3.2<sup>3</sup>. Though the PX4 community is significantly smaller than ROS's, the sample size obtained (11 respondents) was extremely small to draw major conclusions. Interestingly though, it was observed that the majority of the respondents have yet to see a security issue impacting the community (3.2d), only 27% had seen it.

---

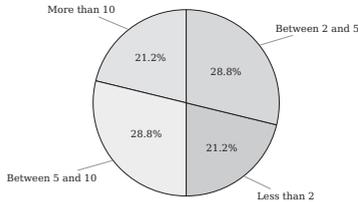
<sup>3</sup>Security survey launched within the PX4 Discourse community [94].



(a) Distribution of respondents by sector of activity. *Sample size 52 respondents.*



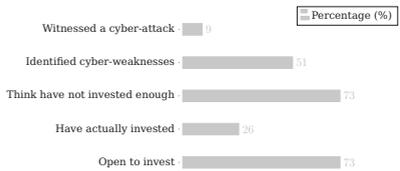
(b) Distribution within the robotics value chain. *Sample size 52 respondents.*



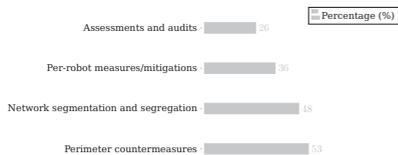
(c) Years of experience with ROS of each respondent. *Sample size 52 respondents.*



(d) Geographical distribution of respondents. *Sample size 52 respondents.*



(e) Percentage favouring each security consideration amongst respondents (in the context of robotics). *Sample size 52 respondents.*



(f) Percentages favouring each mitigation strategy amongst respondents (in the context of robotics). *Sample size 52 respondents.*

Figure 3.1: Surveying the ROS robotics community (2019), from [83]

**Observation 3.3.3.** *Both the PX4 (Figure 3.2d) and the ROS (Figure 3.1e) communities indicated that the majority is yet to witness a cyber-attack. In the ROS community only one out of ten respondents (9%) had seen it whereas in the PX4 group, approximately one out of four (27%).*

The majority of the respondents (81%, figure 3.2e) indicated to be willing to invest and more than 90% confirmed that the amount could be 100 USD or above (Figure 3.2f). This aligns nicely with observation 3.3.1 and further hints that growth should be expected in this field.

#### 3.3.3 Surveying the ROS-Industrial community

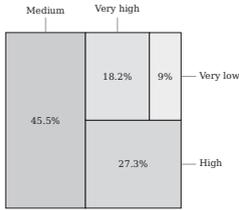
Also in 2020, a series of security-related surveys were launched as part of the European ROS-Industrial Conference, which happens every year in December. Data collected is presented in Figure 3.3<sup>4</sup>. The majority of the respondents (93%) showed awareness about the threats their robots faced and admitted being aware of their exposure to attackers (Figure 3.3b). Unsurprisingly, as a subset of the overall ROS community, the security mitigation actions in the ROS-I community also concentrate on the perimeter which lead to another observation:

**Observation 3.3.4.** *Figures 3.1f and 3.3d confirm that respectively for both ROS and ROS-I groups mitigations concentrate mostly on the perimeter.*

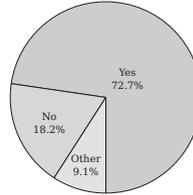
This fact becomes concerning in industrial environments wherein insider threats are as dangerous, and the disruption of ROS could lead to catastrophic consequences for the automation processes [86], impacting more than 5 robots in 44% of the cases according to respondents (Figure 3.3f). The lack of security measures in ROS are particularly concerning since its distributed communication middleware could be easily used to spread malware across connected robots. Such concept was demonstrated by [88], which prototyped an instance of ransomware targeting industrial collaborative robots, leaving these machines and its data completely locked until the corresponding ransom is paid.

---

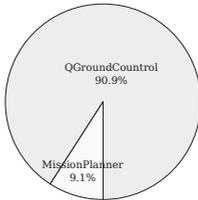
<sup>4</sup>Security surveys launched within the ROS-Industrial community during the digital ROS-I Europe Conference in December 2020.



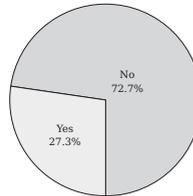
(a) Distribution of respondents to the question: "What's your cybersecurity concern in PX4?". No respondents indicated "Low". *Sample size 11 respondents.*



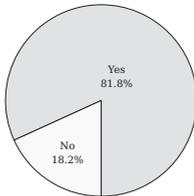
(b) Distribution of respondents to the question: "Does cyber security affect safety?". Other corresponds with user-provided answers. *Sample size 11 respondents.*



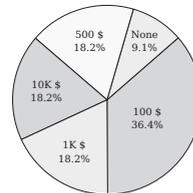
(c) Distribution of respondents to the question: "What's your GCS?". Various additional Ground Control Stations (GCS) were offered but not selected. *Sample size 11 respondents.*



(d) Distribution of respondents to the question: "Have you seen any security issues so far in PX4?". *Sample size 11 respondents.*



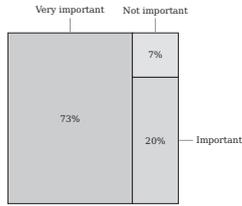
(e) Distribution of respondents to the question: "Would you be willing to invest in cybersecurity for your drone?". *Sample size 11 respondents.*



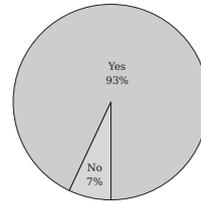
(f) Distribution of respondents to the question: "How much per year is security worth to you?". *Sample size 11 respondents.*

Figure 3.2: Surveying the PX4 robotics community (2020), adapted from [83].

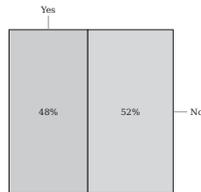
### 3. Ethical Landscape in Robot Cybersecurity



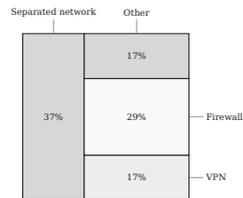
(a) Distribution of respondents to the question: "How important do you think is security for robotics and automation?". *Sample size 30 respondents.*



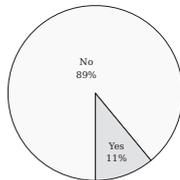
(b) Distribution of respondents to the question: "Do you think your robot can be hacked?". *Sample size 28 respondents.*



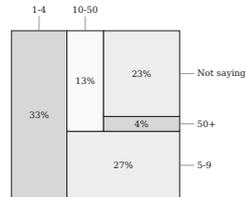
(c) Distribution of respondents to the question: "Have you taken measures to protect your robots?". *Sample size 25 respondents.*



(d) Distribution of respondents to the question: "What measures do you take to secure your robots?". *Sample size 23 respondents.*



(e) Distribution of respondents to the question: "Did you use fuzzing before?". *Sample size 27 respondents.*



(f) Distribution of respondents to the question: "How many robots are you controlling with ROS in your organization?". *Sample size 55 respondents.*

Figure 3.3: Surveying the ROS-Industrial robotics community (2020), adapted from [83].

### 3.3.4 Surveying the European robotics community at the European Robotics Forum (ERF) (2020)

As one of the leading geographies in robotics and cybersecurity, the opinion of European robotics experts was sampled during the annual European Robotics Forum (ERF). Figure 3.4 summarizes the most relevant data collected<sup>5</sup>. The most interesting observation relates to the question "Who is the actor to be responsible for cyber-incidents?":

**Observation 3.3.5.** *In Europe, the majority of the respondents (Figure 3.4b) agree that the responsibility in case of damage as a result of a cyber-incident is to be assumed by the supply chain (86% indicated that it'd sit between System Integrators and robot vendors), with only a 14% pushing the responsibility to the end-user.*

## 3.4 Security research results in robotics

Figure 3.5 depicts summarized vulnerability research results for three vendors: ABB, Mobile Industrial Robots (MiR) and Universal Robots (UR). The data was collected and archived over a multi-year period. Figures 3.5a, 3.5b and 3.5c illustrate the "days until mitigation" for each vulnerability and according to the public data in the Robot Vulnerability Database (RVD) [15]. The flat line represented by a series of data points in figures 3.5b and 3.5c denotes that the vendor hasn't reacted yet to any of these flaws and they remain unmitigated (they are zero days). For ABB robots, the scattered plot in Figure 3.5a denotes more security activity. The following observations are drawn from the data:

**Observation 3.4.1.** *Collaborative robot manufacturers MiR and UR have zero days with an age at least older than one year (figures 3.5b and 3.5c). These flaws continue growing older due to the inactivity from the manufacturers.*

**Observation 3.4.2.** *Vulnerability data affecting ABB robots (Figure 3.5a) shows how according to historical data, vulnerabilities were patched as*

---

<sup>5</sup>Security surveys conducted during the robotics European gathering at the European Robotics Forum (ERF) 2020 in Málaga. The questionnaires were launched during the security sessions.

### 3. Ethical Landscape in Robot Cybersecurity

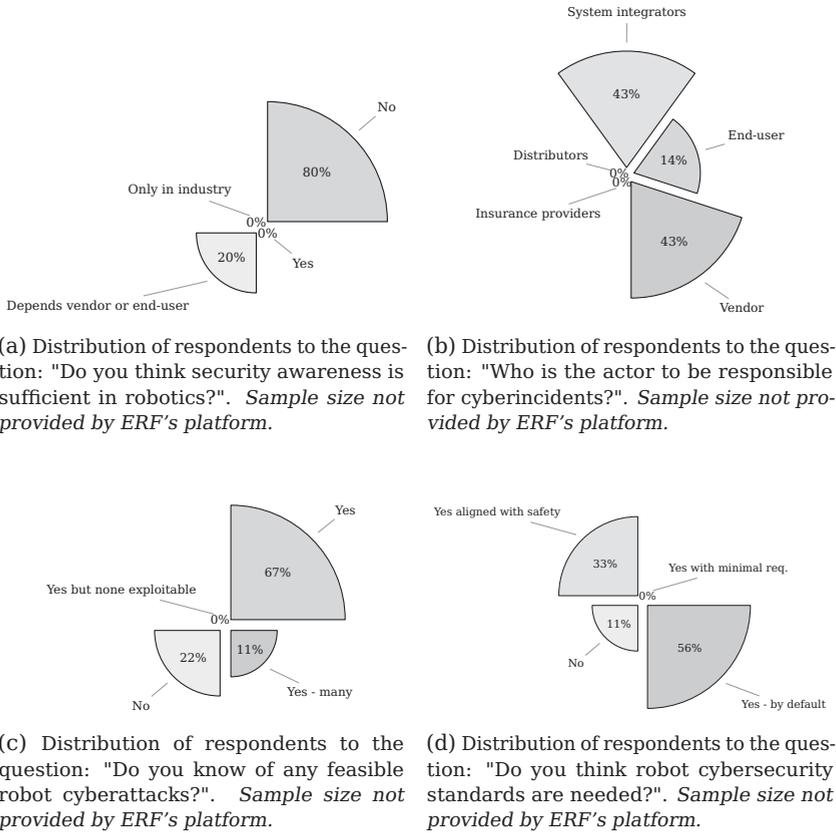


Figure 3.4: Surveying the European robotics community (ERF 2020), adapted from [83].

early as 14 days after its disclosure however the average mitigation time is above four years (1500 days) [15].

On top of these, Figures 3.5d to 3.5i enhance previous data with additional private sources of information and consider vulnerabilities that have yet to reach the public domain. It should be noted that the distribution of vulnerabilities signals the security awareness of the manufacturer. Coherently, figure 3.5g shows how for ABB robots, four out of five vulnerabilities considered have been publicly disclosed, triaged and scored. In contrast, for MiR and UR robots the opposite is observed. Four out of five vulnerabilities have yet to be disclosed publicly.

**Observation 3.4.3.** *The ratio of publicly disclosed vulnerabilities versus the ones remaining private is an indicator when evaluating the security readiness of a robot manufacturer. The threat landscape of a given robot is correlated to this ratio in a direct manner.*

## 3.5 Discussion

### 3.5.1 Ethical Considerations During Surveys

The surveys conducted within various robotics communities were aimed at gathering insights into the state of cybersecurity awareness, practices, and concerns within these communities. While the primary focus was on data collection and analysis, ethical considerations surrounding the conduct of these surveys are essential to address.

1. **Ethical Screening:** Prior to launching the surveys, ethical screening was performed to ensure that the data collection process adhered to established ethical standards. This screening involved a review of the survey content, methodology, and potential risks to participants.
2. **Informed Consent:** Participants were provided with clear and comprehensive information regarding the purpose, storage, and use of their personal data. This information was presented at the beginning or within the announcement of each survey (as examples, see announcement for the ROS and PX4 survey), and participants were required to provide informed consent before proceeding.

### 3. Ethical Landscape in Robot Cybersecurity

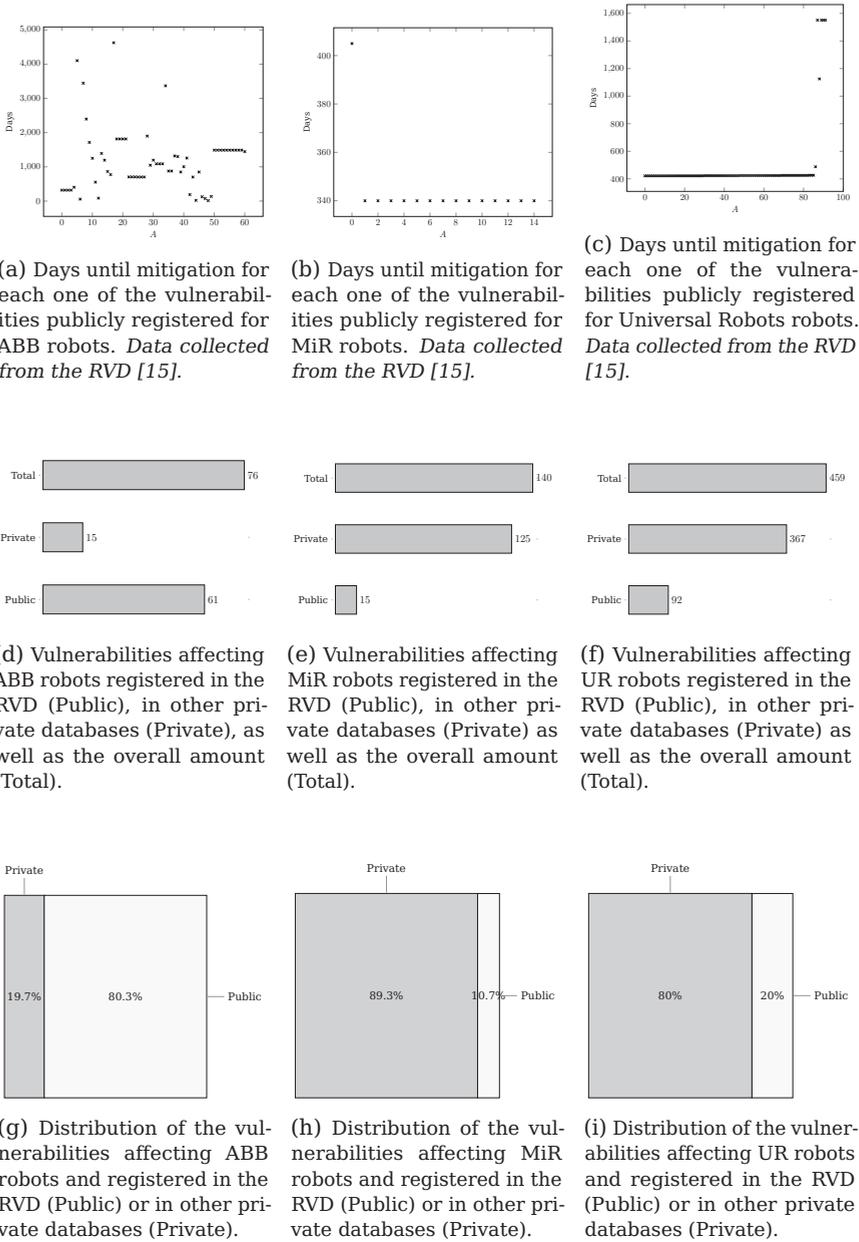


Figure 3.5: Vulnerability data for various robots, adapted from [83].

3. **Confidentiality and Anonymity:** Measures were implemented to ensure the confidentiality and anonymity of participant responses. Personal identifying information was kept separate from survey responses, and only aggregated data was used for analysis and reporting.
4. **Data Storage and Use:** Participant data was securely stored and used solely for the purposes outlined in the survey introduction. Data was accessible only to authorized researchers involved in the study and was not shared with third parties.
5. **Feedback and Accessibility:** Participants were provided with opportunities to provide feedback on the survey process and to access the results of the study upon request. Transparency regarding data handling practices fostered trust and accountability among participants.
6. **Conference Context:** It's important to note that some surveys were conducted within the context of conferences, where participants may have been attendees or members of the respective communities. While this facilitated data collection, it also presented unique challenges in terms of informed consent and data privacy.
7. **Limitations and Future Considerations:** Despite efforts to uphold ethical standards, it's important to acknowledge the limitations of the survey methodology and consider potential improvements for future research endeavors. This includes exploring alternative data collection methods, such as interviews or focus groups, to ensure a more nuanced understanding of participants' perspectives.

In summary, ethical considerations were integral to the design and implementation of the surveys conducted within robotics communities. By prioritizing participant welfare, confidentiality, and transparency, the integrity of the research process was upheld, facilitating meaningful insights into cybersecurity practices and concerns within these communities.

### 3.5.2 Vulnerability Disclosure Policies

Prevailing ethical standards and practices in robot cybersecurity are still on their early stages however, after a few years of research and while

### 3. Ethical Landscape in Robot Cybersecurity

---

interacting with various communities, a few remarks can be made. The subsections below cover some of the common practices observed in robot cybersecurity and discusses their ethical implications.

#### 3.5.2.1 Evolution of Disclosure Policies in Robotics

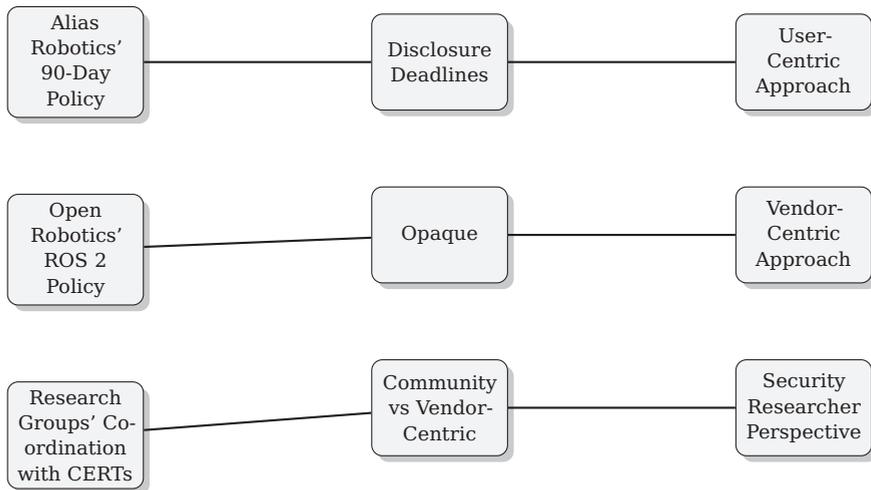


Figure 3.6: Summary of Vulnerability Disclosure Policies in Robotics

The landscape of vulnerability disclosure policies in robotics has undergone notable evolution, particularly influenced by the practices adopted by different organizations and the ethical implications of these choices. This subsection examines three significant contributions in the field, each highlighting different approaches and their resultant impact on the cybersecurity landscape in robotics.

**Alias Robotics' 90-Day Disclosure Deadline** In 2018 and 2019, Alias Robotics pioneered early work on vulnerability disclosure policies in robotics. This was part of their effort to establish the Robot Vulnerability Database (RVD) [15]. This approach adhered to a 90-day disclosure deadline for newly identified vulnerabilities. The rationale behind this timeframe was to balance the urgency for patch development against vendors' capability to respond effectively. The 90-day deadline aimed to

push vendors towards improving their internal processes for vulnerability remediation, which historically had seen delays or even neglect. Alias Robotics' approach was based on the principle that quicker response times to disclosed vulnerabilities would ultimately result in better security for users.

**Open Robotics' ROS 2 Vulnerability Disclosure Policy** Open Robotics, in contrast to Alias Robotics' proactive approach, introduced a ROS 2 Vulnerability Disclosure Policy [95] in 2020, which notably did not incorporate relevant feedback from the Security Working Group [96] and derived from Alias Robotics' initial work. The policy lacked crucial elements like disclosure deadlines, transparent communication, and acknowledgment of the educational value of post-mitigation security research. It diverged from established international security practices, such as assigning CVE IDs, and depended on a non-transparent process for security issue verification. While the policy mentioned a safe harbor, its actual efficiency and effectiveness were questioned. Open Robotics' approach suggested a missed opportunity to prioritize cybersecurity within the ROS community, reflecting a potential lack of commitment to rigorous security standards.

### **Security Research Groups' Direct Coordination with International CERTs**

The ROS 2 Vulnerability Disclosure Policy's limitations led several security research groups to bypass it in favor of direct coordination with international Computer Emergency Response Teams (CERTs). One of such groups included Alias Robotics contributions again and identified relevant vulnerabilities affecting a significant number of ROS robotic endpoints [97] and disclosed them responsibly after meeting reasonable deadlines [98]. The public disclosure got further disseminated in the ROS robotics community [99] (the one owned by Open Robotics) and empowered end-users with knowledge to mitigate outstanding security issues, in contrast to Open Robotics' response, which downplayed the significance of these disclosures and shifted responsibility to third parties <sup>6</sup>. This direct approach with CERTs highlighted a more efficient and impactful way of handling cybersecurity in the robotics ecosystem, as opposed to working with Open Robotics (the acting vendor, as community responsible).

---

<sup>6</sup>Representatives of Open Robotics responded publicly [100] ignoring the relevance of the disclosures and offloading responsibility into third parties (DDS vendors).

#### 3.5.2.2 Ethical Implications of Disclosure Policies

**Necessity of Disclosure Deadlines** The implementation of disclosure deadlines by Alias Robotics was driven by ethical considerations related to the timeliness of vulnerability patching. Slow patch timelines pose a significant risk, as attackers could potentially exploit vulnerabilities before they are patched. The 90-day deadline was designed to incentivize vendors to improve their response times, balancing the need for thorough patch development with the urgency to protect users from potential exploits.

**Transparency and Educational Value of Disclosures** Open Robotics' policy, which lacked transparency and discouraged the disclosure of security research for educational purposes, overlooked the long-term benefits of sharing detailed vulnerability information. Disclosing technical details post-patch offers educational value to the security community, helping to inform and strengthen overall cybersecurity practices. This approach contributes to a more informed and capable pool of security professionals and researchers.

**Community-Centric versus Vendor-Centric Approaches** The different policies reflect a divide between community-centric and vendor-centric approaches to cybersecurity. Alias Robotics' policy, with its emphasis on timely public disclosures, aligns with a community-centric approach that prioritizes user security and education. In contrast, Open Robotics' policy appears more vendor-centric, potentially limiting the flow of critical security information to the wider community and impeding collaborative efforts to address vulnerabilities.

#### 3.5.2.3 Conclusion: The Ethical Dimension of Cybersecurity in Robotics

The exploration of vulnerability disclosure policies in robotics, particularly those of Alias Robotics and Open Robotics, offers a vital perspective on the ethical dimensions of cybersecurity in this domain. These cases illuminate the divergent paths that organizations can take in addressing security vulnerabilities, each path carrying its own ethical implications and impact on end-users.

**User-Centric Approach and Its Importance** Alias Robotics' approach, with its strict 90-day disclosure deadline, embodies a user-centric methodology. This policy not only impels vendors to act swiftly but also aligns with a broader ethical stance that prioritizes the safety and security of end-users. By setting clear deadlines and making vulnerability information public, Alias Robotics upholds the principle of transparency, empowering users with knowledge and means for protection. This approach is demonstrative of an ethical commitment to the community, where the security and well-being of users are placed at the center of cybersecurity practices.

**Vendor-Centric Approach: A Cautionary Tale** On the other hand, Open Robotics' less prescriptive ROS 2 Vulnerability Disclosure Policy, which lacks clear deadlines and a transparent process, reflects a vendor-centric approach. This policy, while ostensibly designed to manage vulnerabilities, falls short in fostering a proactive security culture within the community. Its lack of urgency and transparency may inadvertently lead to prolonged vulnerability exposure, thereby increasing risks for end-users. This approach can be seen as ethically problematic, as it does not adequately address the immediate security needs of the community and potentially places the interests of vendors above those of the users.

**Implications for End-Users** The contrasting policies and their implementation have direct implications for end-users, particularly in terms of their exposure to potential cyber threats. A policy that enforces strict deadlines and encourages open communication about vulnerabilities serves to enhance the overall security posture of the robotics ecosystem. It acknowledges the evolving nature of cyber threats and the importance of keeping users informed and equipped to respond to these threats. In contrast, a policy that lacks these elements may leave end-users vulnerable and uninformed, undermining the trust and reliability essential in the field of robotics.

**Security Researcher Perspective: Navigating Ethical and Practical Challenges** The exploration of vulnerability disclosure policies in the context of robotics also brings to the forefront the unique challenges faced by security researchers. These individuals, who play a pivotal role in identifying and reporting vulnerabilities, often navigate a landscape

### 3. Ethical Landscape in Robot Cybersecurity

---

fraught with ethical dilemmas and practical hurdles. A significant concern in this realm is the tendency of some vendor-centric disclosure policies to pressure researchers into revealing their findings without proper recognition, compensation, or, at times, under the threat of legal action.

**Compromised Incentives and Recognition** Security researchers invest significant time and effort in uncovering vulnerabilities, with the primary goal of enhancing system security and user safety. However, the expectation set by certain disclosure policies to give away this valuable information, often without compensation or adequate acknowledgment, can be demoralizing. This lack of incentive not only undermines the researchers' contributions but also potentially devalues the importance of their work. In an environment where recognition and fair compensation are lacking, the motivation to engage in this critical work can diminish, leading to a scarcity of skilled individuals willing to dedicate their efforts to cybersecurity research in robotics.

**Legal Threats and the Ethical Dilemma** The situation becomes even more complex when legal threats enter the equation. Researchers, in their pursuit to enhance cybersecurity, may find themselves in a precarious position when their findings are met with hostility or legal intimidation from vendors. This scenario presents an ethical quandary: on the one hand, researchers have a moral obligation to disclose vulnerabilities to protect end-users; on the other hand, they face potential legal repercussions for doing so. Such an environment can create a chilling effect, discouraging researchers from participating in vulnerability discovery and reporting, ultimately hindering the advancement of cybersecurity in the robotics field.

**Impact on the Security Landscape** The cumulative effect of these challenges is a security landscape where the vital contributions of researchers are not adequately valued or supported. This situation can lead to a reduction in proactive security research, leaving vulnerabilities undiscovered and unaddressed. Consequently, the overall security posture of robotic systems may be weakened, exposing end-users to increased risks. To foster a more robust and resilient cybersecurity environment, it is essential to recognize and support the critical role of security researchers. This support can manifest in various forms, including fair and transparent

policies, legal protections, recognition of their contributions, and avenues for compensation or incentives.

**The Way Forward** In conclusion, the ethical landscape of cybersecurity in robotics is one where user safety, transparency, and proactive engagement must be paramount. As robotics continue to integrate more deeply into various aspects of society and industry, the responsibility to ensure robust and ethical cybersecurity practices becomes increasingly critical. The experiences of Alias Robotics and Open Robotics serve as instructive examples, highlighting the necessity of policies that are not only technically sound but also ethically grounded. Such policies should aim to protect users, foster an informed and engaged community, and uphold the highest standards of safety and security in the rapidly evolving world of robotics.

### 3.5.3 Lack of learning resources

One of the significant challenges in the realm of robot cybersecurity is the lack of comprehensive learning resources. This shortage is exacerbated by the current vulnerability disclosure policies of various actors and groups, such as Open Robotics, as discussed above. These policies, often more vendor-centric than community-driven, do not facilitate the generation of educational content that could foster a deeper understanding of cybersecurity in robotics. This gap in knowledge and resources is particularly concerning given the increasing integration of robotics into various facets of society and industry.

In response to this void, the Robot Hacking Manual (RHM) [101], authored as part of this research, emerges as a crucial resource. The RHM is an introductory series about cybersecurity for robots, conceived with the intent to raise awareness in the field and underscore the importance of adopting a security-first approach. The manual offers comprehensive case studies and step-by-step tutorials, designed to provide a foundational understanding of robot cybersecurity. The content within the RHM is independent and not aligned with any particular organization, reflecting a genuine effort to democratize knowledge in this critical area.

An integral part of the RHM is the robot hacks list displayed in table 3.1. This list is a non-exhaustive compilation of cybersecurity research in robotics, detailing various robot vulnerabilities and attacks due to cybersecurity issues. The list serves as a testament to the evolving

### 3. Ethical Landscape in Robot Cybersecurity

landscape of threats in robot cybersecurity and underscores the necessity for continued vigilance and education in the field.

Codename/Theme	Robot Tech	Researchers	Description	Date
-	iRobot's Roomba J7	N/A	Personal pictures in a home environment were found on the Internet taken by an iRobot's Roomba J7 series robot vacuum.	19-19-2022
-	Unitree's Go1	d0tslash	A hacker found a kill switch for a gun-wielding legged robot.	09-08-2022
-	Enabot's Ebo Air	Modux	Researchers found a security flaw in Enabot Ebo Air robot leading to potential remote-controlled robot spy units.	21-07-2022
Analyzing DDS	Various DDS implementations	Various	Research on the Data Distribution Service (DDS) protocol revealing vulnerabilities.	19-04-2022
Hacking ROS 2	ROS 2	Various	Discovery of security vulnerabilities in ROS 2's communication middleware.	22-04-2022
JekyllBot:5	Aethon TUG	Cynerio	Collection of vulnerabilities enabling remote control of Aethon TUG smart robots.	01-04-2022
Robot Teardown	Universal Robots	Various	Research uncovering security flaws in UR robots through hardware analysis.	20-07-2021
Rogue Automation	Various industrial robots	Various	Analysis revealing risks in industrial automation programming languages.	01-08-2020
Securing Disinfection Robots	UVD Robots	Alias Robotics	Vulnerabilities in disinfection robots used against COVID-19.	19-09-2020
MiR Bugs Week	Mobile Industrial Robots	Alias Robotics	Public release of multiple vulnerabilities in MiR robots.	24-06-2020
Smart Manufacturing Attacks	Mitsubishi Melfa	Various	Security analysis of attacks on a smart manufacturing system.	01-05-2020
UR Bugs Week	Universal Robots	Alias Robotics	Exposure of security issues in Universal Robots' products.	31-03-2020
Akerbeltz	Universal Robots UR3, UR5, UR10	Alias Robotics	Creation of industrial robot ransomware demonstrated on UR robots.	16-12-2019
Rogue Robots	ABB IRB140	Various	Theoretical and experimental challenges in the security of modern industrial robots.	01-05-2017
Hacking Robots Before Skynet	Various robots	IOActive	Discovery of critical cybersecurity issues in multiple robots.	30-01-2017
ROS: Safe	Insecure	ROS	Early study on security vulnerabilities in ROS.	28-02-2014

Table 3.1: Robot Hacks: Notable Cybersecurity Incidents in Robotics

## 3.6 Conclusion

This chapter, situated within the ethical framework of cybersecurity in robotics, has navigated through the intricate landscape of robot security, revealing its multifaceted nature and current state. The exploration, driven by the goal to understand and critique the prevailing ethical standards and practices in robot cybersecurity, delved deep into various dimensions of this field.

The findings from the systematic review of literature, combined with insights from surveys conducted across prominent robotics communities,

have highlighted several key observations. A noticeable gap exists between the anticipated and actual investment in cybersecurity within the robotics domain, particularly in the ROS community (Observation 3.3.1). This gap not only underlines the potential growth area in cybersecurity actions but also mirrors the nascent stage of maturity in this field compared to established sectors like IT or OT (Observation 3.3.2).

A surprising revelation was the majority of the robotics community's lack of exposure to cyber-attacks (Observation 3.3.3), suggesting a lower perceived threat level. This could potentially lead to a complacency in adopting rigorous cybersecurity measures. Furthermore, the tendency to focus on perimeter defense strategies (Observation 3.3.4), as noted in both ROS and ROS-I communities, signals a possible oversight of insider threats, particularly crucial in industrial environments where robotics systems are integral.

The responsibility for cyber incidents in Europe is largely attributed to the supply chain, with system integrators and robot vendors bearing the brunt of the blame, rather than the end-users (Observation 3.3.5). This points to an ethical standpoint where the onus of security is more on those who manufacture and integrate rather than those who operate.

The situation with collaborative robot manufacturers like MiR and UR is particularly alarming, as they exhibit a pattern of unmitigated zero-day vulnerabilities (Observation 3.4.1). This inaction contrasts sharply with the approach of manufacturers like ABB, who show a varied response to cybersecurity issues (Observation 3.4.2). The ratio of publicly disclosed vulnerabilities to those kept private further serves as an indicator of a manufacturer's readiness and approach towards cybersecurity (Observation 3.4.3).

Table 3.2 offers a summary of observations from robotics security surveys and research. Overall, this chapter has underscored the importance of re-evaluating and reinforcing cybersecurity measures in the rapidly evolving field of robotics. It advocates for a shift from traditional perimeter defense strategies to more comprehensive approaches that consider insider threats and embrace practices like *zero-trust*. The observations made here emphasize the need for increased investment and focus on robot-specific security measures and offensive security assessments. This proactive approach is crucial to enhance the resilience of robotic systems and to bridge the gap between current practices and the evolving cybersecurity landscape in robotics.

### 3. Ethical Landscape in Robot Cybersecurity

---

Observation	Description
3.3.1	There is a gap between the expectations and the actual investment in cybersecurity in the ROS community, indicating potential future growth in cybersecurity actions.
3.3.2	The lack of robot-specific security measures and offensive assessments in the ROS community indicates a lower maturity level in cybersecurity compared to other sectors like IT or OT.
3.3.3	Majority of respondents in both ROS and PX4 communities have not witnessed a cyber-attack, suggesting a lower perceived threat level.
3.3.4	Security mitigations in both ROS and ROS-I groups primarily focus on perimeter defense, indicating a potential underestimation of insider threats in industrial environments.
3.3.5	In Europe, most respondents believe that the responsibility for cyber incidents lies with the supply chain (system integrators and robot vendors), not the end-user.
3.4.1	Collaborative robot manufacturers MiR and UR have unmitigated zero-day vulnerabilities, indicating a lack of responsive action from these manufacturers.
3.4.2	ABB robots show a mixed response to cybersecurity, with some vulnerabilities patched quickly while others remain unaddressed for years.
3.4.3	The ratio of publicly disclosed to private vulnerabilities is an indicator of a robot manufacturer's security readiness. ABB shows a higher rate of public disclosures, while MiR and UR have more undisclosed vulnerabilities.

Table 3.2: Summary of Observations from Robotics Security Surveys and Research

This critical examination of the ethical standards in robot cybersecurity not only sheds light on the current state of affairs but also sets the stage for future advancements. One of the central themes uncovered is the critical role of vulnerability disclosure policies in shaping the ethical landscape of robot cybersecurity. As highlighted in the discussion, there is a stark contrast between the user-centric approach and the vendor-centric approach. The former underscores the ethical imperative of prioritizing end-user safety and system security through timely and transparent disclosure of vulnerabilities. In contrast, the latter's approach, characterized by a lack of urgency and transparency, potentially compromises the security and trust in robotic systems. By challenging established norms and scrutinizing practices like responsible disclosure, this chapter contributes to shaping a more ethical landscape and secure future in robotics, where the safety and integrity of both humans and machines are paramount.



Part III

# **Blueprints**



## Chapter 4

# Cybersecurity in ROS

**Contributions:** *Leading scientific author in [102] wherein I led contributions to the DevSecOps methodology and multi-year contributions to software. Subject-matter expert in robotics and security in [103] wherein I kicked-off the initial implementation discussions, architected the ROS-integration (core of the paper), and released it into ROS Humble for widespread use. Both works are open source.*

---

### 4.1 Introduction

A robot is a network of networks [104]. One that is comprised of sensors to perceive the world, actuators to produce a physical change, and computational resources to process it all and respond coherently, in time, and according to its application. Security is of paramount importance in this context, as any disruption of any of these robot networks can cause the complete robotic system to misbehave and compromise the safety of humans, as well as the environment [2, 105].

The Robot Operating System (ROS) [4] is the *de facto* framework for robot application development. Widely used to govern interactions across robot networks, at the time of writing, the original ROS article [4] has been cited more than 9300 times, which shows its wide acceptance for research and academic purposes. ROS was born in this environment: its primary goal was to provide the software tools and libraries that users would need to

## 4. Cybersecurity in ROS

---

employ to undertake novel robotics research and development. Adoption in industry has also been rapidly increasing over the last few years. According to the latest ROS community metrics [5] sampled every year in July, the number of ROS downloads has increased by over 50%, with about 600 million downloads between July of 2020 and July of 2021. Moreover, based on the download percentages reported from `packages.ros.org`, one can observe a significant increase in adopting ROS 2, which suggests that by 2023 there would be more users using ROS 2 than its predecessor<sup>1</sup>.

ROS was not designed initially with security in mind, but as it started being adopted and deployed into products or used in government programs, more attention was drawn to security issues. Some of the early work on securing ROS included [24, 25] or [26], both appearing in the second half of 2016. At the time of writing, none of these efforts remain actively maintained and the community focus on security efforts has switched to ROS 2. A recent study [83] that surveyed the security interests in the ROS community presented data indicating that 73% of the survey participants considered that they had not invested enough to protect their robots from cyber-threats. The same number of participants indicated that their organizations were open to invest, however only 26% acknowledged to actually have invested. This led the original authors to conclude that there is a gap between the security expectations and the actual investment. In this work, it's argued that this gap is the result not only of the immaturity of security in robotics or the know-how but also by the lack of usability of the available security tools. Being conscious that security in robotics is not a product, but a process that needs to be continuously assessed in a periodic manner [19, 92, 101], this work advocates for a usable security approach in robotics as the best way to remain secure.

In this chapter we introduce SROS2, a series of developer tools, meant to be usable and that facilitate adding security capabilities to ROS 2 computational graphs. We present in here a security methodology consisting of six steps that allow securing ROS 2 graphs iteratively, with the aid of SROS2. Driven by an application use case, we discuss how SROS2 allows achieving security in complex graphs involving popular ROS 2 packages and analyze the security trade-offs and limitations of the current

---

<sup>1</sup>The reader must also note that past studies estimated that by 2024, 55% of the total commercial robots shipped that year would include at least one ROS package. For more details, refer to <https://www.businesswire.com/news/home/20190516005135/en/Rise-ROS-55-total-commercial-robots-shipped>.

tooling. The key contributions of this work are:

- Create SROS2, a set of usable tools for adding security to ROS 2 that: (1) help introspect the computational graph by extracting communication middleware-level information; (2) simplify the security operations creating Identity and Permissions Certificate Authorities (CA) that govern the security policies of a ROS 2 graph; (3) help organize all security artifacts in a consistent manner and within a directory tree that is generated within the current ROS 2 workspace overlay; (4) help create a new identity for each enclave, generating a keypair and signing its x.509 certificate using the appropriate CA; (5) create governance files to encrypt all DDS traffic by default; (6) support specifying enclave permissions in familiar ROS 2 terms which are then automatically converted into low-level DDS permissions; (7) support automatic discovery of required permissions from a running ROS 2 system; and (8) dissect communication middleware interactions, to extract key information for the security monitoring of the system.
- Propose a methodology for securing ROS 2 computational graphs that provides roboticists with a structured process to continuously assess their security.
- Expose insights into how to apply SROS2 to real ROS 2 computational graphs by presenting an application case study focused on analyzing the Navigation2 and SLAM Toolbox stacks in a TurtleBot3 robot.

The core components of SROS2 are disclosed under a commercially friendly open-source license and are available and maintained at <https://github.com/ros2/sros2>.

## 4.2 Background

Considering how ROS was originally intended as a fast prototyping robotic framework, security was not considered a priority feature in its first iteration. As ROS has evolved from the prototyping to the real-world industrial applications the entire stack came to be in dire need of cybersecurity attention [101] [106].

## 4. Cybersecurity in ROS

---

A first partial analysis, with the goal of understanding what prevented ROS from being used industrially, was conducted by McClean *et al.* [107]. By means of a 'honeypot' system, at DEFCON-20, they collected how malicious users would tackle a robot in the wild. Dieber *et al.* [43] provided a complete and in-depth analysis of the security vulnerabilities and attack surfaces in ROS systems and how to exploit them, highlighting the gaps in the security of the framework. A considerable amount of research has been done as regards the publish-subscribe paradigm reviewing the performance and the techniques to secure it either via the communication channel, and ROS' internal mechanisms [108]. In the first case, via message authentication [109], within the later addition of using of encryption and security artifacts [27] [31] [110]. In the latter case, by enhancing the middleware behaviour with some extra, such as a run-time monitor to filter out and log all the requests and operations sent in the graph [111], an Application Level Gateway - that wraps the existing API calls to enforce authentication and authorization - that exposes a permission token to be evaluated before executing the requested operation [112]; to the extent of forking the implementation, modifying the transport mode via IPSec [113], or via a security architecture intended with the addition of x.509 certificates and authorization server [24]. Unfortunately, those approaches suffered to some extent with limitations and downsides, such as a lack of flexibility (e.g. Single Point of Failure (SPOF)) and usability, which were tackled in the Secure ROS (SROS) initiative [34]. With the objective of providing additions to the ROS API and ecosystem to support modern cryptography and security measures, the project introduced new security features to the core of ROS' codebase and, more importantly, a set of tooling to ease the burden on the developers of correctly implementing security.

With the second iteration of the framework in ROS 2, thanks to the adoption of DDS as the communication middleware <sup>2</sup>, we observed how the inherited security measures and methodologies in the system have evolved the framework. However, we can no longer overlook how its complexity still remains prone to human error in processes such as the access control artifacts distribution [33], or even to overlooking exposed attack surfaces [36]. Moreover, keeping track of all the new pieces to the ROS puzzle became even more demanding and lengthy procedurally, requiring continuous attention and systematic security analysis—which

---

<sup>2</sup>[https://design.ros2.org/articles/ros\\_on\\_dds.html](https://design.ros2.org/articles/ros_on_dds.html)

left usability challenging [114]. The work presented here addresses this challenge with a security toolset (SROS2) and a security methodology for robotics.

## 4.3 Approach

A methodology inspired by the DevSecOps approach [17] is proposed to secure ROS 2 computational graphs: (A) introspect the graph and model its security landscape to determine the necessary security policies and enclaves; implement such policies by (B) defining the authentication and (C) authorization configurations; (D) generate all the required security artifacts; (E) deploy them appropriately across robotic systems; and (F) continuously monitor the network, reverting to (A) modeling when appropriate.

### 4.3.1 Modeling

Modeling refers to the use of abstractions to aid in a thought process. In security, threat modeling aids in thinking about risks and determines the threat landscape. The output of this effort is often called the threat model. Commonly, a threat model enumerates the potential attackers, their capabilities, resources and their intended targets. In the context of robot cybersecurity, a threat model identifies security threats that apply to the robot and/or its components<sup>3</sup> while providing means to address or mitigate them for a particular use case. A threat model also provides inputs that are used to then determine a set of policy rules (or principles) that direct how ROS 2 should provide security services to protect sensitive and critical graph resources. When put together these policy rules are called the *security policy*.

SROS2 aims to provide tools to introspect and model the security of ROS 2 computational graphs into the desired security policies. Introspection of the graph can be performed in two ways:

1. By leveraging the ROS 2 API and the framework for ROS 2 command line tools (`ros2cli`), we can pull ROS *Nodes*, *Topics*, *Services* or *Actions* information (among others) from the ROS 2 graph and display these in the CLI, see Listing 4.1. This allows us to get a grasp of

---

<sup>3</sup>both software and hardware, including computational graph resources.

## 4. Cybersecurity in ROS

---

the computational graph from a ROS 2 perspective. Other tools such as RViz [115] or rqt help get a visual depiction of the graph and its abstractions.

2. Monitoring network interactions at the ROS communication middleware-level can be extremely helpful to model security but incredibly cumbersome from a usability perspective unless the right tooling is provided. ROS 2 uses OMG's Data Distribution Service (DDS) [21] as its default communication middleware, which is a complex specification. To facilitate introspection of DDS, SROS2 leverages *scapy* [116], a powerful interactive packet manipulation library that allows us to forge or decode network packets. Particularly, we contributed an open source *scapy* dissector<sup>4</sup> that allows us to dissect the wire-level communication protocol that is used by the default ROS 2 communication middleware: the Real-Time Publish Subscribe protocol (RTPS) [117]. Using this, SROS2 provides tooling that allows monitoring network interactions, capturing DDS databus information directly and displaying these for the security analyst's consumption.

Listing 4.1: SROS2 extends ROS 2 APIs to facilitate computational graph introspection at the networking level for modeling purposes.

```
# ROS 2 CLI API allows direct introspection
ros2 topic list
/cmd_vel
/robot_state_publisher
...
ros2 node list
/turtlebot3_diff_drive
...
# SROS2 extensions allow introspecting DDS
ros2 security introspection
DDS endpoint detected (hostId=16974402, appId=2886795267, instanceId=10045242)
- version: 2.4
- vendorId: ADLINK - Cyclone DDS
- IP: 192.168.1.34
- transport: UDP
DDS endpoint detected (...)
```

For complete threat modeling, we refer the reader to [118] which discusses details around security modeling ROS 2 computational graphs.

<sup>4</sup>see <https://github.com/secdev/scapy/pull/3403>

### 4.3.2 Authentication

Authentication provides proof of a claimed identity ( $\neq$  identification, determination of an unknown entity). ROS 2 offloads authentication to its underlying communication middleware, DDS. By default, DDS allows any arbitrary *DomainParticipant* to join any *Domain* without authentication. DDS however provides the means to verify the identity of the application and/or the user that invokes operations on the DDS databus through its *DDS Security* extensions [18]. With these, for protected DDS *Domains*, a *DomainParticipant* that enables authentication will only communicate with other *DomainParticipants* that also have authentication enabled.

To favour usability and reduce human errors, all implementation details of authentication in ROS 2 through DDS are abstracted away by the SROS2 tools. The appropriate artifacts for enabling authentication capabilities are produced in the *Generation* step (4.3.4) of the DevSecOps methodology, and default to the security mechanisms specified by OMG's DDS Security [18]. In particular, each *DomainParticipant* uses a Public Key Infrastructure (PKI) with a common shared Certificate Authority (CA): Identity CA. All participants interoperating securely must be pre-configured with Identity CA and have a signed certificate from it. Participants are expected to use mutual authentication through a challenge-response mechanism supported by either the Rivest Shamir Adleman (RSA) [119] or the Elliptic Curve Digital Signature Algorithm (ECDSA) [120] asymmetric encryption algorithms. Shared secrets are established using using the Diffie-Hellman (DH) [121] or Elliptic Curve DH (ECDH) (Ephemeral Mode) [122] key agreement protocols.

Listing 4.2 shows an example of how SROS2 tools abstract the complexity of DDS authentication away from ROS developers. The `governance.xml` policy document is auto-generated by SROS2 and captures domain-wide security settings that include authentication aspects. Additional details about the underlying authentication process and the security artifacts are available in [18], [123] and [124].

Listing 4.2: An extract from `governance.xml` policy document generated by SROS2 illustrating domain-wide security settings such as how to handle unauthenticated participants, whether to encrypt discovery or the default rules for access to topics.

```
...
<allow_unauthenticated_participants>false</allow_unauthenticated_participants>
```

## 4. Cybersecurity in ROS

---

```
<enable_join_access_control>true</enable_join_access_control>
<discovery_protection_kind>ENCRYPT</discovery_protection_kind>
<liveliness_protection_kind>ENCRYPT</liveliness_protection_kind>
<rtps_protection_kind>SIGN</rtps_protection_kind>
<topic_access_rules>
  <topic_rule>
    <topic_expression>*</topic_expression>
    <enable_discovery_protection>true</enable_discovery_protection>
    <enable_liveliness_protection>true</enable_liveliness_protection>
    <enable_read_access_control>true</enable_read_access_control>
    <enable_write_access_control>true</enable_write_access_control>
    <metadata_protection_kind>ENCRYPT</metadata_protection_kind>
    <data_protection_kind>ENCRYPT</data_protection_kind>
  </topic_rule>
</topic_access_rules>
...
```

### 4.3.3 Authorization

Authorization helps define and verify the policies that are assigned to a certain identity. Access control instead –also called permissions or privileges– are the methods used to enforce such policies. While access control is handled by the DDS implementation, authorization policies need to be defined by the developer. SROS2 helps map these policies from the ROS 2 computational graph to the underlying DDS databus abstractions through two resources: the Permissions CA and a `permissions.xml` policy document. Listing 4.3 shows an extract from one of the policy documents that defines the authorization profile for a particular ROS 2 Node. Details about how access control is implemented by the underlying communication middleware are discussed in [18] and [125].

Listing 4.3: SROS2 provides means to define authentication policies through XML files.

```
<profile node="turtlebot3_diff_drive" ns="/">
  <xi:include href="common/node.xml"
    xpointer="xpointer(/profile/*)"/>
  <topics subscribe="ALLOW">
    <topic>/cmd_vel</topic>
  </topics>
  <topics publish="ALLOW">
    <topic>odom</topic>
    <topic>tf</topic>
  </topics>
</profile>
```

```
</profile>  
<profile node="turtlebot3_imu" ns="/">  
...  
</profile>
```

#### 4.3.4 Generation

*Modeling* (4.3.1), *Authentication* (4.3.2) and *Authorization* (4.3.3) steps of our methodology (which follows DevSecOps in robotics [17]) help define one or multiple security policies. To help implement such policies, SROS2 provides means to automate the generation of the corresponding security artifacts and simplify the translation to the underlying DDS implementation. To do so, SROS2 maps a security policy to an *enclave*: a set of ROS 2 computational graph resources that operate in the same security domain, use particular Identity CA and Permissions CA authorities, and share the protection of a single, common, continuous security perimeter.

All secure interactions in ROS 2 computational graphs must use an enclave that contains the runtime security artifacts unique to that enclave, yet each Node may not necessarily have a unique enclave. Multiple enclaves can be encapsulated in a single security policy to accurately model the information flow control. Users can then tune the fidelity of such models by controlling at what scope enclaves are applied at deployment. For example, one unique enclave per robot, or per swarm, or per network, etc.

Listing 4.4 shows how SROS2 tools help generate all artifacts to implement a new security policy, inferred directly from the running ROS 2 graph. For a more complex policy that involves multiple enclaves, we refer the reader to [126].

Listing 4.4: SROS2 provides tools to implement security policies and in ROS 2 computational graphs, generating all security artifacts necessary.

```
# Generate a new keystore with Identify and Permission CA keys,  
# associated certificates and a authentication structure through  
# governance.xml file  
ros2 security create_keystore new_keystore  
  
# Inspect current ROS graph and produce a security policy  
ros2 security generate_policy new_keystore/my_policy.xml  
  
# Generate all security artifacts necessary to enforce the policy,  
# this includes enclaves and the access control permission files  
ros2 security generate_artifacts \
```

## 4. Cybersecurity in ROS

---

```
-k new_keystore \  
-p new_keystore/my_policy.xml
```

### 4.3.5 Deployment

Deployment is a relevant phase in the DevSecOps methodology [17] and must be also exercised securely. We consider three types of deployments of both artifacts and secure information:

1. Distribution of policy artifacts: the resulting artifacts from the *Generation phase* (4.3.4) must be securely deployed into the targeted robots and related systems. At the time of writing SROS2 does not provide any particular special utilities to deploy security artifacts. We however direct readers to the ongoing efforts to launch ROS 2 graphs remotely and in multi-machine environments [127] for inspiration.
2. Message authentication: verification of the Message Authentication Codes (MAC) is performed using Advanced Encryption Standard (AES) with Galois MAC (AES-GMAC). DDS security extensions abstract this away from the ROS developer.
3. Encryption of secure DDS interactions: authenticated symmetric cryptography governs all DDS interactions within a security policy using also AES in Galois Counter Mode (AES-GCM). Similar to message authentication, DDS abstracts this away from the developer and is enabled automatically provided that the security policy is configured appropriately.

### 4.3.6 Monitoring and mitigation

The last phase in the DevSecOps methodology leads to a never ending loop of continuous *Monitoring, mitigation* (4.3.6) and *Modeling* (4.3.1). This way, security in ROS 2 computational graphs becomes a moving target, a process—one that demands continuous assessments as changes occur in the robots, the network, or as new security flaws are discovered affecting the running systems.

SROS2 provides tools for monitoring running ROS 2 graphs and detecting possible flaws. Listing 4.5 shows an example:

Listing 4.5: SROS2 provides tools to dissect DDS interactions, extract key information and map it to outstanding security flaws affecting DDS.

```
# monitor the network segment for vulnerabilities
# affecting DDS participants
ros2 security monitor
sniffing the DDS network...
Vulnerable DDS endpoint found (hostId=16974402, appId=2886795267, instanceId=10045242)
- vendorId: Real-Time Innovations, Inc. - Connex DDS
- version: 6.0.1.25
- CVE IDs:
  * CVE-2021-38487
  * CVE-2021-38435
```



## 4.4 Application and analysis

To apply the DevSecOps methodology, as defined originally in [17], we demonstrate the application of SROS2 using two of the most commonly used frameworks in ROS 2, the `navigation2` [128] and `slam_toolbox` [129] stacks. Particularly, the Navigation2 project<sup>5</sup> provides a software stack including path planning algorithms and behavioral navigation servers that can be seamlessly integrated with existing sensor perception pipelines, localization and mapping services, and drivetrain velocity controllers to support various mobile robotic applications. While Navigation2 remains mostly agnostic of robotic platforms, we selected the widely accessible and community supported TurtleBot3<sup>6</sup> as the target robot for analysis—consisting of a differential drive, circular base footprint, and ground level 2D scanning LIDAR. The application case study is depicted in Figure 4.1.

To start, we begin with the modeling (Subsection 4.3.1), authentication (Subsection 4.3.2) and authorization (Subsection 4.3.3) phases in order to bootstrap an initial security policy that captures the minimal spanning set of security measures required for the nominal function of the application across the distributed computation graph. We can either first bring up the ROS 2 application under a controlled network environment with security mode disabled, or provision an initial keystore enclave with temporary key-material and only access control governance disabled; the first option includes minimal setup while being more transparent to debug, while the later is advantageous in modeling policies directly from field deployments across untrusted networks.

With the ROS 2 application running, SROS2 can capture the topology of the computation graph to populate the permission profiles within the initial policy, registering each active ROS 2 node and respective topic publication and subscription. A limitation in SROS2's current snapshot approach however is in accurately modeling more ephemeral resource access events, such as service clients or action requests. While ROS 2's internal graph API (that SROS2 uses to sample topology measurements) provides a middleware agnostic interface, the observation window is only instantaneous and can easily miss asynchronous resource access events.

Given the graph API limitations, it's often necessary to iteratively test the generated policy by using it to update the signed permission

---

<sup>5</sup><https://github.com/ros-planning/navigation2>

<sup>6</sup><https://www.robotis.us/turtlebot-3>

## 4. Cybersecurity in ROS

---

and governance files and relaunch the application with access control enabled. For moderate to advanced applications such as those relying on Navigation2, permission access denied errors may inevitably be encountered. With ROS 2 however, such events can be logged and aggregated into policy refinement, specifying the node and resource namespaces denied.

After iterative policy refinement, once the tested application is fully functional with enforced access control, the policy can then be further optimized. Such policy optimizations include sorting common permission patterns into smaller sub-profiles, being more manageable to audit and modularly reusable across repeating permission sets in a global policy. We demonstrate it in [130]. This auditing process also provides an opportunity to assess the granularity of the policy as well, from both permission Access Control (AC) and Information Flow Control (IFC) perspectives.

While the minimal spanning set of AC permissions may be optimally secure in terms of the Principle of Least Privilege [131], it may not be optimally usable for a target application domain. Though most computation graphs in ROS 2 are largely static at runtime, cases where resource namespaces change over the application's lifecycle do exist. For example, multi-robot systems may fluctuate as agents enter or exit networks for missions or maintenance. Additionally, node namespaces sometimes include sequence numbers to ensure namespace uniqueness. To accommodate such scenarios, permissions could be modified to include wildcards as necessary. While static permissions are straightforward to interpret and less likely to inadvertently introduce policy flaws, wildcarding select permissions provides a usable compromise when required.

When auditing from an IFC perspective, optimizing the policy into assorted enclaves becomes a key consideration. As all ROS 2 nodes composed into a shared process share a common DDS context, they subsequently share the same security enclave or set of permission profiles. This of course is inherently coupled with how the application is architected and to be deployed across a distributed system. As such, security requirements for IFC may then instead dictate aspects of the application's designs. The degree of granularity of IFC sought then dictates the allotment of enclaves used to contain sub-profiles for the application's policy.

In the case of Navigation2 and its large degree of coupling and composition of nodes, the planning stack derived from a single source tree is perhaps best relegated to its own enclave, while still being readily

separable from any other enclave dedicated to perception or control nodes. Admittedly, such auditing procedures in determining the allotment of enclaves remains rather ambiguous for users, and so presents another area of ergonomics for SROS2 to help automate or advise through formal analysis.

The source code of the demonstrated application case study is available at [132]. The resulting security policies of applying the DevSecOps methodology are also available in the same repository and show various profiles that result from a systematic assessment.

## 4.5 ROS 2 over VPN: Tunneling Communications

### 4.5.1 Challenges in Securing ROS 2 Communications

ROS 2, built upon the Data Distribution Service (DDS), faces significant security challenges. Despite the existence of DDS Security extensions [18], these measures are still in their infancy concerning robust security implementations. As outlined in upcoming Chapter 7, modern DDS implementations exhibit numerous vulnerabilities, some of which result from collaborative research by security experts. These vulnerabilities open up potential attack vectors for malicious actors, compromising the integrity of ROS 2 systems.

Furthermore, the DDS Security Specification itself is plagued with unresolved security issues, detailed at <https://issues.omg.org/issues/spec/DDS-SECURITY>. This list highlights various bugs and loopholes that further exacerbate the security risks associated with ROS 2, making it difficult to assert that ROS 2, in its current state, can be deemed secure.

### 4.5.2 ROS 2 over VPN: A Layered Security Approach

Given the aforementioned security concerns, a practical solution is to encapsulate ROS 2 and DDS communications within an additional security layer. This approach involves tunneling all ROS 2 communications through a secure channel using a Virtual Private Network (VPN). This method, referred to as *ROS 2 over VPN*, provides an extra layer of security by encrypting the data transmitted over the network, thereby mitigating the risks associated with potential vulnerabilities in DDS implementations and specifications.

### 4.5.3 Implementation

The concept of *ROS 2 over VPN* was researched and successfully demonstrated in the FogROS2 project [103], which builds upon the original contributions and principles presented earlier at [17]. Figure 4.2 illustrates the architecture employed in this implementation. By routing ROS 2 communications through a VPN, data privacy is enhanced, and the network is safeguarded against eavesdropping and other forms of cyberattacks. FogROS2 relies on a series of extensions of the ROS 2 launch system. We implemented the FogROS2 launch sequence, shown in Fig. 4.2 that starts after processing the launch script.

The steps FogROS2 takes are: (1) connect to the cloud provider through its programmatic interface to create and start a new instance along with setting up security groups to isolate from other cloud computers, and generating secure communication key pairs; (2) install the ROS libraries and dependencies on the cloud machine needed for the robot application to run in the cloud; (3) set up virtual private networking (VPN) on robot and cloud machine to secure the ROS 2 DDS communication between them; (4) copy the workspace and all of its node software, from the robot to the cloud machine; (5) configure the DDS provider’s discovery mechanism to work across the VPN; (6) launch docker instances; (7) launch cloud-based nodes; and finally, (8) launch nodes on the robot.

Once the launch process is complete, the nodes running on the robot and on the cloud machine(s) securely communicate and interact with each other—and the only change needed was a few lines of the launch script.

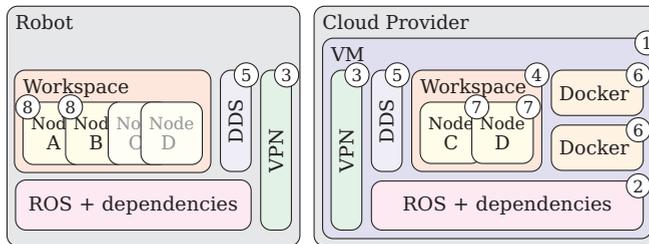


Figure 4.2: FogROS2 proposes an additional security layer to ROS 2 communications using a VPN, reproduced from [103].

The use of VPN in this context offers several advantages:

1. **Enhanced Data Encryption:** VPN encrypts all data transmitted between ROS 2 nodes, ensuring that sensitive information remains confidential even if intercepted.
2. **Network Isolation:** By creating a virtual network overlay, ROS 2 communications are isolated from other network traffic, reducing the exposure to external threats.
3. **Access Control:** VPNs can be configured to allow access only to authenticated and authorized devices and users, providing a controlled environment for ROS 2 operations.
4. **Flexibility and Scalability:** VPN solutions can be easily scaled and adapted to various deployment scenarios, from small-scale laboratory setups to large-scale industrial applications.

In summary, while the native security features of ROS 2 and DDS are still maturing, employing a VPN to tunnel communications offers an effective interim solution to address the current security limitations. This approach provides robust encryption, enhanced network control, and improved data privacy, thereby fortifying ROS 2 applications against potential cybersecurity threats.

However, this added security does not come without trade-offs. Notably, the introduction of a VPN can impact the performance of communications in ROS 2 systems. This topic will further expanded in Chapter 5.

## 4.6 Conclusion

In this chapter we presented various approaches to add security to ROS 2 in a scalable manner. Namely, SROS2, a series of developer tools focused on usable security that allow adding security capabilities to ROS 2 computational graphs. We introduce a methodology around these tools consisting of 6 basic steps and aligning to the common DevSecOps flows [17]: (A) introspect the computational graph and model its security to determine the necessary security policies and enclaves; (B) define authentication and (C) authorization configurations; (D) generate all the required security artifacts for implementing such policies; (E) deploy them appropriately across robotic systems; and (F) continuously monitor the network, reverting to (A) modeling when needed. SROS2 facilitates

each one of these steps by integrating itself tightly into the usual ROS 2 development flows.

We present an application case study discussing how to propose a secure architecture for the TurtleBot3 robot using the `navigation2` and `slam_toolbox` stacks. This is of special interest since it aligns to the software architecture that many industrial and professional robots are using today, given the popularity of these packages.

We introduce security as a process in robotics and correspondingly, the work herein aims to pave the way for enabling security processes, particularly in ROS 2. Alongside the never-ending reality of security, we acknowledge that SROS2 has various limitations that deserve further attention and improvements. Some of these include the lack of granularity of security configurations in the current abstractions, which makes it difficult to configure encryption and authentication options separately. Others refer to the lifecycle management of security artifacts, including updating certificates and keys, wherein secure deployment plays a key role. We are particularly keen on improving SROS2 mechanisms in the future to ensure secure lifecycles while minimizing the downtime impact in ROS 2 graphs. Promising directions for future work also include the development of more advanced monitoring and introspection capabilities, the extension of SROS2 to other communication middlewares (beyond DDS) and finally, the continuous improvement of the usability of the tools. For this, we believe that the use of Graphical User Interfaces (GUIs) represents an interesting opportunity to further facilitate SROS2 usability to non-roboticists. On top of SROS2, we also discuss the shortcomings of the DDS Security extensions [18] and how additional layers of security via VPN could help mitigate them.

The work presented here aims to inspire groups in robotics to add security to their robotic computational graphs. We look forward to security in robotics becoming more usable and accessible, minimizing the threat landscape that lies before us now, and closing the window of opportunity for bad actors. In the next chapter we will discuss the computational performance impact of adding security to ROS robotic systems and how it can be estimated experimentally with state-of-the-art benchmarking methods.

## Chapter 5

# The Cost of Security: Benchmarking Robot Computations

**Contributions:** *Leading author at [133], both scientifically and implementation-wise. Rallied the robotics community towards its adoption and through recorded online sessions, throughout many months. Funded contributors and engaged other researchers and industry to support the final research product. This work is open source.*

---

### 5.1 Introduction

As discussed in the previous chapter, while the inherent security mechanisms of ROS 2 and DDS are undergoing development and refinement, the strategy of tunneling communications through a VPN emerges as an effective interim measure to mitigate existing security vulnerabilities. This method introduces an additional layer of robust encryption, affording enhanced control over the network and bolstering data privacy. As a result, ROS 2 applications gain a fortified defense against a spectrum of cybersecurity threats. However, this added security does not come without trade-offs. Notably, the introduction of a VPN can impact the performance of communications in ROS 2 systems.

## 5. The Cost of Security: Benchmarking Robot Computations

---

### *VPNs in robotic deployments and performance challenges*

As robotics increasingly integrate into industries like healthcare and manufacturing, VPNs are becoming essential for remote operation and access. This shift presents performance challenges due to VPN overhead, especially notable in use cases like teleoperated surgical robots and industrial manipulators. The initial performance hurdle arises from the inherent computing load of VPNs in linking robotic systems.

A second, more complex challenge emerges with the implementation of finer security measures. Traditional VPN setups in robotics often adopt a *'castle-and-moat'* approach, where insiders within each location's VPN are considered trusted, often lacking encryption and access control among them. However, a more secure, *'zero-trust'* model is gaining traction. This model treats all participants, even within the same network, as potential risks, mandating encryption and access control for each interaction. Although this significantly bolsters security, it also adds substantial computational overhead, intensifying the performance challenges introduced by VPNs in robotic deployments.

The performance downgrade can manifest in various ways, primarily due to the additional processing required for encrypting and decrypting data, as well as the overhead introduced by the VPN protocol itself. While VPN solutions like WireGuard [134] are designed for efficiency and minimal overhead, they inevitably add latency to communications. In real-time robotic applications, where milliseconds can be critical, this added latency can affect the responsiveness and synchronization of robotic components. Moreover, the throughput of data can be constrained by the VPN, potentially limiting the bandwidth available for intensive data streams, such as video feeds or large sensor data arrays.

Another aspect to consider is the complexity introduced by VPN routing and network management. While VPNs can enhance security by segmenting and controlling network traffic, they also add layers of network configuration that can be challenging to optimize for high-performance robotic applications. This complexity can lead to inefficiencies in network routing, further impacting communication speed and reliability.

Therefore, while employing a VPN provides a strong interim solution for securing ROS 2 communications, it's important to carefully consider the performance implications, especially in scenarios where real-time

processing and high data throughput are essential. Future advancements in ROS 2 and DDS security features may offer more integrated solutions that balance robust security with the performance needs of advanced robotic systems. For now, the choice to use a VPN must be weighed against the specific requirements and constraints of each individual robotic application.

Beyond security itself, more generally in robotics, in order for robotic systems to operate safely and effectively in dynamic real-world environments, their computations must run at real-time rates while meeting power constraints. Towards this end, accelerating robotic kernels on heterogeneous hardware, such as GPUs and FPGAs, is emerging as a crucial tool for enabling such performance [135, 136, 137, 138, 139, 140, 141]. This is particularly important given the impending end of Moore’s Law and the end of Dennard Scaling, which limits single CPU performance [142, 143].

While hardware-accelerated kernels offer immense potential, they necessitate a reliable and standardized infrastructure to be effectively integrated into robotic systems. As the industry leans more into adopting such standard software infrastructure, ROS [4] has emerged as a favored choice. Serving as an industry-grade middleware, it aids in building robust computational robotics graphs, reinforcing the idea that robotics is more than just individual algorithms. The growing dependency on ROS 2 [144], combined with the computational improvements offered by hardware acceleration, accentuates the community’s demand for a standardized, industry-grade benchmark to evaluate varied hardware solutions. Recently, there has been a plethora of workshops and tutorials focusing on benchmarking robotics applications [145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155], and while benchmarks for specific robotics algorithms [156, 157] and certain end-to-end robotic applications, such as drones [158, 159, 160, 161], do exist, the nuances of analyzing general ROS 2 computational graphs on heterogeneous hardware is yet to be fully understood.

In this chapter, we introduce *RobotPerf*, an open-source and community-driven benchmarking tool designed to assess the performance of robotic computing systems in a standardized, architecture-neutral, and reproducible way, accommodating the various combinations of hardware and software in different robotic platforms. Figure 5.1 presents a high level overview of *RobotPerf*, which targets industry-grade real-time systems with complex and extensible computation graphs using the Robot Operating System (ROS 2) as its common baseline. Emphasizing adaptability, porta-

## 5. The Cost of Security: Benchmarking Robot Computations

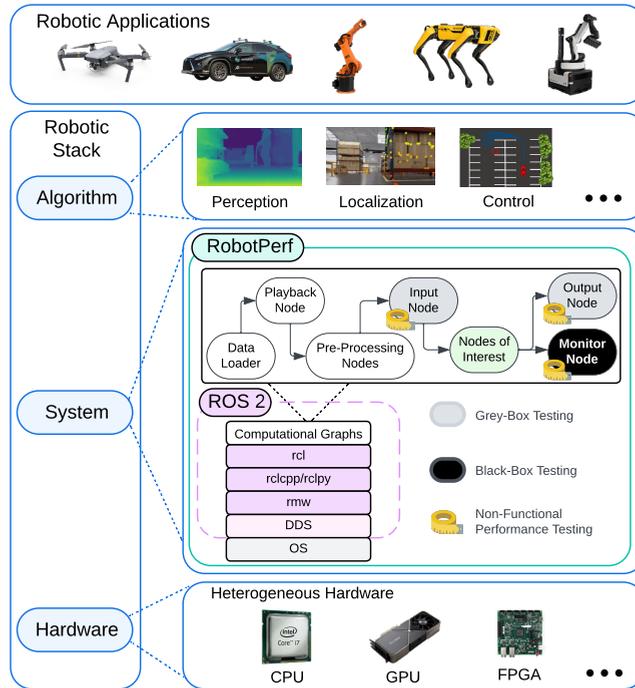


Figure 5.1: A high level overview of RobotPerf, reproduced from [133]

bility, and a community-driven approach, RobotPerf aims to provide fair comparisons of ROS 2 computational graphs across CPUs, GPUs, FPGAs and other accelerators. It focuses on evaluating robotic workloads in the form of ROS 2 computational graphs on a wide array of hardware setups, encompassing a complete robotics pipeline and emphasizing real-time critical metrics. The framework incorporates two distinct benchmarking methodologies that utilize various forms of instrumentation and ROS *nodes* to capture critical metrics in robotic systems. These approaches are: black-box testing, which measures performance by eliminating upper layers and replacing them with a test application, and grey-box testing, an application-specific measure that observes internal system states with minimal interference. The framework is user-friendly, easily extendable for evaluating custom ROS 2 computational graphs, and collaborates with major hardware ac-

	Characteristics						
	Real-time Performance Metrics	Spans Multiple Pipeline Categories	Evaluation on Heterogeneous Hardware	Integration with ROS/ROS 2 Framework	Functional Performance Testing	Non-functional Performance Testing	Community Led
OMPL Benchmark [162]	✓	✗	✗	✗	✗	✓	✗
MotionBenchMaker [163]	✓	✗	✗	✗	✓	✓	✗
OpenCollBench [164]	✗	✗	✓	✗	✓	✗	✗
BARN [165]	✗	✗	✗	✓	✓	✗	✗
DynaBARN [166]	✓	✗	✗	✓	✓	✗	✗
MAVBench [158]	✓	✓	✓	✓	✓	✓	✗
Bench-MR [167]	✓	✗	✗	✗	✓	✗	✗
RTRBench [156]	✓	✓	✗	✗	✗	✓	✗
<b>RobotPerf (ours)</b>	✓	✓	✓	✓	✗	✓	✓

Table 5.1: Comparative evaluation of representative existing robotics benchmarks with RobotPerf across essential characteristics for robotic systems.

celeration vendors for a standardized benchmarking approach. It aims to foster research and innovation as an open-source project. We validate the framework’s capabilities by conducting benchmarks on diverse hardware platforms, including CPUs, GPUs, and FPGAs, thereby showcasing RobotPerf’s utility in drawing valuable performance insights.

RobotPerf’s source code and documentation are available at <https://github.com/robotperf/benchmarks> and its methodologies are currently being used in industry to benchmark industry-strength, production-grade systems.

### 5.2 Background

#### Robotics Benchmarks

There has been much recent development of open-source robotics libraries and associated benchmarks demonstrating their performance as well as a plethora of workshops and tutorials focusing on benchmarking robotics applications [145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155]. However, most of these robotics benchmarks focus on algorithm correctness (*functional* testing) in the context of domain specific problems, as well as end-to-end latency on CPUs [162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179]. A few works also analyze some *non-functional* metrics, such as CPU performance benchmarks, to explore bottleneck behaviors in selected workloads [156, 157, 180].

Recent work has also explored the implications of operating systems and task schedulers on ROS 2 computational graph performance through benchmarking [181, 182, 183, 184, 185] as well as by optimizing the scheduling and communication layers of ROS and ROS 2 themselves [186, 187, 188, 189, 190, 191, 192, 193]. These works often focused on a specific context or (set of) performance counter(s).

Finally, previous work has leveraged hardware acceleration for select ROS Nodes and adaptive computing to optimize the ROS computational graphs [194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210]. However, these works do not provide comprehensive frameworks to quickly analyze and evaluate new heterogeneous computational graphs except for two works that are limited to the context of UAVs [158, 161].

Research efforts most closely related to our work include `ros2_tracing` [211] and `RobotCore` [139]. `ros2_tracing` provided instrumentation that demonstrated integration with the low-overhead `LTTng` tracer into ROS 2, while `RobotCore` illuminates the advantages of using vendor-specific tracing to complement `ros2_tracing` to assess the performance of hardware-accelerated ROS 2 Nodes. Building on these two specific foundational contributions, `RobotPerf` offers a comprehensive set of ROS 2 kernels spanning the robotics pipeline and evaluates them on diverse hardware.

Table 5.1 summarizes our unique contributions. It includes a selection of representative benchmarks from above and provides an evaluation of these benchmarks against `RobotPerf`, focusing on essential characteristics vital

for robotic systems. We note that while our current approach focuses only on non-functional performance benchmarking tests, RobotPerf’s architecture and methodology can be extended to also measure functional metrics.

## 5.3 RobotPerf: Principles & Methodology

RobotPerf is an open-source, industry-strength robotics benchmark for portability across heterogeneous hardware platforms. This section outlines the important design principles and describes the implementation methodology.

### 5.3.1 Non-Functional Performance Testing

Currently, RobotPerf specializes in non-functional performance testing, evaluating the efficiency and operational characteristics of robotic systems. Non-functional performance testing measures those aspects not belonging to the system’s functions, such as computational latency, memory consumption, and CPU usage. In contrast, traditional functional performance testing looks into the system’s specific tasks and function, verifying its effectiveness in its primary goals, like the accuracy of the control algorithm in following a planned robot’s path. While functional testing confirms a system performs its designated tasks correctly, non-functional testing ensures it operates efficiently and reliably.

### 5.3.2 ROS 2 Integration & Adaptability

RobotPerf is designed specifically to evaluate ROS 2 computational graphs, rather than focusing on independent robotic algorithms. We emphasize benchmarking *ROS 2 workloads* because the use of ROS 2 as middleware allows for the easy composition of complex robotic systems. This makes the benchmark versatile and well-suited for a wide range of robotic applications and enables industry, which is widely using ROS, to rapidly adopt RobotPerf.

### 5.3.3 Platform Independence & Portability

RobotPerf allows for the evaluation of benchmarks on a variety of hardware platforms, including general-purpose CPUs and GPUs, reconfigurable

## 5. The Cost of Security: Benchmarking Robot Computations

---

Criteria	Grey-Box	Black-Box
Precision	Utilizes tracers from in-code instrumentation.	Limited to ROS 2 message subscriptions.
Performance	Low overhead. Driven by kernelspace.	Restricted to ROS 2 message callbacks. Recorded by userspace processes.
Flexibility	Multiple event types.	Limited to message subscriptions in current implementation.
Portability	Requires a valid tracer. Standard format (CTF).	Standard ROS 2 APIs. Custom JSON format.
Ease of use	Requires code modifications and data postprocessing.	Tests unmodified software with minor node additions.
Real-Robots	Does not modify the computational graph.	Modifies the computational graph adding extra dataflow.

Table 5.2: Grey-box vs. black-box benchmarking trade-offs.

FPGAs, and specialized accelerators (e.g., ray tracing accelerators [212]). Benchmarking robotic workloads on heterogeneous platforms is vital to evaluate their respective capabilities and limitations. This facilitates optimizations for efficiency, speed, and adaptability, as well as fine-tuning of resource allocations, ensuring robust and responsive operation across diverse contexts.

### 5.3.4 Flexible Methodology

We offer grey-box and black-box testing methods to suit different needs. Black-box testing provides a quick-to-enable external perspective and measures performance by eliminating the layers above the layer-of-interest and replacing those with a specific test application. Grey-box testing provides more granularity and dives deeper into the internal workings of ROS 2, allowing users to generate more accurate measurements at the cost of increased engineering effort. As such, each method has its trade-offs, and providing both options enables users flexibility. We describe each method in more detail below and highlight takeaways in Table 5.2.

#### 5.3.4.1 Grey-Box Testing

Grey-box testing enables precise probe placement within a robot’s computational graph, generating a chronologically ordered log of critical

events using a tracer that could be proprietary or open source, such as LTTng [213]. As this approach is fully integrated with standard ROS 2 layers and tools through `ros2_tracing`, it incurs a minimal average latency of only 3.3  $\mu$ s [211], making it well-suited for real-time systems. With this approach, optionally, RobotPerf offers specialized input and output nodes that are positioned outside the nodes of interest to avoid the need to instrument them. These nodes generate the message tracepoints upon publish and subscribe events which are processed to calculate end-to-end latency.

#### 5.3.4.2 Black-Box Testing

The black-box methodology utilizes a user-level node called the `MonitorNode` to evaluate the performance of a ROS 2 node. The `MonitorNode` subscribes to the target node, recording the timestamp when each message is received. By accessing the propagated ID, the `MonitorNode` determines the end-to-end latency by comparing its timestamp against the `PlaybackNode`'s recorded timestamp for each message. While this approach does not need extra instrumentation, and is easier to implement, it offers a less detailed analysis and alters the computational graph by introducing new nodes and dataflow.

#### 5.3.5 Opaque Performance Tests

The requirement for packages to be instrumented directly within the source code poses a challenge to many benchmarking efforts. To overcome this hurdle, for most benchmarks, we refrain from altering the workloads of interest and, instead, utilize specialized input and output nodes positioned outside the primary nodes of concern. This setup allows for benchmarking without the need for direct instrumentation of the target layer. We term this methodology "opaque tests," a concept that RobotPerf adheres to when possible.

#### 5.3.6 Reproducibility & Consistency

To ensure consistent and reproducible evaluations, RobotPerf adheres to specific common robotic dataformats. In particular, it uses ROS 2 rosbags, including our own available at <https://github.com/robotperf/rosbags>, as well third-party bags (e.g., the r2b dataset [214]).

## 5. The Cost of Security: Benchmarking Robot Computations

---

To ensure consistent data loading and finer control over message delivery rates, we drew inspiration from [215]. Our computational graphs incorporate *modified and improved* `DataLoaderNode` and `PlaybackNode` implementations, which can be accessed at [https://github.com/robotperf/ros2\\_benchmark](https://github.com/robotperf/ros2_benchmark). These enhanced nodes offer improvements that report worst-case latency and enable the reporting of maximum latency, introduce the ability to profile power consumption and so forth.

### 5.3.7 Metrics

We focus on three key metrics: latency, throughput and power consumption including energy efficiency. Latency measures the time between the start and the completion of a task. Throughput measures the total amount of work done in a given time for a task. Power measures the electrical energy per unit of time consumed while executing a given task. Measuring energy efficiency (or performance-per-Watt) captures the total amount of work (relative to either throughput or latency) that can be delivered for every watt of power consumed and is directly related to the runtime of battery powered robots [158].

### 5.3.8 Current Benchmarks and Categories

RobotPerf beta [216] introduces benchmarks that cover the robotics pipeline from perception, to localization, to control, as well as dedicated benchmarks for manipulation. The full list of benchmarks in the beta release can be found in Table 5.1. Aligned with our principles defined above, each benchmark is a self-contained ROS 2 package which describes all dependencies (generally other ROS packages). To facilitate reproducibility, all benchmarks are designed to be built and run using the common ROS 2 development flows (ament build tools, colcon meta-build tools, etc.). Finally, so that the benchmarks can be easily consumed by other tools, a description of each benchmark, as well as its results, is defined in a machine-readable format. As such, accompanying the `package.xml` and `CMakeLists.txt` files required for all ROS packages, a YAML file named `benchmark.yaml` is in the root of each benchmark which describes the benchmark and includes accepted results.

### 5.3.9 Run Rules

To ensure the reliability and reproducibility of the performance data, we adhere to a stringent set of run rules. First, tests are performed in a controlled environment to ensure that performance data is not compromised by fluctuating external parameters. As per best practices recommended by `ros2_tracing` [211], we record and report settings like clock frequency and core count. Second, we look forward to the possibility of RobotPerf being embraced by the community and have results undergo peer review, which can contribute to enhancing reproducibility and accuracy. Finally, we aim to avoid overfitting to specific hardware setups or software configurations by encompassing a broad spectrum of test scenarios.

## 5.4 Evaluation

We conduct comprehensive benchmarking using RobotPerf to evaluate its capabilities on three key aspects vital for a robotics-focused computing benchmark. First, we validate the framework's capacity to provide comparative insights across divergent heterogeneous platforms from edge devices to server-class hardware. Second, we analyze the results to understand RobotPerf's ability to guide selection of the optimal hardware solution tailored to particular robotic workloads. Finally, we assess how effectively RobotPerf reveals the advantages conferred by hardware and software acceleration techniques relative to general-purpose alternatives. All of our results and source code can be found open-source at: <https://github.com/robotperf/benchmarks>.

### 5.4.1 Fair and Representative Assessment of Heterogeneity

Assessing hardware heterogeneity in robotic applications is imperative in the ever-evolving field of robotics. Different robotic workloads demand varying computational resources and efficiency levels. Therefore, comprehensively evaluating performance across diverse hardware platforms is crucial.

We evaluated the RobotPerf benchmarks over a wide list of hardware platforms, including general-purpose CPUs on edge devices (e.g., Qualcomm RB5), server-class CPUs (e.g., Intel i7-8700), and specialized hardware accelerators (e.g., AMD Kria KR260). Figure 5.2 illustrates benchmark

## 5. The Cost of Security: Benchmarking Robot Computations

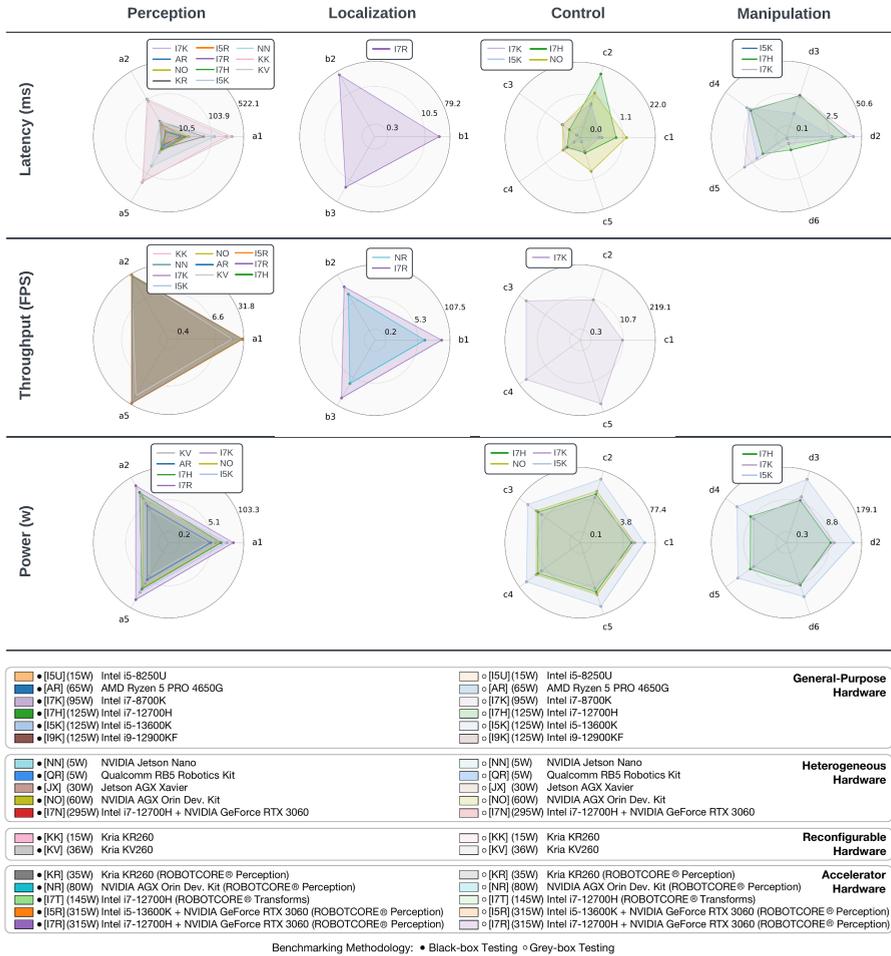


Figure 5.2: Benchmarking results on diverse hardware platforms across perception, localization, control, and manipulation workloads defined in RobotPerf beta Benchmarks. Reproduced from [133].

performance in robotics per category of workload (perception, localization, control, and manipulation) using radar plots, wherein the different hardware solutions are depicted together alongside different robotic workloads

per category. Each hardware solution is presented with a different color. Radar plots illustrate the latency, throughput, and power consumption for each hardware solution and workload, with reported values representing the maximum across a series of runs. Each hardware platform and performance testing procedure is delineated by a separate color, with darker colors representing Black-box testing and lighter colors Grey-box testing. In the figure’s key, the hardware platforms are categorized into four specific types: general-purpose hardware, heterogeneous hardware, reconfigurable hardware, and accelerator hardware. Within each category, the platforms are ranked based on their Thermal Design Power (TDP), which indicates the maximum power they can draw under load. The throughput values for manipulation tasks and power values for localization tasks have not been incorporated into the beta version of RobotPerf. As RobotPerf continues to evolve, more results will be added in subsequent iterations.

Given our ability to benchmark 18 platforms (bottom of Figure 5.2), RobotPerf is capable of benchmarking heterogeneous hardware platforms and workloads, paving the way for community-driven co-design and optimization of hardware and software.

## 5.4.2 Quantitative Approach to Hardware Selection

The rapid evolution and diversity of tasks in robotics means we need to have a meticulous and context-specific approach to computing hardware selection and optimization. A “one-size-fits-all” hardware strategy would be an easy default selection, but it fails to capitalize on the nuanced differences in workload demands across diverse facets like perception, localization, control, and manipulation, each exhibiting distinctive sensitivities to hardware capabilities. Therefore, a rigorous analysis, guided by tools like RobotPerf, becomes essential to pinpoint the most effective hardware configurations that align well with individual workload requirements.

The results in Figure 5.2 demonstrate the fallacy of a “one-size-fits-all” solution. For example, focusing in on the latency radar plot for control from Figure 5.2 (col 3, row 1), we see that the i7-12700H (I7H) outperforms the NVIDIA AGX Orin Dev. Kit (NO) on benchmarks C1, C3, C4, and C5, but is  $6.5\times$  slower on benchmark C2. As such, by analyzing data from the RobotPerf benchmarks, roboticists can better determine which hardware option best suits their needs given their specific workloads and performance requirements.

## 5. The Cost of Security: Benchmarking Robot Computations

---

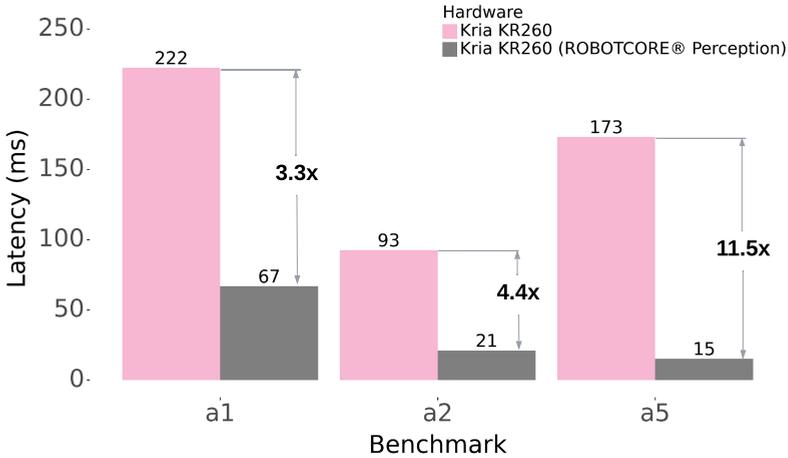


Figure 5.3: Benchmark comparison of perception latency (ms) on AMD’s Kria KR260 with and without the ROBOTCORE Perception accelerator. Reproduced from [133].

One general lesson learned while evaluating the data is that each workload is unique, making it hard to generalize across both benchmarks and categories. To that end, RobotPerf results help us understand how the use of various hardware solutions and dedicated domain-specific hardware accelerators significantly improves the performance.

### 5.4.3 Rigorous Assessment of Acceleration Benefits

In the rapidly advancing field of computing hardware, the optimization of algorithm implementations is a crucial factor in determining the success and efficiency of robotic applications. The need for an analytical tool, like RobotPerf, that facilitates the comparison of various algorithmic implementations on uniform hardware setups becomes important.

Figure 5.3 is a simplified version of Figure 5.2, depicting AMD’s Kria KR260 hardware solution in two forms: the usual hardware and a variant that leverages a domain-specific hardware accelerator (ROBOTCORE Perception, a soft-core running in the FPGA for accelerating perception robotic computations). The figure demonstrates that hardware acceleration can enable performance gains of as much as  $11.5\times$  (from 173 ms down to 15 ms for benchmark a5). We stress that the results obtained here should

be interpreted according to each end application and do not represent a generic recommendation on which hardware should be used. Other factors, including availability, the form factor, and community support, are relevant aspects to consider when selecting a hardware solution.

#### 5.4.4 Benchmarking the security impact in ROS 2 communications

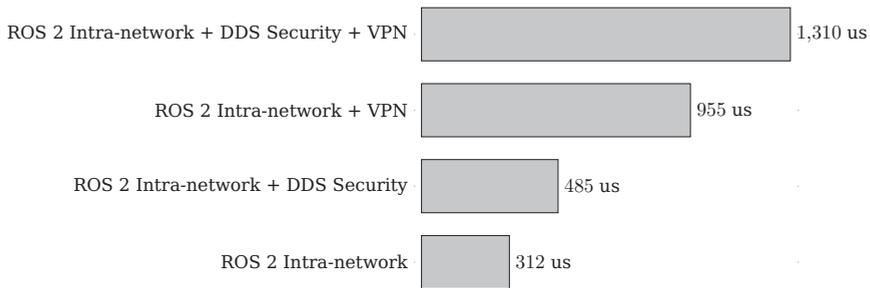


Figure 5.4: Mean latency of ROS 2 intra-network communications with various different security measures after 1000000 samples

Building upon the previous RobotPerf work and results, in this subsection, we delve into a series of benchmarks designed to evaluate the latency and performance impacts of implementing security protocols within ROS 2 environments. ROS 2, as a critical framework in the robotics domain, must balance the dual demands of robust security and efficient performance. Our benchmarks focus on various configurations, encompassing communications with no security (*ROS 2 Intra-network*), internal security measures within the communication middleware, specifically the DDS (Data Distribution Service) Security extensions (*ROS 2 Intra-network + DDS Security*), as well as the effects of tunneling communications through VPNs (*ROS 2 Intra-network + VPN*). Additionally, we explore the compounded impact when these two security layers are combined (*ROS 2 Intra-network + DDS Security + VPN*). Results are depicted in Figure 5.4.

The DDS Security extensions [18] are integral to securing ROS 2 communications, providing mechanisms for authentication, encryption, and access control. However, the computational overhead associated with

## 5. The Cost of Security: Benchmarking Robot Computations

---

these security features can potentially affect the system's responsiveness and throughput. Our communication benchmarks displayed in Figure 5.4 illustrate the DDS Security extensions overhead (*ROS 2 Intra-network + DDS Security*). When implemented, they lead to a mean latency which is **1.55** $\times$  higher than the no security baseline (*ROS 2 Intra-network*), offering insights into the trade-offs between security robustness and communication efficiency in ROS 2 environments.

Furthermore, the use of VPNs for remote operation and access of robotic systems introduces another layer of security, albeit with its own performance implications. VPNs can safeguard data transmission across various networks, crucial for operations like teleoperated surgery or remote industrial automation. Yet, the encryption and routing processes inherent in VPNs can introduce additional latency. Our benchmarking efforts demonstrate that tunneling ROS 2 over a modern VPN (*ROS 2 Intra-network + VPN*) lead to a mean latency which is **3.06** $\times$  higher than the no security baseline (*ROS 2 Intra-network*), providing a comprehensive understanding of the performance costs associated with secure, remote robotic operations over VPN.

Lastly, we investigate scenarios where DDS Security extensions and VPNs are used concurrently (*ROS 2 Intra-network + DDS Security + VPN*). This dual-layered security approach represents a robust defense mechanism, aligning with the zero-trust model. However, it also poses the greatest challenge in terms of computational overhead and potential latency increases. Our benchmarks hint towards a mean latency increase **4.19** $\times$  higher than the no security baseline (*ROS 2 Intra-network*). This result elucidates the cumulative impact of security measures, offering a holistic view of their implications on ROS 2's latency performance. This analysis will aid stakeholders in making informed decisions about security implementations, balancing the crucial aspects of security and efficiency in robotic systems.

### 5.5 Conclusion

RobotPerf represents an important step towards standardized benchmarking in robotics, including its security capabilities. With its comprehensive evaluation across the hardware/software stack and focus on industry-grade ROS 2 deployments, RobotPerf can pave the way for rigorous co-design of robotic hardware and algorithms. As RobotPerf matures with community

involvement, we expect it to compare CPU, GPU and FPGA, exploring their power consumption and flexibility in augmenting real-world robotic computations. With a standardized robotics benchmark as a focal point, the field can make rapid progress in delivering real-time capable systems that will unlock the true potential of robotics in real-world applications.

Through RobotPerf, we observe quantitatively significant overheads due to security implementations. Specifically, the DDS Security extensions increase latency by approximately  $1.55\times$  compared to a non-secure baseline. When implementing VPNs, this latency escalates to about  $3.06\times$  the baseline. Moreover, combining DDS Security with VPNs results in a latency increase of around  $4.19\times$ . These insights clearly indicate that VPNs constitute a major bottleneck in secure robotic communications. This finding highlights the urgency of developing solutions that can accelerate these security computations, thereby meeting the real-time demands of advanced robotic systems. Addressing this limitation is crucial, especially for applications requiring real-time responses. The next chapter aims to explore strategies to mitigate these overheads, focusing on the hardware acceleration enablement of ROS computations and communications to align with the real-time requirements of robotic systems.



## Chapter 6

# Hardware Accelerators in Robotics

**Contributions:** *Leading author at [139] and sole code contributor. Self-funded work. Minor help in the article visualizations and final edition from co-authors. This work is partially open source.*

---

In this chapter we embark on a critical exploration of how hardware accelerators and their corresponding acceleration kernels can address the computational and communication bottlenecks identified in robotic systems, specifically those related to security, as revealed in the previous chapter. The focus is on how integrating hardware accelerators—such as GPUs and FPGAs—into robotic architectures can drastically reduce latency and improve efficiency. This chapter delves into the potential of hardware accelerators to revolutionize robotics by enabling real-time processing capabilities, a vital aspect for many robotics applications. We discuss how these accelerators can be optimized for robotic tasks, including but not limited to, security protocol computations, thereby aligning with the real-time performance requirements crucial in robotics. The chapter particularly focuses on highlighting the innovative approaches and challenges involved in integrating these accelerators into the existing ROS 2 framework, shaping the future of robotics towards more efficient and secure systems.

## 6.1 Introduction

Recent work has seen an explosion of specialized robotics acceleration on nontraditional computing platforms such as GPUs, FPGAs, and ASICs [6, 135, 136, 138, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228,

229, 230, 231, 232, 233]. This has been sparked by the decline of Moore's Law and Dennard Scaling, which limits the performance of traditional CPU computing, positioning hardware acceleration as an emerging solution to achieve high performance and power efficiency in robotics applications.

However, this increased diversity of computing platforms leads to a dramatic growth in design space complexity that makes it difficult for users to easily deploy robotics applications on hardware accelerators without substantial expertise in each specific accelerator platform. The Open Computing Language (OpenCL) [234] is an effort to standardize hardware acceleration under a common language, but its adoption across silicon vendors has been uneven and support for it varies. As a result, current hardware acceleration usage is often tied to a particular vendor's solutions and platforms. This not only impedes interoperability and reuse of acceleration kernels, but presents yet another layer of complexity that users must overcome while implementing robotic systems that use acceleration kernels. A key obstacle is that each hardware acceleration vendor provides their own framework for development, but these are often disconnected from the common tools and libraries in robotics, and mostly aimed at hardware engineers, not roboticists.

To address this challenge, we present *RobotCore*, an *open architecture for hardware acceleration* that extends the Robot Operating System (ROS) [4]. ROS is widely used by academia and industry, and early work has demonstrated its potential for hardware-accelerated robotics applications [6, 209, 218]. We facilitate this emerging direction by implementing a vendor and platform-agnostic abstraction layer for hardware acceleration in robotics. Figure 6.1 depicts the open architecture for hardware acceleration in ROS 2, which extends the ROS 2 build system to support vendor and platform-agnostic deployment of robotics applications on accelerator hardware. The integrated tracing and benchmarking infrastructure enables users to analyze the system and make strategic design improvements to optimize performance. Starting with a popular robotics API as the foundation, our ROS 2-based acceleration architecture provides a common ground for both academic researchers and silicon vendors alike to develop specialized robotics acceleration kernels, and deploy them for easy usage by a large, established user base.

Once roboticists can easily harness hardware acceleration across multiple platforms, the next major challenge is *profiling and benchmarking the application*. Benchmarking is needed to determine the best mapping

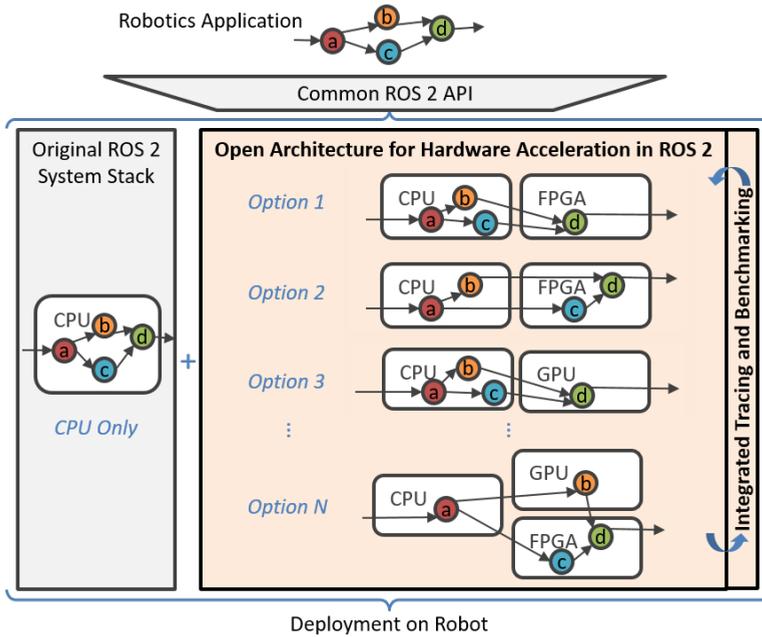


Figure 6.1: The open architecture for hardware acceleration in ROS 2, reproduced from [139].

of the robotics computational graph to the different hardware resources available to optimize overall robot system performance. This is a difficult task, however, since every application is different and deployment scenarios are widespread. Full end-to-end system analysis is required to understand how different implementation tradeoffs impact overall performance. To enable this analysis, we demonstrate how to leverage prior work [235] to benchmark accelerated ROS 2 kernels with a low-overhead framework for real-time tracing based on the Linux Tracing Toolkit next generation (LTTng) [213]. We demonstrate analysis of a case study deployment using CPU and FPGA nodes for a simple perception pipeline.

Using our framework and benchmarking, we diagnose that substantial latency bottlenecks in this computational graph come from inter-node interactions across ROS 2 layers in the CPU. We recognize this as an opportunity for design optimization in hardware accelerators, because

interaction with the CPU should not be necessary for dataflow between nodes co-located on the same non-CPU platform (e.g., FPGA).

Based on the benchmarking analysis, we demonstrate two novel separate paths toward *hardware acceleration*: (1) kernel fusion, and (2) improved message passing. Kernel fusion results in the highest speedup, an average of 26.96%, but it requires manual redesign of the underlying kernels. To avoid manual redesign entirely and improve design re-use and portability, we alternatively develop an intra-FPGA ROS 2 node communication queue template that leverages AXI4-Stream interfaces [236] and transfers data in a sequential streaming manner directly between acceleration kernels. Using this design pattern improves the overall inter-node performance in our computational graph by 24.42 % on average, while requiring no change in the accelerated kernels. This template extends to applications beyond our case study, since it can be reused for any ROS 2 inter- or intra-process communication by adapting its data types.

In summary, key contributions of this chapter are that we:

- Create a new open infrastructure to increase the performance of robotics applications by enabling *integration of hardware acceleration* into ROS 2 that is *flexible across accelerator platforms* (e.g., FPGAs, GPUs) and system deployments (e.g., edge devices, workstations, data centers, and cloud);
- Expose insights into how to optimize overall system-wide performance by extending and providing a template API for *low-overhead tracing and benchmarking framework* to analyze application performance across hardware accelerated ROS 2 computational graphs, laying foundation to analyze mixed-platform systems (e.g., combinations of CPU and FPGA-based nodes); and
- Increase ROS 2 node-to-node dataflow performance to achieve an average overall accelerator speedup of 24.42% over CPU in our experiments by designing a template for *intra-FPGA ROS 2 node communication queues*, based on insights uncovered using our open acceleration infrastructure and low-overhead benchmarking on a case study of a simple perception graph.

The core components of our architecture are disclosed under a commercially friendly open-source license and are available and maintained

at the ROS 2 Hardware Acceleration Working Group GitHub organization: <https://github.com/ros-acceleration>.

## 6.2 Background

### 6.2.1 ROS and ROS 2

The Robot Operating System (ROS) is an open-source collection of software frameworks and tools designed to provide a *structured communications layer* for robotics applications running on heterogenous computer hardware [4].

ROS applications are designed around event driven graphs of *Nodes* which communicate through *Messages* on various *Topics*, *Services*, and *Actions*. Each Node can be thought of as a software process which applies an algorithm to the input message and then broadcasts the resulting output message. By managing all inter-Node communications across abstraction layers (e.g., `rclcpp`, `rcl`, `rmw`), ROS simplifies the robotic system deployment process and enables roboticists to quickly develop and test new algorithms. ROS also provides substantial infrastructure to facilitate the automatic building, evaluation, and deployment of robotic systems, including dependency managers, package managers, build systems and tools, simulators, and visualizers.

ROS 2 is a re-design of ROS that modernizes and updates all of its components while adhering to its core design principles. ROS 2 provides a stronger partitioning of the communication middleware from the robotics logic, enabling more flexibility, scalability, and reliability [218]. ROS 2 also provides an updated build system, `ament`, and a new universal build tool, `colcon`. This provides a single simple interface for managing the building and deployment of complete robotics applications. Leveraging these tools, roboticists can write new algorithms and rely on ROS 2 to handle all lower level operations and middleware management.

### 6.2.2 Hardware Acceleration for ROS and ROS 2

There has been previous work that has focused on ways to accelerate robotics applications by developing tools and methodologies to help roboticists leverage hardware acceleration for select ROS Nodes and to optimize the ROS computational graph through adaptive computing [194,

## 6. Hardware Accelerators in Robotics

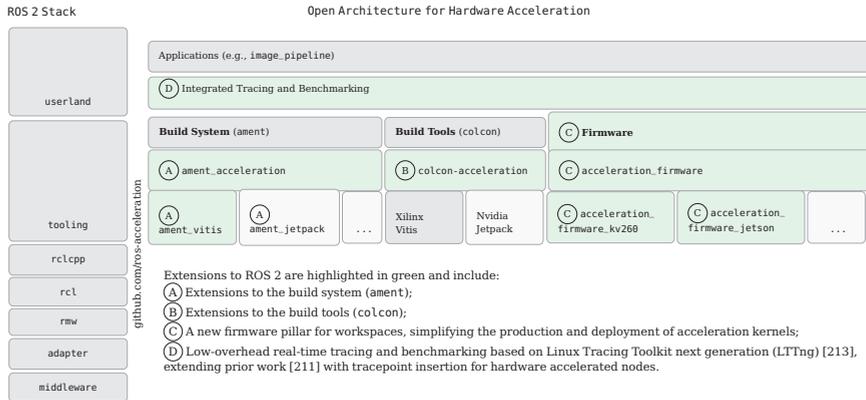


Figure 6.2: Overview of the components of the open architecture for hardware acceleration in ROS 2. Reproduced from [139].

195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209]. There has also been some work to accelerate the scheduling and communication layers used by ROS and ROS 2 [186, 187, 188, 189, 190, 191, 192, 193]. Unfortunately, the majority of these efforts assume an end-user has substantial experience with embedded systems and embedded hardware flows, or is customized to a specific hardware acceleration board or deployment scenario.

Our proposed open architecture takes a ROS-centric approach to integrate the hardware and embedded flows directly into the core ROS 2 ecosystem. This enables a separation between those who produce accelerated kernels and those who use them by providing end-users with a build and deployment experience for hardware accelerators similar to the standard, non-accelerated ROS 2 experience.

### 6.3 An Open Architecture for Hardware Acceleration in ROS 2

Our open architecture (Figure 6.2) extends the core ROS 2 build system and tools to provide platform-agnostic (i.e., supports edge, workstation, data center, or cloud targets) and technology-agnostic (i.e., supports FPGAs and GPUs), hardware-accelerated ROS 2 capabilities for roboticists. We:

A) extend the ROS 2 build system, `ament`; B) extend the ROS 2 meta build tool, `colcon`; and C) develop integrated ROS 2 firmware extensions. We also D) integrate a low-overhead tracing and benchmarking framework to enable the analysis of holistic application performance across ROS graphs. This section describes these extensions in detail.

### 6.3.1 Extending the ROS 2 Build System

The first pillar of our open architecture, Figure 6.2 (A), allows roboticists to generate acceleration kernels directly from the ROS 2 build system (`ament`) in the same way they generate CPU binaries. To do so, the `ament_acceleration` ROS 2 package and its extensions abstract the ROS build system from vendor-specific accelerators (e.g. FPGAs or GPUs), including their frameworks and software platforms. This allows the build system to easily support hardware acceleration across commercial solutions while using the same syntax, simplifying the work of ROS 2 package maintainers.

Under the hood, `ament_acceleration` abstracts away the corresponding vendor-specific firmware. For example, `ament_vitis`<sup>1</sup> relies on the proprietary Xilinx Vitis [237] and on the Xilinx Runtime (XRT) library [238]. This simplifies the creation of acceleration kernels and separates firmware concerns from algorithm development. This way, robotics engineers can focus on improving their computational graphs with a ROS-centric development flow. Separately, hardware experts, potentially sponsored by silicon vendors, can improve acceleration kernels for a particular commercial solution. Overall, these extensions help achieve the objective of simplifying the creation and integration of acceleration kernels from different vendors into ROS 2 computational graphs.

Figure 6.2 depicts the build system extensions showing how `ament_acceleration` abstracts the build system from vendor-specific solutions. As an example of an alternative acceleration technology supported, `ament_jetpack` is included and illustrates the integration of Nvidia JetPack [239].

---

<sup>1</sup>[github.com/ros-acceleration/ament\\_vitis](https://github.com/ros-acceleration/ament_vitis)

### 6.3.2 Extending the ROS 2 Build Tools

The second pillar of our open architecture, Figure 6.2 (B), extends the `colcon` ROS 2 meta build tool to integrate hardware acceleration flows into the ROS 2 Command Line Interface (CLI) commands. Examples of these extensions include the selection of the target accelerator and build-time through mixins, emulation capabilities to speed-up the development process and facilitate design without access to the real hardware, raw disk image production tools, and simplified configuration of hypervisors. These extensions are implemented by the `colcon-acceleration`<sup>2</sup> ROS 2 package. As in Section 6.3.1, `colcon_acceleration` further enables roboticists to leverage hardware accelerators while using standard ROS 2 commands and flows.

### 6.3.3 Adding Firmware Extensions

Represented by the abstract `acceleration_firmware` ROS package and its corresponding specializations (e.g. `acceleration_firmware_kv260`<sup>3</sup> for the Xilinx Kria KV260 board), the third pillar of our open architecture, Figure 6.2 (C), firmware extensions, are meant to provide firmware artifacts for each supported technology solution. This again simplifies the process for ROS package consumers and maintainers, and further aligns hardware acceleration workflows with typical ROS development flows. Each ROS 2 workspace can leverage multiple firmware packages, but can only use one at a time. As `colcon_acceleration` supports the selection of the active firmware in the ROS workspace, by separating the firmware out into their own packages, our open architecture enables silicon vendors to maintain an `acceleration_firmware_<solution>` package that automatically integrates into standard ROS 2 workflows.

### 6.3.4 Low-Overhead Real-Time Tracing & Benchmarking

In the context of hardware acceleration in robotics, it is fundamental to be able to inspect performance improvements. To that end, it is important to benchmark and trace the system. Benchmarking is the process of running a computer program to assess its relative performance, whereas

---

<sup>2</sup>[github.com/ros-acceleration/colcon-acceleration](https://github.com/ros-acceleration/colcon-acceleration)

<sup>3</sup>[github.com/ros-acceleration/acceleration\\_firmware\\_kv260](https://github.com/ros-acceleration/acceleration_firmware_kv260)

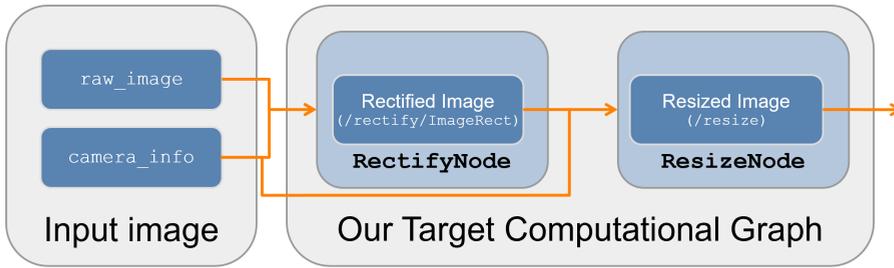


Figure 6.3: Computational graph of our case study perception application, reproduced from [139].

tracing is a technique used to understand what is happening in a system while it is running. Tracing helps determine which pieces of a Node are consuming more compute cycles or generating indeterminism, and are thereby good candidates for hardware acceleration. Benchmarking instead helps investigate the relative performance of an acceleration kernel versus its CPU scalar computing baseline. Similarly, benchmarking also helps with comparing acceleration kernels across different hardware acceleration technology solutions (e.g., Kria KV260 vs. Jetson Nano) and across kernel implementations within the same hardware acceleration technology solution.

In order to trace and evaluate the relative performance of both ROS 2 individual Nodes and complete computational graphs, we leverage Linux Tracing Toolkit next generation (LTTng [213]) for tracing and benchmarking, Figure 6.2 (D). Building upon prior work [211], LTTng provides a collection of flexible tracing tools and multipurpose instrumentation for ROS 2 that allow collecting runtime execution information in real-time in distributed systems using low-overhead tracers. For example, when enabling all ROS 2 instrumentation, end-to-end message latency overhead is below 5.5us [211], making it suitable for a wide variety of hardware acceleration use cases. Building on top of this foundation, we developed a tracing and benchmarking template that enables roboticists to easily instrument both their accelerated and non-accelerated code in a vendor-agnostic manner. This infrastructure also lays a foundation for future integration with platform-specific performance counters and tracing tools that can extend analysis to more fine-grained introspection and profiling of the kernels running onboard an accelerator device.

## 6. Hardware Accelerators in Robotics



Figure 6.4: Tracepoints instrumented across ROS 2 abstraction layers on CPU for case study computational graph (Figure 6.3). Breakdown summary in Figure 6.5. Reproduced from [139].

### 6.4 Case Study: Accelerating ROS 2 Perception

For our case study, we trace, benchmark, and accelerate a subset of `image_pipeline` [240], one of the most popular packages in the ROS 2 ecosystem, and a core piece of the ROS perception stack. We compose a simple computational graph consisting of two nodes, `resize` and `rectify`, as shown in Figure 6.3. The computational graph of our case study perception application in Figure 6.3 uses the ROS `image_pipeline` package, containing two ROS 2 nodes: (1) `RectifyNode` subscribes to the `/camera/image_raw` and `/camera/camera_info` topics from Gazebo [241] and publishes a *rectified image* to (2) `ResizeNode`, which publishes the final *resized image*. We leverage our open architecture for hardware acceleration (Section 6.3) to benchmark, trace and accelerate our computational graph, comparing a CPU to an FPGA implementation. In this section we describe the methodology of our approach, and analyze our timing results, presenting a case study for how our open architecture can help enable hardware accelerated applications in ROS 2.

#### 6.4.1 Method

We propose the following steps to analyze a ROS 2 application and design appropriate acceleration: (i) instrument both the core components of ROS 2 and the target kernels; (ii) trace and benchmark the kernels on the CPU to establish a baseline; (iii) develop a hardware accelerated implementation

on alternate hardware (e.g., GPU, FPGA); and (iv) trace, benchmark against the CPU baseline, and improve the accelerated implementation.

Following this methodology, in our case study we begin by instrumenting both ROS 2 and our target kernels with LTTng probes. Reusing past work and probes [211] allows us to easily get a grasp of the dataflow interactions within `rmw`, `rcl`, and `rclcpp` ROS 2 layers. We then also instrument the `ResizeNode` and `RectifyNode` components of the `image_pipeline` package used in our case study. The relevant tracepoints placed in our computational graph across ROS 2 stack layers are listed in Figure 6.4 and 6.5 (full list in Pull Request 717 in the `image_pipeline` repository [240]). On the CPU, these tracepoints enable us to isolate the latency of computation within a node from the time it takes ROS 2 to package and pass information between nodes.

In the following sections we report timing results from using a Xilinx Kria® KV260 Vision AI Starter Kit [242], which has an onboard integrated Quad-core Arm® Cortex®-A53 CPU and an FPGA containing 256K System Logic Cells and 1.2K DSP Slices. All benchmark results report the mean value obtained from a 60 second continuous run of the computational graph. The FPGA kernels are synthesized, placed and routed with a 250MHz clock.

### 6.4.2 CPU-Only Tracing Results

Figure 6.4 demonstrates the results of instrumenting and tracing our target computational graph (Figure 6.3) across multiple ROS 2 stack layers on the CPU, and Figure 6.5 summarizes the breakdown of timing results across operations, establishing the CPU baseline for our application. The breakdown in Figure 6.5 shows the time taken to do the computations within each node, as well as the time taken by the ROS 2 lower-level message-passing system across the various abstraction layers. We find that the message-passing overhead in our application consumes more than 73.3% of the total time and is therefore a large bottleneck in the total computation time of the full graph. We next explore FPGA hardware acceleration options, comparing performance to the CPU baseline.

### 6.4.3 Accelerating and Benchmarking CPU & FPGA

In this section, we explore hardware acceleration options for an FPGA for our case study application (Figure 6.3). In Section 6.4.3.1, we first explore

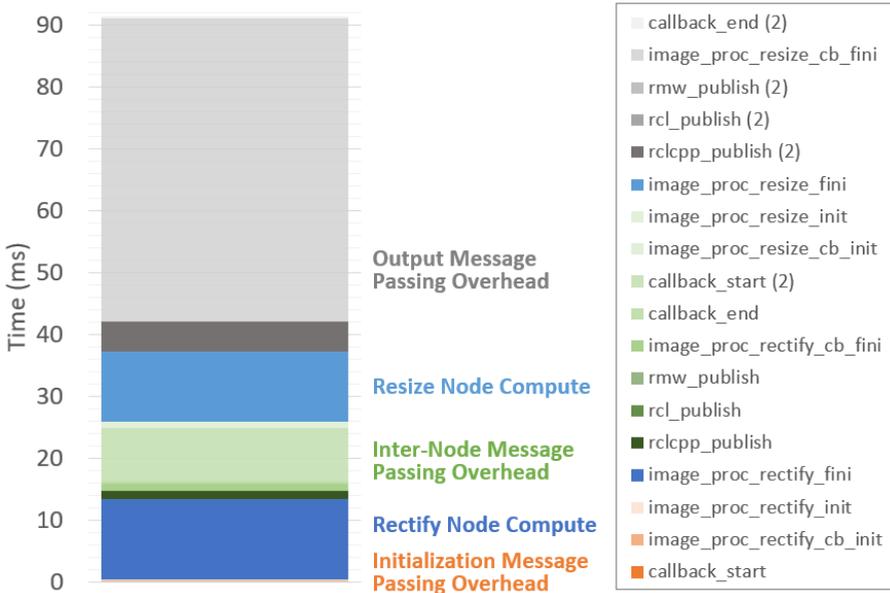


Figure 6.5: Breakdown of CPU runtime derived from tracing and benchmarking. Total computation time of our case study graph is dominated by message passing overheads, a bottleneck consuming over 73.3% of total runtime. Reproduced from [139].

hardware acceleration kernels for the core logic of each of the Nodes (*rectify* and *resize*), harnessing our open architecture for implementation. In Section 6.4.3.2, we then explore two different FPGA designs to accelerate the computational graph by optimizing dataflow interactions between FPGA-based nodes, addressing the ROS 2 *communication infrastructure* performance bottleneck revealed by the CPU baseline in Section 6.4.2.

### 6.4.3.1 Accelerating Nodes & Components on an FPGA

We first accelerate the computations at each one of the graph nodes. The RectifyNode and ResizeNode *Components* of Figure 6.3 are accelerated using Xilinx’s HLS, XRT, and OpenCL targeting the Kria KV260 <sup>4</sup>. Each

<sup>4</sup>[github.com/ros-acceleration/image\\_pipeline/blob/ros2/image\\_proc/src/{rectify,resize}\\_fpga.cpp](https://github.com/ros-acceleration/image_pipeline/blob/ros2/image_proc/src/{rectify,resize}_fpga.cpp)

ROS 2 *Component* has an associated acceleration kernel <sup>5</sup> that leverages the Vitis Vision Library, a computer vision library optimized for Xilinx silicon solutions and based on OpenCV APIs. These accelerated *Components* and their kernels easily integrate with the rest of the ROS meta-package through our open architecture (Figure 6.2), and are openly available to the public. Building the accelerators is abstracted away from roboticist end-users, and takes no significant additional effort than the standard build of the `image_pipeline`.

After benchmarking the accelerated *Components* using the trace points of Section 6.4.2, we observe an average 6.22% speedup in the total computation time of the perception pipeline when offloading tasks to the FPGA (see Figure 6.6). For this case study example, it is not surprising that accelerating the computational nodes and components alone only gives a modest performance increase because, as we saw in Section 6.4.2, the performance bottleneck in the baseline CPU system was communication overhead, not computation.

### 6.4.3.2 Accelerating the Computational Graph on an FPGA

In our case study application, message-passing overheads across the ROS 2 abstraction layers far outweigh other operations, so in this section we focus on optimizing these dataflows. Addressing performance bottlenecks in our system leads to overall lower computational graph latency, and to faster robots. To seize this acceleration opportunity in our case study example, we optimize the dataflow within the computational graph and across ROS 2 Nodes and Components through two different design approaches: (a) kernel fusion, and (b) dedicated streaming queues.

The speedup obtained by integrating both ROS *Components* on the FPGA into a *single unified kernel* is shown in Figure 6.6. The benefits of doing this are two-fold. First, we avoid any message-passing between the *Rectify* and *Resize Nodes' Components*. Second, we avoid the compute cycles wasted while memory is mapped back and forth between the host CPU and the FPGA. This results in an overall latency speedup of 26.96% over the CPU. In addition to speeding up the perception stage, another added benefit of this improvement is that such speedups make room for other robot tasks in a complete end-to-end system. Note, however, that this

---

<sup>5</sup>[github.com/ros-acceleration/image\\_pipeline/tree/ros2/image\\_proc/src/image\\_proc](https://github.com/ros-acceleration/image_pipeline/tree/ros2/image_proc/src/image_proc)

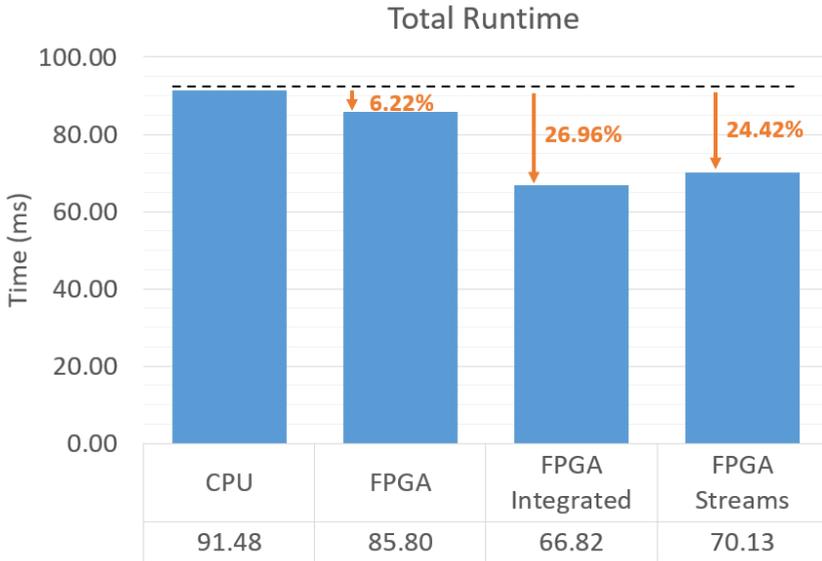


Figure 6.6: Total runtime of CPU baseline and FPGA, FPGA-Integrated, and FPGA-Streaming hardware-accelerated implementations of case study application. Acceleration enables up to 26.96% speedup over CPU. Reproduced from [139].

improvement required the construction of an entirely new ROS Node and unified acceleration kernel on the FPGA.

We then develop a template for an *accelerated ROS 2 message passing* interface on the FPGA. This interface is *Node* and *Component*-agnostic and can be leveraged by roboticists to accelerate the communication channels of any computational graph on an FPGA. This is done by leveraging an AXI4-Stream interface to create an intra-FPGA ROS 2 communication queue template which is then used to pass data across Nodes in the FPGA without sending messages to the CPU <sup>6</sup>. This allows us to completely bypass the original CPU-centric ROS 2 message-passing system and optimizes dataflow, achieving an overall latency improvement of 24.42% over the CPU in our application (see Figure 6.6).

<sup>6</sup>AXI4-Stream interfaces are data-type specific and as such our template may require type adaptations for other use cases depending on the Node-to-Node data interactions.

Based on these results, for this case study, we show that implementing FPGA-accelerated versions of key ROS 2 *Components* is easily feasible, and that addressing the right bottleneck is key to improving performance. Tracing and benchmarking the CPU baseline suggested that communication is the bottleneck in our case study. In fact, independent examination of, e.g., a single run of the fused-kernel accelerator using the Xilinx Vitis Analyzer, confirms that this is also the case on the FPGA—we note that integrating device-specific profiling tools into our foundational tracing infrastructure in future work can further automate this type of fine-grained introspection of kernels onboard accelerator devices. We can achieve overall performance improvements by either combining Nodes or streamlining intra-FPGA communication. While combining nodes may result in slightly higher performance, it is a much more labor-intensive design effort. By contrast, our accelerated intra-FPGA-Node communication queue template can be applied by any roboticist, to any computational graph.

## 6.5 Conclusion

In this chapter we presented a new open infrastructure to introduce hardware acceleration in ROS 2 in a scalable and technology-agnostic manner. Our architecture allows us to increase the performance of robotics applications through the *integration of hardware acceleration* with ROS 2 APIs and its conventional flows. We do so by extending ROS 2 in a way that is *portable across accelerator platforms* (e.g., FPGAs, GPUs) and system deployments (e.g., edge devices, workstations, data centers, and cloud). We also present a *template for low-overhead tracing and benchmarking* to analyze performance across both hardware accelerated and standard ROS 2 computational graphs.

We use our open architecture and our tracing and benchmarking infrastructure to demonstrate a principled design methodology for ROS 2 hardware acceleration, exposing insights into how to optimize overall system-wide performance by analyzing a CPU baseline, and comparing accelerator design iterations to that original baseline. We examine a case study using the Xilinx Kria KV260 platform to demonstrate FPGA acceleration of one of the most popular packages in the ROS perception pipeline: `image_pipeline`. We first demonstrate a modest performance speedup of 6.22% from offloading perception tasks to the FPGA, and then increased speedup by additionally addressing the communication

overheads that we identified as bottlenecks by analyzing our CPU baseline. We achieved a speedup of 26.96% from re-architecting the graph to combine nodes and avoid inter-FPGA-node communication delays inflicted by interactions with the CPU, but this approach requires substantial effort from users to re-architect their graphs. Instead, to avoid this overhead and stay in alignment with the ROS 2 programming model, we then design a novel *template for intra-FPGA ROS 2 Node communication queues* that allows ROS *Nodes* and *Components* to deliver faster dataflows, achieving a 24.42% speedup over a CPU without excessive manual per-kernel design effort.

We contribute our open architecture to the ROS community, so that future work can use our infrastructure and extend to new applications beyond our case study example. Promising directions for future work include: benchmarking computational graphs with other hardware solutions (e.g., GPUs) to establish consistent cross-accelerator comparisons; extending our tracing and benchmarking approach to include additional tracing information (e.g., profiling within FPGA or GPU devices) for more fine-grained introspection of kernels running onboard accelerators; and applying our open architecture and analysis to other ROS 2 packages.

Our code is disclosed under a commercially friendly open-source license and is available and maintained at the ROS 2 Hardware Acceleration Working Group GitHub organization: <https://github.com/ros-acceleration>. This work is being further integrated into the ROS ecosystem through a community standardization effort, REP-2008 [243].

### 6.6 Future work

The promising outcomes of this chapter lay the groundwork for the development of dedicated *robot security accelerators*. Future initiatives will concentrate on commercializing these innovations. Preliminary findings suggest that our security accelerator prototypes could significantly enhance intra-network ROS 2 communications, potentially reducing average networking latency to single-digit microseconds, which is over  $100\times$  faster than current software implementations on CPUs. Particularly in worst-case scenarios, these accelerators maintain deterministic and isochronous performance, a stark contrast to the higher latencies observed with CPU-based software implementations. The potential for a maximum latency speedup exceeding  $100,000\times$  faster communications and security

computations marks a significant leap forward in the field of robotics. Due to the commercial potential of these developments, a detailed discussion is outside the scope of this research, but these early results are indicative of the transformative impact these accelerators could have on the future of robotic communications.



Part IV

# **Cyber-attacking robots**



## Chapter 7

# Attacking Robot Software

**Contributions:** *One of the top two contributors of [97] and core contributor to software, methodology and research results. Overall research conducted over a 6-month period, with other leading security researchers from industry participating. Produced various open source security package dissectors (e.g. this one, and this other one). All findings have been publicly disclosed.*

---

This chapter delves into the security analysis of robot software and particularly, the Data Distribution Service (DDS) [21] standard, a middleware pivotal in operating a wide range of robotic systems and devices, including ROS 2 (DDS is the default communication middleware of ROS 2). The summarized investigation presented herein is the result of a multi-year research effort involving various researchers [97] and led to uncover multiple vulnerabilities in DDS, leading to the issuance of various new CVE IDs for common DDS implementations and even in the standard specifications. An extensive scanning exercise revealed numerous public-facing DDS services across various industries, some affected by these CVEs, highlighting significant security risks due to their unintended public exposure. We emphasize the need for continuous security testing of DDS and similar technologies, offering both immediate and long-term recommendations for secure integration and advocating for robust supply-chain management and ongoing security practices.

### 7.1 Introduction

Even within the industry, a big percentage of practitioners are unaware that the Data Distribution Service (DDS) drives systems such as industrial robots (manipulators), railways, autonomous cars, airports, surgical robots, spacecraft, diagnostic imaging machines, luggage handling, industrial robots, military tanks, and frigates, among others. It has been in use for about a decade, and its adoption continues to steadily increase.

We discovered and reported vulnerabilities in DDS that warranted new CVE IDs: Five with a score of greater or equal than 7.0, four with a score of > 8.5, one vulnerability in the standard specifications, and other deployment issues in the DDS software ecosystem (including a fully open production system).

Successful exploitation of these vulnerabilities can facilitate initial access (MITRE ATT&CK [244] Technique ID TA0108) via exploitation of remote services (T0866, T0886) or supply chain compromise (T0862), and allow the attacker to perform discovery (TA0102, T0856) by abusing the discovery protocol. The consequences of successful exploitation, in any of the critical sectors where DDS is used, range from denial of service (T0814) via brute forcing (T0806), to loss of control (T0827), or loss of safety (T0880). The DDS protocol itself can also be abused to create an efficient command and control channel (T0869). Based on our analysis, we recommend mitigations such as vulnerability scanning (ATT&CK Mitigation ID M1016), network intrusion prevention (M1031), network segmentation (M1030), filter network traffic (M1037), execution prevention (M1038), and auditing (M1047).

### Findings in Brief

Given this technology's versatility, we analyzed and discovered multiple security vulnerabilities, resulting in 13 new CVE IDs for the six most common DDS implementations. This includes one vulnerability in the standard specifications and other deployment issues in the DDS software ecosystem (including a fully open production system). Some of these vulnerabilities have been patched or mitigated by the vendors since we reported them. Others remain exposed and can still be abused.

By measuring the exposure of DDS services, in one month we found over 600 distinct public-facing DDS services in 34 countries affecting

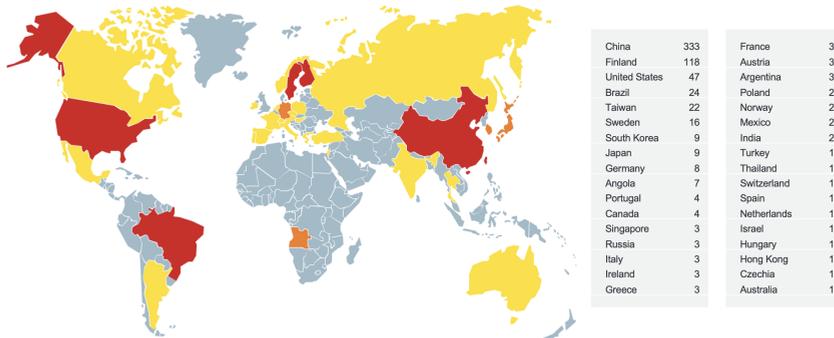


Figure 7.1: We found exposed DDS systems in 34 countries [97], including vulnerable ones, identified via distinct IPs leaking data. Color code hints about criticality by vulnerable targets: red for most, yellow/grey for least. Reproduced from [97].

100 organizations via 89 internet service providers (ISPs). Of the DDS implementations by seven distinct vendors (one of which we were initially unaware of), 202 leaked private IP addresses (referring to internal network architecture details), and seven supposedly secret URLs. Some of these IP addresses expose unpatched or outdated DDS implementations, which are affected by some of the vulnerabilities that we’ve discovered and disclosed in November 2021.

During our research, we interviewed key DDS users and system integrators to collect their feedback on our findings and the importance of DDS for innovation in their respective sectors. In this chapter, we analyze and discuss the specifications of DDS and the six most actively developed implementations maintained by certified vendors and with millions of deployments worldwide. We also released an open-source software: a Scapy-based dissector and several fuzzing harnesses for three open DDS implementations.

## 7.2 Background

DDS is a standardized middleware software based on the publish-subscribe paradigm, helping the development of middleware layers for machine-to-machine communication. This software is integral especially to embedded

## 7. Attacking Robot Software

systems or applications with real-time requirements. Maintained by the Object Management Group (OMG), DDS is used in all classes of critical applications to implement a reliable communication layer between sensors, controllers, and actuators.

### About real-time in DDS and robotics

*Real-time* which in robotics is interpreted as meeting time deadlines in their computations, is used in a generic and non-strict (hard real-time) manner in DDS specifications. We were unable to find timing guarantees (hard real-time, firm real-time, or soft real-time) provided in any of the reviewed documents. For the purposes of this research, we conclude that DDS targets remote soft real-time communications at best, leaving firm and hard real-time interactions to other technologies.

DDS is at the beginning of the software supply chain, making it easy to lose track of and is an attractive target for attackers. Between 2020 and 2021, 66% of attacks focused on the suppliers' codes [245]. While we were in the process of doing this research, we encountered an exposed source-code repository hosting a proprietary implementation of DDS. Left open, this would have let an attacker infect the source code (MITRE ATT&CK T0873, T0839).

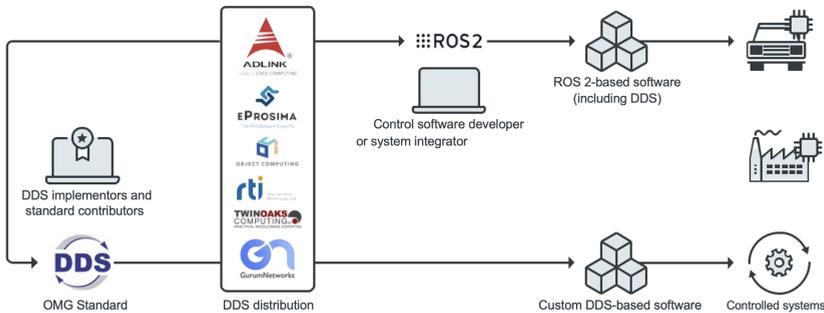


Figure 7.2: DDS is a standardized software library used for software-based controlled systems, directly or via ROS 2. Reproduced from [97].

Notably, the following companies and agencies use DDS (note that this is not an exhaustive list of currently using the technology):

- Military robots (e.g. legged robots from Ghost Robotics)
- National Aeronautics and Space Administration (NASA) at the Kennedy Space Center
- Siemens in wind power plants
- Volkswagen and Bosch for autonomous valet parking systems
- Nav Canada and European CoFlight for air-traffic control

DDS is the foundation of other industry standards such as OpenFMB [246] for smart-grid applications and Adaptive AUTOSAR [247], among other sectors that we identify in the next section. The Robot Operating System 2 (ROS 2), which is the de facto standard operating system for robotics and automation and as previously mentioned, also uses DDS as the default middleware. We also noted that, according to a confidential document leaked online, NVIDIA has listed DDS as a strategic tool for system-virtualization and cloud-computing applications, mainly for exchanging data within and across virtual machines.

### 7.3 DDS and Real-Time Publish-Subscribe (RTPS) Packets

There are many software-based controlled systems in the world that connect sensors, actuators, and controlling and monitoring applications. DDS was invented for such systems, with a strong focus on interoperability and fault tolerance. It is optimized for publish-subscribe and peer-to-peer applications as most applications can't afford a single point of failure. The middleware relies on multicast (group communication or data transmission to multiple recipients) for discovery, allowing everything to run without needing initial configurations.

From a software developer standpoint, DDS is a communication middleware that facilitates interoperability of processes across machines in all main programming languages. From another viewpoint, DDS is a data-centric, publish-subscribe communication protocol that allows developers to build a flexible shared data “space” or “bus” for virtually any application that requires two or more nodes to exchange typed data. The DDS layer is encapsulated into real-time publish-subscribe (RTPS) packets, which for

## 7. Attacking Robot Software

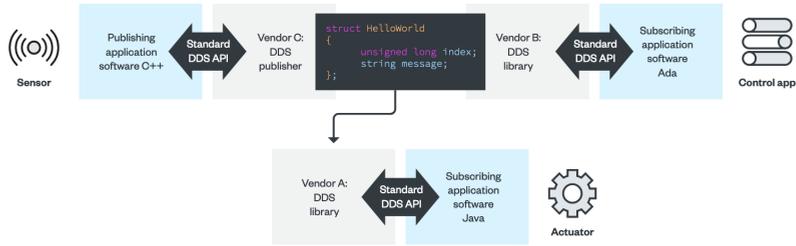


Figure 7.3: Simplified software-based control system with actuators, controller, sensors, communicating by exchanging data over DDS. Reproduced from [97].

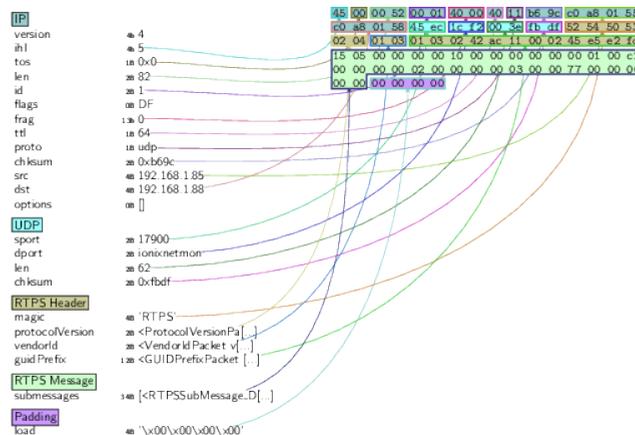


Figure 7.4: An example of a dissected RTPS message with a DATA submessage. Reproduced from [97].

now can be considered as a collection of sub-messages (such as timestamp, discovery, data, and security metadata), as shown in Figure 5. Given the strong dependency between DDS and RTPS, we focused our research on RTPS for increased generality. Because of its flexibility, DDS and its underlying layers do not come as a ready-to-use, off-the-shelf product like other middleware tools (such as Message Queuing Telemetry Transport or MQTT). Rather, DDS is a programming library that developers use to build custom middleware protocols with advanced features such as custom data

types, quality of service (QoS) policies, network partitioning, authentication, and encryption.

*Understanding DDS as an robotics databus (as opposed to a database)*

DDS can be understood as databus for robotic solutions. A databus is a data-centric software framework for distributing and managing real-time data in intelligent distributed systems. In intelligent distributed systems, managing dataflow is critically important. The databus — designed specifically to manage dataflow in intelligent distributed systems — simplifies application and integration logic with a powerful data-centric paradigm.

**Research Scope: RTPS, DDS, and Robot Operating System 2 (ROS 2)**

Product name	Developer	HQ region	Open source	Core language	Developed Since
Fast-DDS	eProsima	EMEA	Apache License 2.0	C++	2014
Cyclone DDS	Eclipse Foundation project, driven by ADLINK	EMEA	Eclipse Public License 2.0 and Eclipse Development License 1.0	C	2011
OpenDDS	OCI	NABU	Custom	C++	2005
Connexxt DDS	RTI	NABU	Extensions are open source	C++	2005 (NDDS – 1995)
CoreDX DDS	TwinOaks	NABU	Not open source	C	2009
Gurum DDS	GurumNetworks	APAC	Not open source	C	

Table 7.1: A list of DDS implementations that we analyzed in this research. Reproduced from [97].

In addition to the DDS standard specifications, we focused our investigation on the six DDS implementations listed in Table 7.1. These implementations are used globally and have customers and users in the critical sectors identified in the same table. Because DDS depends on RTPS as a lower-layer standard protocol, each DDS implementation ships with its own RTPS implementation. In other words, DDS data is contained as a sub-message within RTPS, thereby focusing on both protocols. To make sure we highlight the criticality of this for robotics, again, the Robot

Operating System 2 (ROS 2), which is the default standard meta-operating system for robotics and automation, has DDS as its default middleware. For this reason, the impact of each vulnerability extends beyond DDS alone, and includes all ROS 2 instances.

### 7.4 Research Methodology and Technical Details

Refer to [97] for details on the final findings derived from this research. This subsection instead discusses the technical details of how those findings were encountered. The offensive methodology explained here applies to other (software) robotic endeavours. Figure 7.5 provides an overview of the research methodology we followed. We analyzed each DDS implementation from two main angles: network and source — or binary — code. We wanted to have a deep understanding of the low-level details of the RTPS network layer to craft arbitrary test packets, and prove that it is possible for an attacker to mass scan a network and map the attack surface. In a parallel investigation, we manually studied the original or decompiled source code, with the main goal to find good fuzz targets (i.e., functions that receive and process untrustworthy data like network packets).

#### 7.4.1 A New Scapy Layer to Dissect and Forge RTPS and DDS Data

Although Wireshark already includes an RTPS dissection plugin, we needed something more scriptable. Since we spent some time manually crafting RTPS packets at the beginning, we decided to develop a RTPS Scapy-based dissector. We released the resulting Scapy layer as open-source code under the GNU General Public License v2.0.74 which is available ever since  <https://github.com/secdev/scapy/pull/3403>.

##### 7.4.1.1 Crafting RTPS probes with Scapy

Without going into the details of our Scapy RTPS implementation, note that it can be used to programmatically create RTPS packets by writing Python code, like any other Scapy layer. In practice, that's seldom what a researcher would do, especially for "thick" protocols with lots of options.

Instead, the developer's typical workflow can be:



```
pkt = RTPS(  
    protocolVersion=ProtocolVersionPacket(major=2, minor=1),  
    vendorId=VendorIdPacket(vendor_id=b"\x01\x10"),  
    guidPrefix=GUIDPrefixPacket(  
        hostId=17849486, appId=752113735, instanceId=4200214739  
    ),  
    magic=b"RTPS",  
) / RTPMessage(  
    submessages=[  
        RTPSSubMessage_INFO_TS(  
            submessageId=9,  
            submessageFlags=1,  
            octetsToNextHeader=8,  
            ts_seconds=1635160430,  
            ts_fraction=3848061961,  
        ),  
        PID_DEFAULT_UNICAST_LOCATOR(  
            parameterId=49,  
            parameterLength=24,  
            locator=LocatorPacket(  
                locatorKind=16777216,  
                port=port,  
                address=ip,  
            ),  
        ),  
        PID_METATRAFFIC_UNICAST_LOCATOR(  
            parameterId=50,  
            locator=LocatorPacket(  
                locator=LocatorPacket(  
                    locatorKind=16777216,  
                    port=port,  
                    address=ip  
                ),  
            ),  
        ),  
    ],  
)
```

Figure 7.6: With the Scapy RTPS layer, a developer can create arbitrarily complex (and unexpected but valid) RTPS packets. Reproduced from [97].

RTPS class, which will automatically dissect it.

- Generate Python code automatically. Use Scapy’s built-in `.command()` method to output the Python code that will declaratively generate the packet that has just been dissected. If necessary, modify the packet so obtained (as exemplified in Figure 7.11).
- Test modified packet against the endpoint. Either use Scapy’s built-in `send()/sendp()` functions or Python’s `socket` module to send the packet to a target DDS endpoint and check if it triggers the desired behavior.

The following section briefly describes how, almost by accident, we discovered the amplification vulnerability while dissecting and modifying packets during the early stages of the development of our Scapy RTPS layer.

### 7.4.1.2 Finding the Amplification Vulnerability

Although network fuzzing via Scapy was not directly effective in our research, creating a Scapy layer helped and motivated us to investigate all the RTPS packets’ fields in depth. The activity led us to find the amplification vulnerability (CVE-2021-38425, CVE-2021-38429, CVE-2021-38487, CVE-2021-43547). In the long run, we recommend that developers and users leverage our Scapy RTPS layer — or similar libraries — as a building block for building continuous network fuzzers for RTPS and DDS.

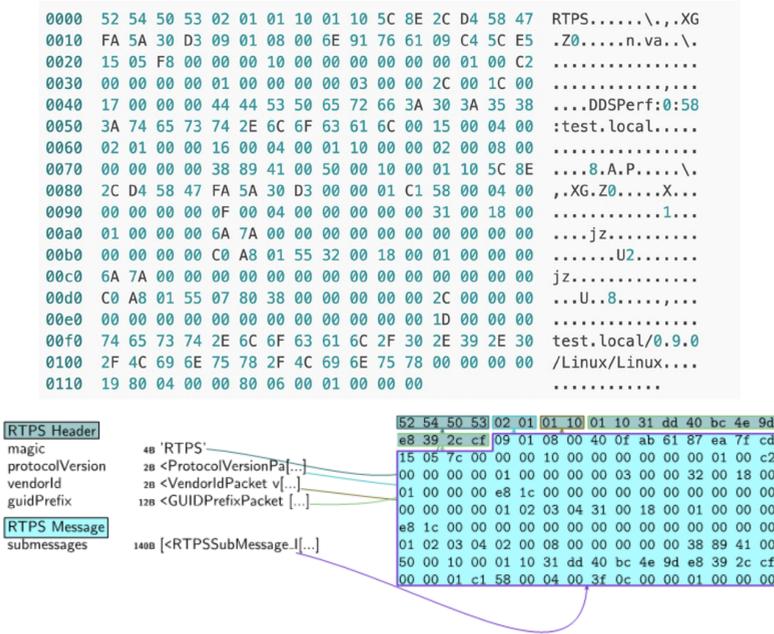


Figure 7.7: The UDP payload contains the RTPS header and subsequent data. Reproduced from [97].

The goal of the RTPS discovery phase is to send “probe” packets (e.g., to multicast addresses) and wait for responses from new locators. Locators could be IP-port pairs (see the `PID_DEFAULT_UNICAST_LOCATOR` in Figure 7.6, right side of the screenshot) or memory offsets in a shared-memory transport. Before reading the specifications in depth, we assumed that an RTPS discovery packet would allow us to restrict the locator to the IP addresses within the network the machine is connected to, and would not blindly send RTPS data to any IP-port found in the locator field. On a second read, however, this is exactly how discovery works by design.

We discovered this by setting the `PID_DEFAULT_UNICAST_LOCATOR` to the first IP address that came to mind (in this case, the Google DNS, because it’s easy to type at “8.8.8.8”). Almost immediately, a flow of outbound packets came from the DDS node, as shown in Figure 7.7 and 7.9.

## 7. Attacking Robot Software

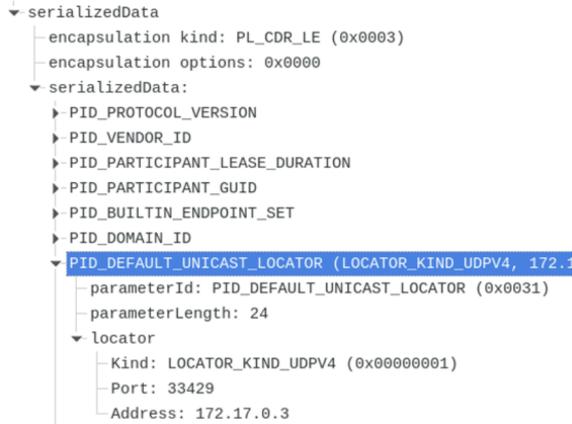


Figure 7.8: In case of UDP or TCP transport, the locator is the IP-port pair. Reproduced from [97].

172.17.0.4	172.17.0.3	RTPS	350 INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	366 INFO_DST, INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	162 INFO_DST, ACKNACK, ACKNACK, ACKNACK
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.1	172.17.255.255	UDP	86 57621 → 57621 Len=44
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	366 INFO_DST, INFO_TS, DATA(p)
172.17.0.3	8.8.8.8	RTPS	174 INFO_DST, HEARTBEAT, HEARTBEAT, HEARTBEAT
172.17.0.3	8.8.8.8	RTPS	162 INFO_DST, ACKNACK, ACKNACK, ACKNACK

Figure 7.9: We found the amplification vulnerability almost by accident, by setting the PID\_DEFAULT\_UNICAST\_LOCATOR to the first IP address that came to mind and easy to type. Reproduced from [97].

### 7.4.2 Source-code and Binary Fuzzing

Of all the implementation vulnerabilities that we disclosed for this research, all but three have been found through source-code or binary fuzzing, and three through scripting a file-format input mutator (RADAMSA). There

are many fuzzing tools freely available to researchers, and we chose one based on what has been used successfully for years by the largest public fuzzing platform (Google OSS-Fuzz), which uses a combination of AFL++, libFuzzer, and Honggfuzz. Although the choice of the specific tool can influence the efficiency of a fuzzing campaign, we focused our attention on the most important piece: finding good fuzz targets and writing good fuzzing harnesses.

#### 7.4.2.1 Source-code Fuzzing with AFL++ and libFuzzer

We used AFL++ for fuzzing with multiple sanitizers in LLVM. AFL++ requires the project compile with the latest version of LLVM and the build system of some DDS implementations required some work. Aside from this, most of the effort in this phase went into finding the right fuzz target and implementing a harness while keeping the code deterministic (for example, no threading).

From the high-level viewpoint depicted in Figure 7.11 and 7.14, we were interested in finding the most self-contained function in charge of processing data coming from the network. We found a repeating pattern in all the DDS implementations: upon receiving network data (i.e., `recv()` or some abstraction on top of it), there are one or more deserialization functions in which we likely find a switch-case control structure, used to dispatch the RTPS sub-message IDs to the right routine.

Given the importance of finding the right fuzz target, we dedicate the remainder of this section to showcase some examples of fuzz targets.

We began with the supplied Docker images or make files to compile example programs, run them under GDB, and inspect debug traces, like exemplified for OpenDDS in Figure 7.11.

By following the function calls with the aid of the code analyzer part of Visual Studio Code, we were able to see that all three DDS implementations were using very similar procedures for deserializing network payloads. In particular, we found that they all had a switch-case to handle the RTPS sub-message types, as exemplified for OpenDDS in Figure 7.15.

In some cases, we adjusted the source code right before the beginning of the de-serialization to dump the binary data being passed to the first function. This was useful to confirm that it was the network payload that we expected the function to receive, as exemplified for Cyclone DDS in

## 7. Attacking Robot Software

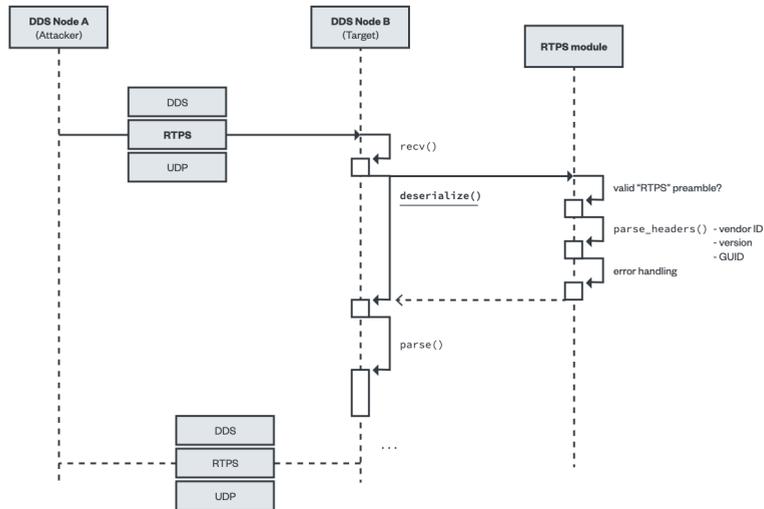


Figure 7.10: Abstract representation of the data flow in a typical DDS/RTSPS message exchange. From a fuzzing perspective, the `deserialize()` step is the fuzz target. Reproduced from [97].

```
[gdb] bt
#0  OpenDDS::DCPS::Serializer::ismemcpy (this=0x7ffc615e18a0, to=0x7ffc614dddc0 "%", from=0x612000000780 "\017", m=2) at DCPS/Serializer.cpp:374
#1  0x0000000004d3db in OpenDDS::DCPS::Serializer::doread (this=0x7ffc615e18a0, dest=0x7ffc614dddc0 "%", size=2, swaps=optimized out, offset=0)
#2  OpenDDS::DCPS::Serializer::buffer_read (this=optimized out, dest=optimized out, size=optimized out, swaps=optimized out) at /usr/local/
#3  0x00007fdaf4d1439b in OpenDDS::DCPS::operator>> (s=..., x=0x7ffc614dddc0: 119) at DCPS/Serializer.inl:1173
#4  0x00007fdaf16e0d2f in OpenDDS::DCPS::operator>> (outer_strm=..., uniw=...) at RtpsCoreTypeSupportImpl.cpp:11440
#5  0x00007fdaf16dff18 in OpenDDS::DCPS::operator>> (strm=..., seq=...) at RtpsCoreTypeSupportImpl.cpp:8985
#6  0x0000000004d051f in main (argc=optimized out, argv=optimized out) at test.cpp:106

if ((buff_.size() >= 4) && ACE_OS::memcmp(buff_.rd_ptr(), "RTSPS", 4) == 0) {
    RTPS::Message message;

    DCPS::Serializer ser(buff_, encoding_plain_native);
    Header header;
    if (!((ser >> header) {
        ACE_ERROR((LM_ERROR,
                    ACE_TEXT("(%p|%t) ERROR: Spdp::SpdpTransport::handle_input() - ")
                    ACE_TEXT("failed to deserialize RTPS header for SPDP(%i)"));
        return 0;
    }

bool MessageReceiver::checkRTSPSHeader(
    CORBA::Message_t* msg)
{
    //check and process the RTPS Header
    if (msg->buffer[0] != 'R' || msg->buffer[1] != 'T' ||
        msg->buffer[2] != 'P' || msg->buffer[3] != 'S')
    {
        logInfo(RTPS_MSG_IN, IDSTRING "Msg received with no RTPS in header
        return false;
    }
}

msg->pos += 4;
```

Figure 7.11: Starting from a debug trace, we found interesting functions and explored further by manually looking into the source code with the aid of Visual Studio Code engine. Reproduced from [97].

Figure 7.16.

```

while (buff_>length() > 3) {
    const char subm = buff_>rd_ptr()[0], flags = buff_>rd_ptr()[1];
    ser_swap_bytes((flags & FLAG_D) != ACE_CDR_BYTE_ORDER);
    const size_t start = buff_>length();
    CORBA::UShort submessageLength = 0;
    switch (subm) {
    case DATA: {
        DataSubmessage data;
        if (!ser >> data) {
            ACE_ERROR(LM_ERROR, ACE_TEXT("(NP)N) ERROR: Sdpd::SdpdTransport::handle_input() - ")
                ACE_TEXT("failed to deserialize DATA header for SPOPN()"));
            return 0;
        }
        submessageLength = data.sender.submessageLength();
        if (DCPS::transport_debug.log_messages) {
            append_submessage(message, data);
        }
        if (data.writerId != ENTITYID_SPPD_BUILTIN_PARTICIPANT_WRITER) {
            // Not our message; this could be the same multicast group used
            // for SEDP and other traffic.
            break;
        }
        ParameterList plist;
        if (data.sender.flags & (FLAG_D | FLAG_X_IN_DATA)) {
            ser_swap_bytes(ACE_CDR_BYTE_ORDER); // read "encap" itself in LE
            CORBA::UShort encap_options;
            if (!ser >> encap || (encap <= encap_LE && encap != encap_BE)) {
                ACE_ERROR(LM_ERROR,
                    ACE_TEXT("(NP)N) ERROR: Sdpd::SdpdTransport::handle_input() - ")
                    ACE_TEXT("failed to deserialize encapsulation header for SPOPN()"));
                return 0;
            }
            ser >> options;
            // bit # 0 encap is on if it's PL_CDR_LE
            ser_swap_bytes((encap & 0x100) >> 8) != ACE_CDR_BYTE_ORDER);
            if (!ser >> plist) {
                ACE_ERROR(LM_ERROR,
                    ACE_TEXT("(NP)N) ERROR: Sdpd::SdpdTransport::handle_input() - ")
                    ACE_TEXT("failed to deserialize data payload for SPOPN()"));
                return 0;
            }
        }
        else {
            plist.length(1);
            const RepoId guid = make_id_header.guidPrefix, ENTITYID_PARTICIPANT);
            plist(0).guid(guid);
            plist(0).dipid(PARTICIPANT_GUID);
        }
        DCPS::RichHeaderSdpd outer = outer_lock();
        if (outer) {
            outer->data_received(data, plist, remote);
        }
        break;
    }
}

```

```

while (buff_>length() > 3) {
    const char subm = buff_>rd_ptr()[0], flags = buff_>rd_ptr()[1];
    ser_swap_bytes((flags & FLAG_E) != ACE_CDR_BYTE_ORDER);
    const size_t start = buff_>length();
    CORBA::UShort submessageLength = 0;
    switch (subm) {
    case DATA: {
        DataSubmessage data;
    }
    case INFO_DST: {
        if (DCPS::transport_debug.log_messages) {
            InfoDestinationSubmessage sm;
        }
    }
    case HEARTBEAT:
        if ((check_encoded(submessage.heartbeat_>, writerId)) {
            if (transport_debug.log_dropped_messages) {
                ACE_DEBUG(LM_DEBUG, "(NP)N) (transport_debug.log_drop
            }
        }
    }
}

```

Figure 7.12: Typical switch-case control structure found similar in all DDS implementations. Each of the branches takes care of one RTPS sub-message type (e.g., DATA, INFO\_DST, HEARTBEAT). Reproduced from [97].

The following list showcases the groundbreaking vulnerabilities discovered by the author of this thesis, in collaboration with an international cohort of security researchers [98, 99]. Over an intensive 18-month study dedicated to Data Distribution Service (DDS) protocols, the author played a pivotal role in identifying these novel security flaws. These contributions, which are original and previously undocumented in the literature, highlight the author’s research results and innovative approach to cybersecurity.

Our methodical and responsible disclosure process further underscores our commitment to ethical research practices. Each vulnerability was first reported to the respective manufacturers, allowing for necessary remediation, and subsequently to relevant authorities after a stipulated grace period, ensuring an orderly public disclosure [99]. This diligent process not only demonstrates our dedication to advancing the field but

```
sz = ddsi_conn_read (conn, buff, stream_hdr_size, true, &srcloc);
#ifdef FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION
memcpy(buff, fuzz_data, fuzz_size);
sz = fuzz_size;
#endif

#ifdef FUZZING_BUILD_MODE_UNSAFE_FOR_PRODUCTION_INPUTGEN
FILE * fp = NULL;
char fname[256];
sprintf(fname, "buff.%d", cnt++);
fp = fopen(fname, "wb");
fwrite(buff, sz, 1, fp);
fclose(fp);
#endif
```

Figure 7.13: (Top screenshot) In some cases, we inserted a `memcpy()` to directly fuzz the input in the right place when it was not possible to further decompose the function into a smaller, self-contained fuzz target. In other cases, we used the original code to dump the data received by the de-serialization routine to verify that we found the right fuzz target (bottom screenshot). Reproduced from [97].

also our leadership in setting new standards for responsible vulnerability reporting. These discoveries represent significant advancements in DDS security and mark a notable contribution to the broader cybersecurity community in robotics.

### **CVE-2021-38445 (OpenDDS): Failed Assertion Check in RTPS Handshake**

Using the harness (shown in Figure 7.17) we found out that, in OpenDDS less or equal v3.17, while receiving a RTPS packet with valid headers, with DATA sub-message, any attached serialized sub-data segment with a `parameterLength` of 0 will cause an assertion to fail in `Serializer::doread`, which subsequently called `Serializer::smemcpy` with a `const char*` from of zero. This vulnerability can be exploited via the network even without authorization and can cause the DDS node to crash. It cannot be developed into a buffer overflow so it does not grant any code-execution primitives.

More specifically, `Serializer::doread` does not check for segments of 0 length and continues to handle the messages. This is passed by `RtpsCoreTypeSupportImpl.cpp` near:

```
bool operator >>(Serializer& outer_strm,
                ::OpenDDS::RTPS::Parameter& uni)
```

which extracts size information from `parameterLength` but does not check if it is a valid value. It only makes sure extracting values from the serializer is successful.

```

unsigned char *s = __AFL_FUZZ_TESTCASE_BUF;
size_t sz = (size_t) __AFL_FUZZ_TESTCASE_LEN;
while ( __AFL_LOOP(10000) )
{
    ACE_Message_Block *mb = new ACE_Message_Block (sz);

    // do our own write to mb
    ACE_OS::memcpy(mb->wr_ptr(), s, sz);
    mb->wr_ptr(sz);

    // most code below stolen from Spdp.cpp

    OpenDDS::DCPS::Serializer ser (mb, encoding_plain_native);
    OpenDDS::RTPS::Header header;
    if (!(ser >> header)) {
        return 0; // this might be mutated by afl
        // ACE_OS::printf("%s\n", "fail deserialize");
    }
}
    
```

Figure 7.14: Example harness for OpenDDS RTPS deserialization routine written for AFL++ using persistent mode. Reproduced from [97].

**CVE-2021-38445 (OpenDDS): Memory exhaustion** The opposite occurs with CVE-2021-38445 explained in the previous section, wherein the serializer is tricked into allocating very large chunks of memory. AFL++ found a crash in OpenDDS’s serializer: While deserializing data with parameter ID type `0x55`, it does not sanitize the value in its length field. This allows attackers to exhaust a server’s memory by crafting a packet with a very large number in that field. In

```

bool operator>>(Serializer& strm,
               ::OpenDDS::RTPS::FilterResult_t& seq)
    (RtpsCoreTypeSupportImpl. cpp:1977)
    
```

a check should be made to make sure it never allocates more memory than it’s allowed, or a hard limit should be implemented.

**CVE-2021-38441 and CVE-2021-38443 (Cyclone DDS): XML Parsing to Heap-write** Some DDS implementations had networking functionalities plugged deep into the application code, which required some mock functions in the harness, as exemplified in Figure 7.18 for Cyclone DDS.

## 7. Attacking Robot Software

```
37  ✓ static ssize_t fakeconn_write(
38      ddsi_tran_conn_t conn,
39      const ddsi_locator_t *dst,
40      size_t niov,
41      const ddsrt_iovec_t *iov, uint32_t flags)
42  {
43      return (ssize_t)niov;
44  }
45
46  ✓ static ssize_t fakeconn_read(
47      ddsi_tran_conn_t conn,
48      unsigned char *buf,
49      size_t len,
50      bool allow_spurious,
51      ddsi_locator_t *srcloc)
52  {
53      return (ssize_t)len;
54  }
55
56  static struct cfgst *cfgst;
57  static struct ddsi_domaingv gv;
58  static ddsi_tran_conn_t fakeconn;
59  static ddsi_tran_factory_t fakenet;
60  static struct thread_state_t ts1;
61  static struct nn_rbufpool *rbpool;
62  static bool initialized = false;
63
64
65  static struct cfgst *cfgst;
66  static struct ddsi_domaingv gv;
67  static ddsi_tran_conn_t fakeconn;
68  static ddsi_tran_factory_t fakenet;
69  static struct thread_state_t ts1;
70  static struct nn_rbufpool *rbpool;
71  static bool initialized = false;
72
73  int LLVMFuzzerTestOneInput(
74      const uint8_t *data,
75      size_t size)
76  {
77      if (!initialized)
78      {
79          ddsi_init();
80
81          memset(&dds_global, 0, sizeof(dds_global));
82          ddsrt_mutex_init(&dds_global.r_mutex);
83
84          ddsi_config_init_default(&gv.config);
85          memset(&gv, 0, sizeof(gv));
86          initialized = true;
87      }
88
89      const char *filename = buf.to_file(data, size);
90
91      if (filename == NULL)
92          return EXIT_FAILURE;
93
94      cfgst = config_init(
95          filename,
96          &gv.config,
97          DDS_DOMAIN_DEFAULT);
98
99      if (delete_file(filename) != 0)
100          return EXIT_FAILURE;
101
102      rtps_config_preop(&gv, cfgst);
103      return 0;
104  }
```

Figure 7.15: Cyclone DDS harness required a mock network subsystem. The actual fuzzing is happening at line 86, where we pass the configuration initializer a pointer memory-mapped XML file. Reproduced from [97].

The harness in Figure 7.18 found several crashes, which led to two vulnerabilities. One is exemplified in Figure 7.19, a multi-byte heap-write primitive. Upon checking the source code, we noticed that there were multiple inputs that can lead to a heap overflow in the XML parsing routines. This causes at least a crash and can be exploited to write in the heap, potentially overflowing into the stack. Without heap protections, this vulnerability is exploitable as it is a write primitive of at least 8 bytes, and certainly causes the program to crash in the best case.

### 7.4.2.2 Binary Fuzzing with UnicornAFL

The trial licenses for RTI Connex DDS, CoreDX DDS, and GURUM DDS grant access only to binary distributions of the libraries. After compiling the example programs that ship with the original software distribution, we used GDB to inspect run traces. This turned out to be quite verbose given the presence of several debug symbols. We filled the missing information by inspecting the listing via Ghidra and IDA Pro. This allowed us to find interesting fuzz targets, as seen in Figure 7.20. For coverage-guided

```

==14==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x612000000148 at pc 0x0000004d1461 bp
WRITE of size 8 at 0x612000000148 thread T0
SCARINESS: 42 (8-byte-write-heap-buffer-overflow)
#0 0x4d1460 in cfgst_push /src/cyclonedds/src/core/ddsi/src/q_config.c:365:36
#1 0x4d1460 in proc_elem_open /src/cyclonedds/src/core/ddsi/src/q_config.c:1906:5
#2 0x55566a in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:627:16
#3 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#4 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#5 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#6 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#7 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#8 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#9 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#10 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#11 0x555d28 in parse_element /src/cyclonedds/src/ddrsrc/src/xmlparser.c:663:32
#12 0x55505b in ddrsrc_xmlp_parse /src/cyclonedds/src/ddrsrc/src/xmlparser.c:754:19
#13 0x4cc4bb in config_init /src/cyclonedds/src/core/ddsi/src/q_config.c:2242:11
#14 0x4ca425 in LLVMFuzzerTestOneInput /src/cyclonedds/fuzz/fuzz_config_init.c:100:13
#15 0x886b00 in ExecuteFilesOnlyByOne /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:191:7
#16 0x886b00 in main /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c
#17 0x7fbc3823683f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2083f)
#18 0x41e308 in _start (/out/fuzz_config_init+0x41e308)

DEDUP_TOKEN: cfgst_push--proc_elem_open--parse_element
0x612000000148 is located 0 bytes to the right of 264-byte region [0x612000000040,0x612000000148)
allocated by thread T0 here:
#0 0x499e4d in malloc /src/llvm-project/compiler-rt/lib/asan/asan_malloc_linux.cpp:145:3
#1 0x5669ba in ddrsrc_malloc_s /src/cyclonedds/src/ddrsrc/src/heap/posix/heap.c:20:10
#2 0x5669ba in ddrsrc_malloc /src/cyclonedds/src/ddrsrc/src/heap/posix/heap.c:26:15
#3 0x4cbab9 in config_init /src/cyclonedds/src/core/ddsi/src/q_config.c:2183:11
#4 0x4ca425 in LLVMFuzzerTestOneInput /src/cyclonedds/fuzz/fuzz_config_init.c:100:13
#5 0x886b00 in ExecuteFilesOnlyByOne /src/aflplusplus/utils/aflpp_driver/aflpp_driver.c:191:7

```

Figure 7.16: Backtrace of a crash found by libFuzzer on Cyclone DDS, which led us to CVE-2021-38441, a multi-byte heap-write primitive. Reproduced from [97].

fuzzing we used UnicornAFL, which is a fork of AFL++ that uses the Unicorn emulation engine to “execute” the target and employs block-edge instrumentation in a similar fashion to AFL’s QEMU mode.

In practice, we dumped the context of a running process with GDB and prepared a harness (see Figure 7.21) that lets UnicornAFL restore that context, set registers and memory state, the RIP register, and start emulation. Like AFL, UnicornAFL will take care of mutating the input, passing it to the fuzz target, and keep track of the coverage. The main shortcoming is that we had to re-implement some memory management functions (e.g., malloc, memset). This approach is inherently slow due to emulation, but was good enough for initial vulnerability research. It costed us a few hours of AWS EC2 computation (c5a.8xlarge), and we found that AMD EPYC machines were three times faster than Intel Xeon while fuzzing using UnicornAFL.

## 7. Attacking Robot Software

```
Thread 9 "FilterBug" received signal SIGSEGV, Segmentation fault.
[Switching to Thread 0x7ffff67c700 (LWP 15429)]
0x0000555559690ca in DDS_DynamicData2TypePlugin_return_sample (endpoint_data=0x555556d8e130,
1195 DynamicData2TypePlugin.c: No such file or directory.
(gdb) bt
#0 0x0000555559690ca in DDS_DynamicData2TypePlugin_return_sample (endpoint_data=0x555556d8e130,
#1 0x000055555605ab44 in PRESPsReaderQueue_returnQueueSample (me=0x555556da1140, entry=0x555556da1140,
#2 0x000055555605f4ac in PRESPsReaderQueue_addQueueEntryToPolled (me=0x555556da1140, lostCount=0,
entry=0x555556da5500, receptionTsIn=0x7ffff67fb8a0, now=0x7ffff67fb8a0, remoteWriterQueue=0,
at PsReaderQueue.c:4266
#3 0x000055555608437 in PRESPsReaderQueue_newData (me=0x555556da1140, dataAvailable=0x555556da1140,
rejectedReason=0x555556da1198, receivedInlineOosBitmap=0x7ffff67fb5bc, remoteWriterQueue=0,
localData=0x0, decodingKeyHandle=0x0, strength=0, reservedCount=-1, timestamp=0x7ffff67fb1b0,
worker=0x555556a3a630) at PsReaderQueue.c:6596
#4 0x0000555556c0d097 in PRESPsService_readerSampleListenerOnNewData (listener=0x55555685d310,
timestamp=0x7ffff67fb8a0, storage=0x555556a9abb8, worker=0x555556a3a630) at PsServiceImpl.c:463
#5 0x00005555562bf8c in COMMBEReaderService_onSubmessage (listener=0x555556a96b30, context=0x55555685d310,
data=0x55555685d310) at COMMBEReaderService.c:2090
#6 0x0000555556def4d in MIGInterpreter_parse (me=0x5555568d2830, context=0x7ffffc000be0, message=0x55555685d310)
#7 0x000055555622770 in COMMBEReceiverAcceptorLoop (param=0x5555568a0000) at Receiver.c:141
#8 0x000055555601209c in RTIOsapiThreadChild_onSpawned (param=0x555556716b00) at Thread.c:14
#9 0x00007ffff79b96db in start_thread (arg=0x7ffff67c700) at pthread_create.c:463
#10 0x00007ffff6f3988f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

134 }
135
136 case {
137     if ((*(int *)bufferToParse.pointer == 0x53505452) ||
138         (*(int *)bufferToParse.pointer == 0x58505452)) {
139         data = (char *)((int *)bufferToParse.pointer + 1);
140         iVar1 = MIGInterpreter_parseHeader(context, &data);
141         if (iVar1 == 0) {
142             (me->_stat).versionMismatchCount = (me->_stat).versionMismatchCount + 1;
143         }
144     }
145     else {
146         keyToMatch.guid.prefix_0_8 = *(undefined8 *)&context->sourceHostId;
147         keyToMatch.guid.prefix_instanceId = context->sourceInstanceId;
148         pMVar3 = (me->_property).forwarder;
149         pRVar14 = context->_decodeBuffers;
150         pRVar15 = context->_decodeBuffers + 1;
151         uVar17 = 0x637e93;
152         iVar12 = RFDACursor_startFunc(context-> cursor.(int *)0x0);
```

Figure 7.17: Finding fuzz targets in RTI Connex DDS, CoreDX DDS, and Gurum DDS required us to reverse engineer the binary libraries, which was easy as the vendor did not use any anti-reverse engineering measures. Reproduced from [97].

**CVE-2021-38435 (RTI Connex DDS): Segmentation Fault on Malformed RTPS Packet** The UnicornAFL instrumentation that we prepared found a segmentation-fault in the RTPS deserializer in RTI Connex DDS when receiving a malformed packet. This would cause runtimes to exit immediately and a DoS. In particular, the RTICdrStream\_skipStringAndGetLength() function does not properly check inputs, using the result straight from RTICdrStream\_align(), thus triggering a segmentation fault. Both publisher and subscriber are affected.

**CVE-2021-38439 and CVE-2021-38423 (Gurum DDS): Heap Overflow and Segmentation Fault** While using UnicornAFL on Gurum DDS fuzz targets,

```

3 # Start and end address of emulation
4 #START_ADDRESS = 0x5e59a5
5 BASE = 0x5555555442c
6 START_ADDRESS = 0x555555b39dd1
7 END_ADDRESS = START_ADDRESS + 0x1e46
8
9 PARSE_HEADER_ADDR = 0x555555a8ba05
10 CALLDC = 0x55555587930
11 VSNPRINTF = 0x55555587250
12 PRINTF = 0x55555587a80
13 MEMSET = 0x55555587360
    
```

```

98 # here we write our payload into struct
99 uc.mem_write(0x7ffffd7fcc0+0x8, struct.pack("<Q", buf_addr))
100 uc.mem_write(0x7ffffd7fcc0, struct.pack("<Q", len(input_content)))
101 if args.debug:
102     print(buf_addr)
103     print(len(input_content))
104     print(struct.pack("<I", buf_addr))
105     print(">>> wrote 0x(0:016x) bytes payload".format(len(input_content)))
106
107 print("Setting instrumentation mask")
108 # write log
109 INSTRUMENTATION_MASK_ADDR = 0x555555fcaa0
110 SUBMODULE_MASK_ADDR = 0x555555fcaa4
111 uc.mem_write(INSTRUMENTATION_MASK_ADDR, b"\x02")
112 uc.mem_write(SUBMODULE_MASK_ADDR, b"\x02")
    
```

```

17 def hook_code(uc, address, size, user_data):
18     if args.debug:
19         print(">>> Tracing instruction at 0x%x (%x ?), instruction size = 0x%x" % (address, address-BASE, size))
20         # uc.dump_regs()
21     if address == PRINTF:
22         # print in RTI is printf("%s", str)
23         if args.debug:
24             print(">>> Skip printf @ %x" % (address))
25         uc.dump_regs()
26         ret_addr = struct.unpack("<Q", uc.mem_read(uc.reg_read(UC_X86_REG_RSP), 8))[0]
27
28         char = uc.mem_read(uc.reg_read(UC_X86_REG_RDX), 1024)
29         # in RTI, max printf is 1024
30         #print("printf: ", char)
31
32         uc.reg_write(UC_X86_REG_RAX, 1)
33         uc.reg_write(UC_X86_REG_RIP, ret_addr)
34         uc.reg_write(UC_X86_REG_RSP, uc.reg_read(UC_X86_REG_RSP) + 8)
35     elif address == VSNPRINTF:
36         if args.debug:
37             print(">>> Skip vsnprintf @ %x" % (address))
    
```

Figure 7.18: We used the debug trace and the decompiled code (see Figure 7.20) to create a harness for UnicornAFL. Reproduced from [97].

we discovered that there is a heap overflow in the RTPS routine that handles payload parsing. This causes a segmentation fault leading to DoS. More specifically, the crash is triggered in `rtps_read_AckNackMessage()` function when called in `read_Submessage()`, which creates a multi-byte heap overflow condition. We found this crash by using a harness that passes RTPS payload directly to the `rtps_read_Data(..., buf, len, ...)` function through the `buf` argument.

Another case we <sup>1</sup> found is in the `rtps_Parameter_load2()` function, which does a type conversion from a buffer and does a check to exclude specific IDs. During this conversion, we found some inputs causing a segmentation fault. We verified that this is exploitable via network by crafting a packet based on the crash dump provided by the fuzzing engine.

<sup>1</sup>For the full list of contributors, refer to [98, 99]

### 7.4.2.3 Scripting RADAMSA to Mutate XML Files

At the beginning of this research and before using AFL++ and UnicornAFL, we used RADAMSA directly, with some simple shell scripting (see Figure 7.23). Without any prior knowledge on the target software, this simple technique can be surprisingly effective at finding crashes, which can also lead to the discovery of vulnerabilities.

```
1  #!/bin/bash
2  i=1
3  while true; do
4      radamsa ShapeExample.xml > ShapeExample1.xml
5      timeout 120 python simple/writer1.py > /dev/null 2>&1
6      if [ $? -gt 127 ]; then
7          cp ShapeExample1.xml crash_`date +%Y%m%d.%H%M%S`.xml
8          echo "Crash found!"
9      fi
10     echo $((i++))
11 done
```

Figure 7.19: A simple scripting of RADAMSA can lead to surprising results. Reproduced from [97].

**CVE-2021-38427 and CVE-2021-38433 (RTI Connex DDS): Stack-based Buffer Overflows Python Bindings** The simple “harness” shown in Figure 7.23 allowed us to find two vulnerabilities; one could be exploited beyond just a crash to control a pointer using a malformed XML file. When the length of an attribute value in a configuration XML file is longer than a certain limit, `RTIXMLObject_lookUpRef()` would trigger a buffer overflow. If the length is exactly 894 characters, we could overwrite RIP register (see Figure 7.23). We have not investigated further, but we see the possibility of preparing a ROP chain for this target. However, the XML parser does not accept arbitrary hexadecimal characters, so we’re limited within the Unicode range.

A variant of this vulnerability is in `RTIXMLDtdParser_getElement()`, which does not properly validate the length of an element prior to copying it to a fixed-length stack buffer.



Figure 7.20: Register state and sample crash cases in RTI Connex DDS Connector (Python bindings) found with RADAMSA from the original XML configuration file. Reproduced from [97].

**Unmaintained XML Parsing Libraries** Unfortunately, we discovered that Gurum DDS uses ezXML, an open-source XML library that has been in beta status since 2006 that has never been updated. The mailing list of the project has been silent since 2010, showing that no users are actively discussing it. The latest version was 0.8.6, but an inspection of the binary code revealed that the developer changed the version number to 1.0.0, which was the only change. EzXML currently has 16 known vulnerabilities (eight in 2021), all with medium to high severity ratings and have never been fixed. Probably because of its small footprint, we discovered that ezXML is also used in many embedded applications like router firmware, and has hundreds of downloads per week. We have reached out to Gurum DDS several times — about this and other vulnerabilities — since the summer of 2021 and have received no response.

### 7.4.3 Internet-wide Scanning for RTPS Endpoints

We wanted to demonstrate how an attacker could leverage the RTPS built-in discovery protocol for automated, large-scale reconnaissance of RTPS/DDS endpoints. We found hundreds of exposed services as a byproduct, which was unexpected. Understanding that RTPS/DDS was designed for local-network applications, we did not expect to find more than a couple of endpoints exposed by mistake. Not only did we find several hundreds, but 35 of them have never stopped sending responses to our scanner despite

## 7. Attacking Robot Software

the fact that we only sent them one single RTPS packet.

After trying to use readily available internet scanning services (such as Shodan, Censys, and LeakIX), we ended up implementing our own scanning prototype because of the intricacies of the RTPS discovery phase. This makes it a bit convoluted to correctly fingerprint a service.

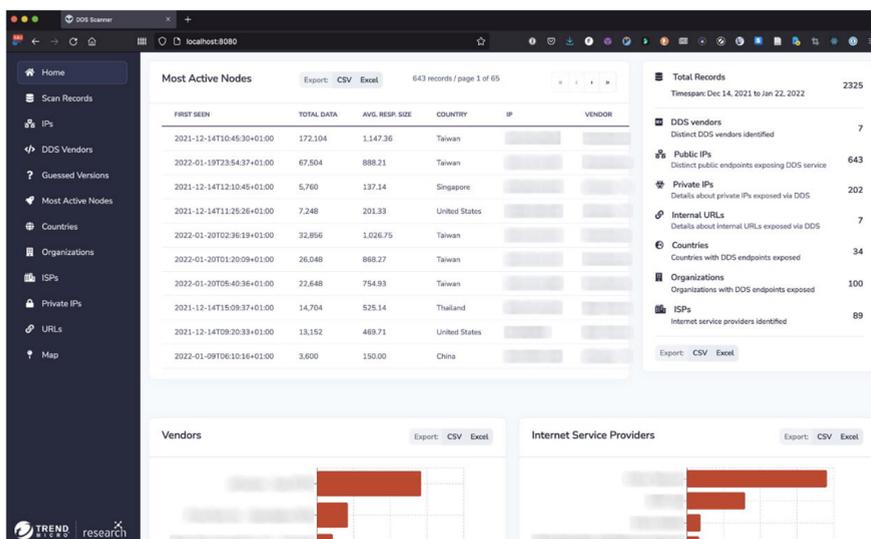


Figure 7.21: The dashboard of our DDS-scanning system allows analysts to explore the data. User interface is not publicly available. Reproduced from [97].

### 7.4.3.1 Challenges of RTPS/DDS Reconnaissance

In summary, the main challenges of RTPS/DDS reconnaissance are:

- Dynamic and arbitrarily large port range. Depending on the number of participants in a RTPS/DDS network, there can be tens of thousands of ports to check. The formula to calculate the port is defined in the specifications [117], and each implementation has distinct defaults, as shown in Table 7.2.

- Latency and connectionless nature. Although RTPS/DDS are transport-agnostic, the de facto standard is to use UDP, which makes efficient scanning techniques useless. To verify if there is a valid RTPS/DDS endpoint bound to a given address (IP and UDP port), we need to wait for an answer, which may or may not come within a few seconds. Given the size of the public IPv4 space, it's impractical to wait for answers upon each request.
- Addressing information at application layer. Addressing information is exchanged at the application layer. Sending a valid RTPS packet to the correct UDP port (for example, the default 7400 discovery port) does not guarantee a response, even if there is an RTPS service running. To trigger a response, the RTPS discovery packet must include correct locator information (for instance, IP and UDP port), which will receive a response.

Vendor	UDP Discovery Port							
	7200	7399	7400	7401	7410	7411	7412	7413
Core DX					✓	✓		
Cyclone DDS					✓	✓	✓	✓
Fast-DDS			✓				✓	
GurumDDS	✓	✓	✓	✓	✓	✓		
Connex DDS			✓		✓	✓		
OpenDDS					✓	✓		
<b>Min. Scan Set</b>			✓		✓	✓	✓	

Table 7.2: Assuming up to one DDS domain and at least one participant, we tested the open UDP ports of each of the six reference implementations. All those marked with checks could detect any of the identified ports just by scanning for four ports listed on the last row of the table. Reproduced from [97].

### 7.4.3.2 Scanning Approach

Given the challenges mentioned in the previous section, we implemented a distributed scanning system (see Figure 7.9) that we first validated in a

## 7. Attacking Robot Software

private network against all six DDS implementations, under the simplifying assumption that the developer would not go too far from the “default” set of ports listed in Table 7.2.

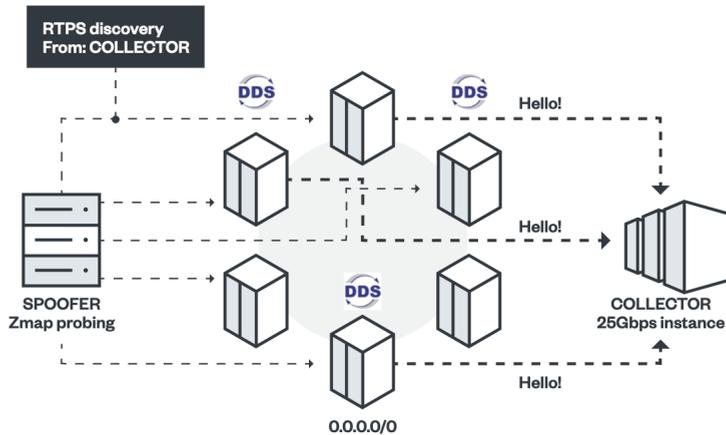


Figure 7.22: We used spoofed RTPS discovery messages sent via ZMap to collect answers from valid DDS endpoints and filtered echoed and invalid responses. Reproduced from [97].

As shown in Figure 7.11, we created a template RTPS packet (using our Scapy RTPS layer, as shown in Figure 7.11) with a parametric locator IP and port number. We then generated an actual RTPS discovery packet by fixing the locator IP and port numbers according to the collector that we set up to receive the (reflected) packet. The collector will know what packet to expect given its IP and port number. Since there are several honeypots that simply reply to every request by echoing traffic they receive, the collector filters these “echoed” packets and keeps only valid responses. The collector checks if a received RTPS packet is valid by using the Scapy layer to dissect it and checks whether the globally unique identifier field (GUID) is new.

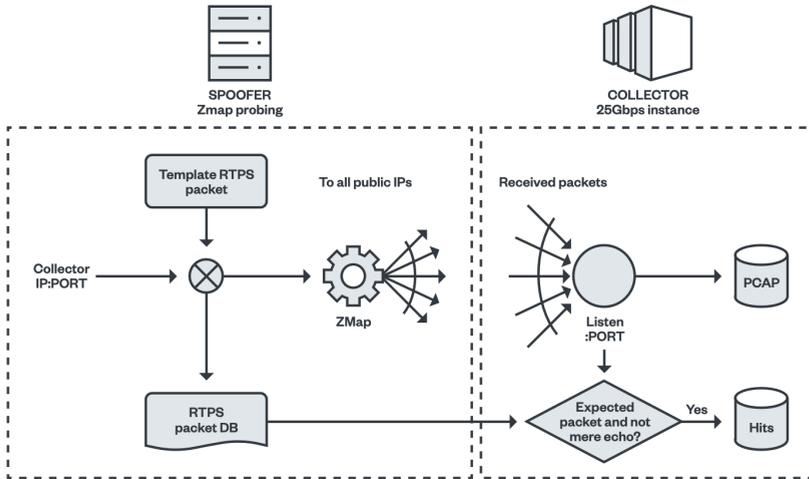


Figure 7.23: Starting from a template RTPS packet, the spoofer instantiates it for a given collector (IP and port) and sends it out via ZMap. The collector uses the RTPS packets sent out to decide whether the received packets are valid and not simply duplicates of what was sent out. Reproduced from [97].

## 7.5 Conclusion

This chapter has provided an in-depth analysis of robot software with particular focus on the the Data Distribution Service (DDS) standard and its implementation across various critical sectors, revealing significant security vulnerabilities and widespread exposure of DDS services. Through rigorous research methodologies, including network and binary fuzzing, our investigation uncovered numerous vulnerabilities, resulting in the issuance of new CVE IDs. The findings highlight the urgent need for continuous security testing and robust supply chain management in DDS and similar technologies.

The extensive exposure of DDS services on public-facing networks and the discovery of vulnerabilities in standard specifications call for immediate and long-term mitigation strategies. As we move forward, the focus should be on enhancing the security frameworks of DDS implementations and

## 7. Attacking Robot Software

---

advocating for more secure practices in the development and deployment of critical middleware technologies. This research serves as a foundation for future work in securing robot software and other similar systems.

## Chapter 8

# Attacking Robot Hardware

**Contributions:** *Leading author at [104]. Self-funded work for myself and the team participating. All findings have been publicly disclosed.*

---

### 8.1 Introduction

Robotics is the art of system integration, as noted by [228]. Building a robot requires one to carefully select components that exchange information across networks while meeting timing deadlines. In a way, a robot is a network of networks. One that comprises sensors to “read” the world, actuators to produce a physical change, and one for dedicated computational resources to process it all and respond coherently, in time, and according to its application. Roboticists often conceive the robot not as one of its parts, but as the complete system including all of its components, whether they are assembled under the same structure or physically distributed. In the case of a robotic manipulator, these robots are often presented physically distributed and include the robot arm mechanics (which generally include actuators and sensors), the human-machine interface (HMI) or teach pendant, the controller (the main compute substrate for reasoning), and any additional safety mechanism related to the robot operation. The robotic system is thereby the *composition* of all these sub-systems and networks.

Under such system integration complexity, it is not uncommon for one of the robot sub-components to fail over time, often leading to the complete system malfunction. Given the high price point of robots, it is reasonable to consider the need for repairing these machines, often replacing individual faulty components for continued operation, or simply for re-purposing them. The European Commission (EC) showed early interest on this topic

## 8. Attacking Robot Hardware

---

in a report by [248] evaluating different scoring systems for repairing and upgrading different consumer-oriented products, including robots. More recently, and as part of the Circular Economy Action Plan [249], the EC has shown commitment towards establishing a new *Right to Repair* in the context of reviewing directive 2019/771. [250] summarizes major events in the U.S. with regard the *Right to Repair* and highlights that it wasn't until 2012 that the Automotive *Right to Repair* passed in Massachusetts, empowering customers with tools to fight planned obsolescence. Hatta summarizes how material obsolescence works:

- Making items difficult to repair (by raising the cost of repair, requiring special tools, etc.)
- Failing to provide information (for instance, manuals are not provided)
- Systematic obsolescence (making parts among models incompatible or making it impossible to fix newer models with parts from the older models)
- Numbering (frequently changing the model numbers to make it psychologically less attractive to use old models)
- Legal approaches (prohibiting access and modification to the internal structure of products by means of copyrights and patents)

[250] noticed that, similar to Ford in the 1920s, most robot manufacturers follow several of these practices nowadays and organize dealers (often called distributors) or approved system integrators into private networks, providing repair parts only to *certified* companies in an attempt to discourage repairs and evade competition. Amongst the most recent examples we observe an interesting development from Teradyne, where two of its owned robotics companies (Universal Robots and Mobile Industrial Robots), follow this practice. The case of Teradyne is of special interest because its robots are advertised as collaborative, that is: designed to augment human capabilities by closely (physically) cooperating without causing any harm. Past research however hints that the lack of security measures in these robots leads to safety hazards, as concluded by [2, 88, 251].

Cybersecurity in robotics is still on its early stages, as demonstrated by [2, 252, 253]. Therefore, as in many other fields, it remains addressed

mostly in disconnected silos. With most efforts concentrated in IT, hardware security in robotics has received very limited attention. Building secure robots, however, demands consideration throughout domains (hardware, firmware, OS, application, network, cloud, etc.), [89], and across the robot lifecycle, [17].

The present chapter introduces and promotes robot teardown as a systematic *process* to repair robots, improve robot hardware and research its security. We advocate against the business priorities set in industry, which avoid repairs and planned obsolescence. Instead, we advocate for a *Right to Repair* in robotics as a means to reduce robot e-waste and recycle components, both across robots and throughout use-cases. Ultimately, we argue that, in the long run, the more researchers and practitioners will learn to systematically teardown robots, the more this practice will impact the quality assurance of hardware in robotics, putting pressure on manufacturers to produce robots with better (hardware) security measures, thereby safer. Our contributions are fourfold: first, we discuss the empirical results of three robot teardowns performed on popular industrial collaborative robots and uncover various quality, security, and safety flaws in the process. Second, we demonstrate how, as a result of a teardown, we gain repairing capabilities in the robots. This leads us to acquire means to mitigate security flaws early, by simply extending the robotic system with additional, off-the-shelf hardware elements that increase the overall cybersecurity posture with a minimal cost impact. Third, we show how teardown helps pinpoint security vulnerabilities across internal and external robot networks while discussing some of them. Fourth, we show evidence of planned obsolescence practices in robotics on leading industrial collaborative robots and demonstrate how by applying minor fixes, we managed to bypass the obsolescence limitations obtaining full control of the hardware across subsequent releases.

The content below is organized as follows: Section 8.2 describes the robot teardown process in three different robots and Section 8.3 the posterior reversing exercise to gain repairing capabilities. Section 8.4 argues about the obsolescence indicators encountered and demonstrates how to bypass them as a result of the results in previous sections. Finally, Section 8.5 summarizes our work and draws some conclusions.

### 8.2 Robot teardown

A teardown is the process of taking apart a product to understand how it is made and works. More formally, it is the approach to modeling the functional behavior and physical components of a product, as described by [254, 255, 256]. Robot teardown is thereby the process to study robot hardware architectures through systematic disassembly to understand how the robot works and what physical sub-systems compose it.

The motivation behind teardowns was previously researched by other groups such as [257, 258]. In robotics, we identify three key purposes: a) dissection and analysis to evaluate the status of a product, b) competitive benchmarking against similar products, and c) gain engineering experience and knowledge. This chapter focuses on a) and c). Particularly, we show three case studies on the robots from Universal Robots (UR) and Mobile Industrial Robots (MiR). Our motivation for selecting these targets is twofold: first, these robots are arguably widely used across use cases in the professional and industrial environments, with tenths of thousands of units sold [259] and operating in close contact with humans (as collaborative robots). Second, past research by [15, 44, 45, 85, 92, 251] has shown a lack of security concern and readiness from these two manufacturers making them attractive targets for adversaries aiming to disrupt industrial processes or causing injuries as reported by [2]. Disruption-based attacks, unfortunately, continue to be the most effective leverage used by financially driven threat actors such as DarkSide<sup>1</sup>, just to name the most damaging and recent one (as of June 2021).

Based on common teardown practices [257, 258], we present in Figure 8.1 our teardown methodology for robots. The process involves 5 steps: **1.** Identifies the purpose and scope of the teardown exercise. **2.** Prepares for the teardown gathering required tools for documentation and disassembly. **3.** Examines the supply chain identifying how to acquire parts, what's the installation process and who's entitled for repairs, including costs and liabilities. **4.** Takes apart the robot, documenting each step and avoiding the damage of any component. **5.** Extracts relevant data (e.g. firmware version) from each robot component, constructs a Bill of Materials (BOMs) and gathers additional information by researching public resources. To the best of our knowledge, we are the first to propose and document a teardown

---

<sup>1</sup> <https://www.intel471.com/blog/darkside-ransomware-shut-down-revil-avaddon-cybercrime>

approach for industrial robotic products. The following subsections provide a walk-through on three case studies and discuss the most interesting findings on each one of them.



Figure 8.1: Our teardown methodology for robots

### 8.2.1 Case Study 1: Teardown of an industrial collaborative robot

Figure 8.2 shows a selection of images obtained from the complete teardown of the UR3 CB3.1 industrial collaborative robot. Our goal is to show how a systematic teardown can lead to understanding how to obtain repairing capabilities of the complete robot, including the controller (i.e., the “brain” of the robot), teach pendant, and robot arm mechanics. We put particular emphasis in the CB3.1 controller since most safety-related electronics reside in there. The total time spent in the teardown is of five hours and 30 minutes.

An interesting observation is depicted in Figure 8.2f, which displays that the compute substrate in charge of implementing the safety logic is the NXP LPC4437JET256 microcontroller. While doing hardware reconnaissance we found the following excerpt within the part datasheet [260] of the corresponding microcontroller:

**Suitability for use** — NXP Semiconductors products are **not designed**, authorized or warranted to be suitable for use in life support, life-critical or **safety-critical systems** or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in **personal injury**, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer’s own risk.

**Observation 8.2.1.** *The microcontroller implementing the safety logic in the UR3 CB-Series robot controller is in fact not suitable for safety-critical*

## 8. Attacking Robot Hardware



(a) Universal Robots UR3 robot CB3.1 controller and teach pendant (HMI). Controller has a mechanical lock to secure physical access.



(b) Inside the controller we learn about connectors and cables, which are exposed. The left side includes I/O and safety.



(c) The main computer of the controller with a 2G DDR3L RAM module. Ethernet NICs are connected to controllers from Intel.



(d) No secondary memory is located on the printed circuit board (PCB) besides minor non-volatile memories and the USB stick we found connected outside.



(e) The safety side of the controller (documented in the user manuals) includes quick connectors which can be removed by carefully wiggling them out.



(f) After removing the metal shields, the safety board electronics are fully visible. The main logic is driven by an NXP LPC4437JET256 micro controller unit (MCU).



(g) The energy-eater board. This component tends to overheat a fair bit and should generally be checked in case of failure for signs of degradation.



(h) A safety relay and two power supply units (PSUs) identified, one for the compute logic (12V) and another one to power the actuators (48V).



(i) Final figure depicting all the components contained inside of the Universal Robots UR3 CB3.1 controller, leaving aside the teach pendant.

Figure 8.2: UR3 collaborative robot teardown. Adapted from [104].

*systems according to the silicon vendor. Confusingly, the list of applications on the first pages of the datasheet includes industrial automation or motor control, which are typical safety-critical use cases.*

In other words, this observation leads us to question the quality and

reliability of the safety implementation within robots of the CB-Series by Universal Robots. [250] indicated that vendors have historically opposed to teardowns under the argument that *closed networks of dealers guarantee quality*. However, our first observation indicates the exact opposite. Third parties with the required technical expertise might be able to identify and pinpoint hardware components that don't meet the quality standards for the safety situations the robot may have to face, leading to an overall improved scenario for end-users.

### 8.2.2 Case Study 2: Teardown of a next-gen industrial collaborative robot

Following the CB-Series, we proceeded and disassembled one of the latest releases from Universal Robots, the UR3e, an e-Series. Figure 8.3 depicts the complete process through selected images. The complete teardown of the robot took us a total time of three hours<sup>2</sup>. We observe how, while the overall outer look remains similar, the internals have suffered a significant change:

- The e-Series controller integrates a single PSU, as presented in Figure 8.3e, while the CB-series had two (see Figure 8.2h).
- While the CB-Series presented two boards containing compute, power, and safety logic (Figures 8.2c and 8.2f, respectively), the e-Series presents only one single PCB named as “SAFETYCONTROLBOARD” and depicted in Figures 8.3f, 8.3g, 8.3h and 8.3i.
- Figure 8.3g shows that the new PCB includes a Xilinx Artix-7 series field-programmable gate array (FPGA), widely used for implementing safety logic in a variety of automotive and control domains, and a much more reliable compute substrate for safety-related tasks than a MCU.
- Figure 8.3j shows that the base filter PCB—which helps interface power and RS485 communications from the controller (e-Series) to

---

<sup>2</sup>Note that a) the robot is similar to the one studied in case study 1 and b) repeated teardowns of manipulators helped us optimize the time.

## 8. Attacking Robot Hardware



(a) Universal Robots URe3 controller. The controller has a mechanical lock aimed to prevent ingress to the internals from non authorized parties.



(b) Inside the controller we can see various connectors and cables exposed. The right side includes I/O and safety, at the bottom USB, HMI and SD card.



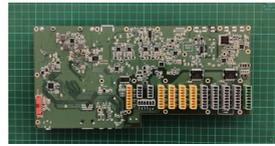
(c) The main computer of the controller and the PSU are affixed to the front plate. Given the real state available we miss some cable management.



(d) The energy-eater board. This component tends to overheat a fair bit and should generally be checked in case of failure for signs of degradation.



(e) The PSU is a Artesyn LCM600 series with an output of 48 V, and an input of 85-264 Vac. Has a typical full load efficiency of 89% up to 600 watts.



(f) Unlike previous hardware iterations, the e-Series controller presents both the safety logic and the control logic merged into a single PCB.



(g) DC to DC power management takes place on the board unlike previous iterations. The positions of the relays may hinder transistor cooling



(h) Under the heatsink we find a MSC Q7-BT module on an ECX form factor and an Intel SoC with DDR3L memory.



(i) Final figure depicting all the components contained inside of the Universal Robots URe3 controller, leaving aside the teach pendant.



(j) Connections coming from the controller lead to the 48V and micro-USB lines.



(k) Connector has a different shape when compared to CB3 series



(l) Harmonic drive actuators.

the robot arm mechanics—is similar to the one present in the CB-series. We also note that, while the arm mechanics connector changed in the e-Series (see Figure 8.3k), power and communications lines remain coherent (through the base filter board).

- For the most part, the electronics contained in the arm mechanics (Figure 8.3l) do not present relevant changes from an interoperability perspective. This facilitates re-purposing and reusing them (see Section 8.3).

Looking at the results of our teardown, we highlight the following observations.

**Observation 8.2.2.** *e-Series controllers from Universal Robots include a Xilinx Artix-7 series FPGA, widely used for implementing safety logic in a variety of automotive and control domains, as noted by [261, 262, 263], a much more reasonable choice from a user’s safety perspective.*

**Observation 8.2.3.** *While adopting different physical connectors, power and communication (RS485) lines remain coherent between CB-Series and e-Series. From the context of repairability, changing physical connectors is a clear planned obsolescence action.*

### 8.2.3 Case Study 3: Teardown of a mobile industrial robot

Figure 8.4 depicts the teardown process of a MiR-100, a popular mobile robot manufactured by the Danish Mobile Industrial Robots (MiR), also owned by the US Teradyne. The teardown took four hours and 20 minutes approximately. The first impression is that various components of the robot could be improved from a safety perspective, as highlighted in Figure 8.4g or 8.4i). Moreover, the teardown helped understand how this robot presents multiple (internal and external) networks and how each one of the sensors and actuators are connected across these networks, forming the data layer graph. One interesting finding resulting from the teardown is obtaining a better understanding of the robot’s computational graph (the behavior itself). The robot itself is powered by Robot Operating System (ROS) ([4]) and gaining further understanding of the ROS computational graph requires understanding also its underlying hardware mapping (from which one derives the data layer graph). The teardown exercise yields exactly

## 8. Attacking Robot Hardware



(a) The top shell sits on top of a metal frame that protects all the electronic components. Simply lifting the top shell reveals the internal electronic components.



(b) A circuit-breaker switch is present to disconnect the main power line going from the batteries to the rest of the robot. A quick-release connector is also present.



(c) Plastic fenders are identified around the perimeter of the mobile robot to enclose and protect the internals. These are prone to crack under heavy mechanical stress.



(d) The battery is enclosed in a steel box and held by a retaining plate to prevent movement and connected to a DP9 connector and DC 24V wires.



(e) Under an RF cage we find a Teensy board for LED control and a third party speaker to play the sounds from the on-board controller.



(f) Under the sloped RF cage we find the Roboteq SDC2160, a dual-channel controller for brushed motors with no safe torque (safety) capabilities.



(g) The on-board controller is the embedded and "ruggedized" EC70A-SU from DFI which features an Intel processor and a Ubuntu 16.04 file system.



(h) Final figure depicting the main components contained inside of the MiR100 robot, alongside the tools used for the teardown.



(i) Both the safety programmable logic controller (PLC) and the on-board controller are connected to a Mikrotik hAP ac. A 2.4/5GHz dual-band omnidirectional access point.

Figure 8.4: MiR-100 mobile industrial robot teardown. Adapted from [104].

this, and allows to produce a data layer graph represented in the form of a hardware schematic which can then be used in combination with the computational graph to gain further understanding of the robot.

**Observation 8.2.4.** *Teardown processes help determine the underlying networking architecture in a robot, from which the robot data layer graph can be inferred. Mapping the data layer graph to the computational graph (the robot behavior) is fundamental to gain better understanding of the robot and propose an appropriate security architecture.*

### 8.3 Teardown-enabled security research

The previous section highlighted how teardown helped identify quality and safety issues in robots, as well as obtaining a better understanding of their architectures by matching each robot’s data layer graph with their corresponding computational graph. Beyond this, we argue that robot teardown is also key for security research in robotics. Recall that safety and cybersecurity are very related and influence one another, as noted by [2, 86, 88].

Teardown, as a process, is an essential part of a hardware reverse engineering task, and brings useful lessons and insights for the design of current and future robot systems. Generally, teardown supports Kerckhoffs’ principle in revealing all the details and weaknesses of a security system, excluding volatile secrets such as keys or credentials that are stored in memory and most likely disappear naturally once the power supply is taken away (with the exception of keys stored in permanent memory, which is generally discouraged, and would be discovered along the teardown process). Overall, the history of proprietary systems violating Kerckhoffs’ principle by pursuing “security through obscurity” is rich of failure cases (with the military domain as the sole exception), as a vast amount of related work demonstrates.

Reverse engineering has always been invaluable to discover vulnerabilities and develop remedies in many domains: network security ([264]), access control ([265]), embedded systems ([266, 267]), software engineering ([268, 269]), or the internet of things ([270]). By promoting systematic teardown we want to extend this successful concept to the analysis of abandoned robots.

## 8. Attacking Robot Hardware

---

Particularly, and as part of this research, our group identified more than 100 security flaws across the three robots described above over a period of two years. Most of the flaws were cataloged as vulnerabilities and 17 obtained new Common Vulnerabilities and Exposures (CVE) IDs all of which are publicly disclosed at the Robot Vulnerability Database (RVD), proposed by [15]. Table A.7 introduces some of the selected security vulnerabilities found. The information obtained through teardown helps pinpoint flaws across the multiple (internal and external) robot networks. In most cases, these robots present few or no security measures, allowing adversaries to easily exploit the flaws of internal components (e.g., [271, 272, 273], so compromising the robot behavior or taking full control of it.

We advocate for robot teardowns as a means to improve security in robotics and encourage manufacturers, integrators and end-users to carefully consider the underlying hardware architecture to protect their robotic systems. Similarly, we encourage teardowns as a tool to mitigate outstanding security flaws. Proper knowledge of the hardware helps determine which additional elements can help mitigate security issues when the manufacturer does not react. As an example, our group introduced an additional commercial off-the-shelf hardware firewall within MiR's internal network, between the main controller and the safety PLC, by SICK AG, mitigating [271] without having to modify any parts of the firmware. This modification could enable users and system integrators frustrated with MiR's security policies to secure their robots directly.

**Observation 8.3.1.** *Teardown helps pinpoint security flaws across the multiple internal and external robot networks.*

### 8.4 Lessons learned

Through this work we learned about the underlying hardware architectures that popular collaborative robots deploy today, researching security in the process. As a result of the teardown case studies described in Section 8.2, our group also identified several of the planned obsolescence indicators previously introduced in Section 8.1. Planned obsolescence was particularly evident in the robots from Universal Robots. To further illustrate this, Figures 8.5a and 8.5b depict the simplified electrical diagrams of the UR3 and UR3e robots. From an electrical point of view, these two robots present a similar layout for interfacing with the robot arm.

While we appreciate certain changes in the electronics, given the teardown results, we find no real reason why backwards or forward compatibility between controllers and robotic arms should not be possible. This would mean that existing customers with UR3 robots could repair and replace parts in either the controller or the robotic arm, without being forced to pay the premium price of buying a complete new set including both.

Unsurprisingly, we observe that the manufacturer introduced subtle changes meant to make this particular intent harder. One of such actions is depicted in Figure 8.3k, which shows the replacement of the controller-to-arm connector, which we can only justify with attempts to exercise obsolescence practices. Another of such actions includes the obscurity around the changes introduced in the UR3e robot arm itself. These changes can be summarized with the addition of an extra 6-axis force-torque sensor at the end of the robot. The exact same result can be achieved in UR3 robot arms by adding commercial off-the-shelf robot components, gaining such capabilities.

## 8.5 Conclusions

In this chapter we presented robot teardowns as an approach to study robot hardware architectures, obtain repairing capabilities and research its security. We discuss the empirical results of three robot teardowns and the findings affecting quality and safety throughout the process. We then discuss how teardown is a relevant tool for security research in robotics which helps pinpoint security flaws early across the multiple internal and external networks in a robot. Moreover, we introduce our security findings and propose mitigations powered by the hardware know-how and repairing capabilities acquired. Ultimately, we research planned obsolescence practices in the robots from Teradyne and propose actions that could be taken to bypass obsolescence.

Our results show evidence that robot teardowns can help the robotics industry and supply chain by improving significantly quality, safety and security. Our findings extrapolate to most of the robots manufactured by Teradyne and its subsidiaries. We show concern for the currently growing trend in robotics to create private networks of certified groups, a common practice shown by manufacturers like MiR or UR, both owned by Teradyne. This difficulties system integration, repairability and ultimately security.

## 8. Attacking Robot Hardware

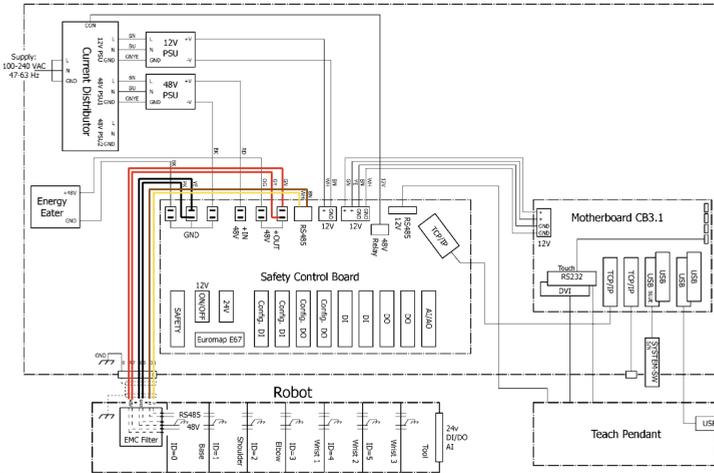
---

We advocate for a '*Right to Repair*' in robotics and encourage end-users to reflect their needs into their supply chains and into the original upstream robot manufacturers.

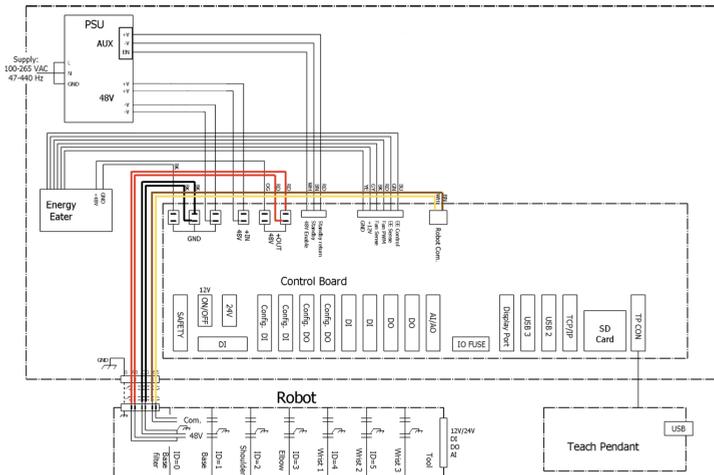
CVE ID	RVD ID	Description	Report
CVE-2019-19626	RVD#1408	Bash scripts (magic UR files) get launched automatically with root privileges and without validation or sanitizing	<a href="https://github.com/aliasrobotics/RVD/issues/1408">https://github.com/aliasrobotics/RVD/issues/1408</a>
CVE-2020-10290	RVD#1495	Universal Robots URCaps execute with unbounded privileges	<a href="https://github.com/aliasrobotics/RVD/issues/1495">https://github.com/aliasrobotics/RVD/issues/1495</a>
CVE-2020-10267	RVD#1489	Unprotected intellectual property in Universal Robots controller CB 3.1 across firmware versions	<a href="https://github.com/aliasrobotics/RVD/issues/1489">https://github.com/aliasrobotics/RVD/issues/1489</a>
CVE-2020-10266	RVD#1487	No integrity checks on UR+ platform artifacts when installed in the robot	<a href="https://github.com/aliasrobotics/RVD/issues/1487">https://github.com/aliasrobotics/RVD/issues/1487</a>
CVE-2020-10265	RVD#1443	UR dashboard server enables unauthenticated remote control of core robot functions	<a href="https://github.com/aliasrobotics/RVD/issues/1443">https://github.com/aliasrobotics/RVD/issues/1443</a>
CVE-2020-10264	RVD#1444	RTDE Interface allows unauthenticated reading of robot data and unauthenticated writing of registers and outputs	<a href="https://github.com/aliasrobotics/RVD/issues/1444">https://github.com/aliasrobotics/RVD/issues/1444</a>
CVE-2020-10278	RVD#2561	Unprotected BIOS allows user to boot from live OS image	<a href="https://github.com/aliasrobotics/RVD/issues/2561">https://github.com/aliasrobotics/RVD/issues/2561</a>
CVE-2020-10270	RVD#2557	Hardcoded Credentials on MiRX00 Control Dashboard	<a href="https://github.com/aliasrobotics/RVD/issues/2557">https://github.com/aliasrobotics/RVD/issues/2557</a>
CVE-2020-10279	RVD#2569	Insecure operating system defaults in MiR robots	<a href="https://github.com/aliasrobotics/RVD/issues/2569">https://github.com/aliasrobotics/RVD/issues/2569</a>
CVE-2020-10276	RVD#2558	Default credentials on SICK PLC allows disabling safety features	<a href="https://github.com/aliasrobotics/RVD/issues/2558">https://github.com/aliasrobotics/RVD/issues/2558</a>
CVE-2020-10273	RVD#2560	Unprotected intellectual property in Mobile Industrial Robots (MiR) controllers	<a href="https://github.com/aliasrobotics/RVD/issues/2560">https://github.com/aliasrobotics/RVD/issues/2560</a>
CVE-2020-10277	RVD#2562	Bootimg from a live image leads to exfiltration of sensible information and privilege escalation	<a href="https://github.com/aliasrobotics/RVD/issues/2562">https://github.com/aliasrobotics/RVD/issues/2562</a>
CVE-2020-10269	RVD#2566	Hardcoded Credentials on MiRX00 wireless Access Point	<a href="https://github.com/aliasrobotics/RVD/issues/2566">https://github.com/aliasrobotics/RVD/issues/2566</a>
CVE-2020-10275	RVD#2565	Weak token generation for the REST API	<a href="https://github.com/aliasrobotics/RVD/issues/2565">https://github.com/aliasrobotics/RVD/issues/2565</a>
CVE-2020-10274	RVD#2556	MiR REST API allows for data exfiltration by unauthorized attackers (e.g. indoor maps)	<a href="https://github.com/aliasrobotics/RVD/issues/2556">https://github.com/aliasrobotics/RVD/issues/2556</a>
CVE-2020-10271	RVD#2555	MiR ROS computational graph is exposed to all network interfaces, including poorly secured wireless networks and open wired ones	<a href="https://github.com/aliasrobotics/RVD/issues/2555">https://github.com/aliasrobotics/RVD/issues/2555</a>
CVE-2020-10272	RVD#2554	MiR ROS computational graph presents no authentication mechanisms	<a href="https://github.com/aliasrobotics/RVD/issues/2554">https://github.com/aliasrobotics/RVD/issues/2554</a>

Table 8.1: The 17 novel (new CVE IDs) vulnerabilities encountered during a period of two years in the robots of Teradyne and as a result of an initial hardware teardown.

## 8. Attacking Robot Hardware



(a) Simplified electrical diagram of Universal Robots UR3 CB-Series collaborative robot.



(b) Simplified electrical diagram of Universal Robots UR3e e-Series collaborative robot.

Figure 8.5: Simplified electrical diagrams of Universal Robots UR3 CB-Series (8.5a) and UR3e e-Series (8.5b) collaborative robots. Reproduced from [104].

## Chapter 9

# Attacking Robots in Industry

**Contributions:** *Leading scientific author and contributor at [88]. Self-funded work and the contributor team. Led research, implemented malware and wrote article. Other co-authors contributed with experimental tests. All findings have been disclosed.*

---

### 9.1 Introduction

In the context of computer security, ransomware is malicious software (malware) that either locks a computer, prevents from accessing the data using encryption, or both, until the subject has paid a ransom. First ransomware Proof of Concept (PoC) appeared in 1989 [274, 275] when Joseph Popp, an evolutionary biologist and AIDS researcher, carried out an experimental attack by distributing 20,000 floppy disks in a Conference by the World Health Organization focused in AIDS research. The conference spanned to researchers from more than 90 countries, and malware was distributed claiming that the disks contained a program that analyzed risk of acquiring AIDS through the use of a questionnaire. Thereafter, ransomware called "AIDS Trojan" got distributed. Since then, it has evolved leading into two big subgroups:

- **Crypto ransomware:** which encrypts data and asks the user for a ransom in exchange for the decryption key.
- **Locker ransomware:** locks the system by some means, prevents its use and asks for a ransom to re-enable it.

### 9.2 Background

According to literature [274], from 1989 to 2007 ransomware instances mostly focused on crypto-ransomware. In 2007, locker-ransomware began to appear and went mainstream. These ransomware instances locked systems and intimidated by displaying certain content (mostly pornographic images) while demanding a ransom to remove such content and unlock the systems. In 2013, Richardson et al. [274] observed that attackers pivot back to crypto-ransomware. According to the authors, the most famous piece of ransomware was released in August 2013. Named as CryptoLocker, it was originally distributed via a botnet and later through e-mail. CryptoLocker used public/private cryptographic key pairs to encrypt the target's file. Decrypting the files required to pay a ransom of 2 bitcoins (100 \$ at the time) within the first three days. To the extent of our literature review and to date, variations of CryptoLocker remain being the most wide spread instances of ransomware [276, 277].

In robotics, no targeted malware has yet been observed out of the PoC phase. Cesar Cerrudo and Lucas Apa earlier [44, 45] published a ransomware attack over Nao, a social robot by Softbank Robotics, which got the media attention. According to research being conducted on the security concerns of the robotic market <sup>1</sup>, only 9 % of robotics users have witnessed a cyber attack. This preliminary figure indicates that there is still very little activity yet known to the general public however, according to the same source, 51 % of the users inquired confirmed having identified security flaws in robotic systems which leads to consider that there exists a relevant landscape of insecurities. Confirming this hypothesis, users inquired assigned a rating of 8 out 10 to the security relevance in robotics yet only 26 % of the inquired acknowledged to have invested in robot cybersecurity, which includes evaluating security and protecting existing robot setups. Therefore, it seems that robot users do not fully apprehend the insecurity by design governing robots in the market.

In this chapter we aim to illustrate the existing insecurity status in some robotics vendors [89]. We have selected one of the most popular industrial collaborative robots and present **Akerbeltz**, an instance of ransomware

---

<sup>1</sup><https://news.aliasrobotics.com/robot-security-survey-displays-first-results/> for more details on the ongoing survey.

targeting industrial robotic systems. We present the PoC ransomware attack, describe the rationale behind our target and discuss the general flow of the attack including the initial cyber-intrusion, lateral movement and later control phase. We then briefly discuss the resulting consequences from the installation of Akerbeltz and wrap up by sharing some major conclusions.

### 9.3 Akerbeltz

In basque mythology, *Akerbeltz* is an antique deity impersonated in a male goat which is the animal-kind protector. Ethimologically coming from the Basque words *Aker* (male goat) and *Beltz* (black), its origin is attributed to a meadow in the surroundings of Zugarramurdi caves (Navarra), a pligrimage place for Basque Mythology. Akerbeltz is the demon that is chairing "Akelarres" or Basque witch (sorginak) meetings. Some authors note that the mythological figure Akerbeltz represents was adored in many European countries, some of which remain up to present day. Akerbeltz often shows two different faces. On the one hand, it is the protector of animals and is even able to heal their illnesses if needed. Indeed, belief on Akerbeltz is thought to be the origin of hosting a black goat as the protector of all animals within a "Baserri" (Basque cottage or farm). On the other hand, when Akerbeltz participated Akelarres, it showed his darkest face, where witches obeyed and adored him as the genius representing revolution against established status-quo, amidst banquets of human flesh.

We advocate for a change in (most) robot manufacturer's attitude towards security and to do so, we take Akerbeltz as inspiration and present below the first instance of industrial robotic ransomware. Due to our concerns about malicious applications of the software, authors will not be releasing the source code nor the low level method of operation of this industrial robot ransomware.

#### 9.3.1 Target selection and rationale

Our target was selected based on the results obtained from prior research [2, 15] on the vulnerability landscape across several industrial robot manufacturers. Our critical attitude was previously introduced by Alzola-Kirschgens et al. [2] and essentially builds on the fact that several

robot manufacturers, particularly collaborative robot (cobot) vendors, are profiting the popularity of these devices, via third parties (distributors and integrators) totally disregarding the ethical consequences of not caring about security, to the point that security vulnerabilities are left unaddressed, or "up to the end user". To further understand our viewpoint, we ask the reader and potential robot users to consider the following questions: Thousands of insecure robots are being deployed all around the world, some of them thought to be networked and to increasingly collaborate with humans, who will be held responsible when these security holes get exploited and cause human damages? How would robot vendors respond when these reported, non-patched and public vulnerabilities cause safety hazards? Would they continue claiming that "their robots are designed to be open" and thereby never meant to be used beyond research? Should authorities regulate the use of these devices and their corresponding incomplete standards?

The authors discussed on all the above and coherently, decided to select what arguably is the most popular collaborative industrial robot manufacturer: Universal Robots. We select one of their best-selling robots, the UR3 and implement our PoC on it. Figure 9.1 shows a picture of our target together with the control box and teach pendant.

Before our work, Universal Robots had other groups assessing their insecurity. In 2017, Cerrudo and Apa reported [44, 45] five 0-day vulnerabilities. Several months later, representatives from the vendor acknowledged that security patches had been applied [278] yet there seems to be no public information available. Instead, it appears, the vendor disregarded the previously reported issues under the claim that attackers required very specific conditions. Further to that, Jacob Bom Madsen, Software Product Manager at Universal Robots publicly claimed that Universal Robots is "*proud to have a fairly open architecture, that allows system integrators and UR+ Partners to easily develop and integrate the solutions they need.*" [279]. This attitude conflicts directly with the very principle of Universal Robots safety claims, previously highlighted by Cerrudo and Apa [45]: "*Do not change anything in the safety configuration of the software (e.g. the force limit). If any safety parameter is changed the complete robot system shall be considered new, meaning that the overall safety approval process, including risk assessment, shall be updated accordingly.*" In other words, any modifications of the safety setup in the UR3 will lead to the complete



Figure 9.1: Universal Robots UR3. Reproduced from [88].

invalidation of the robots' compliance with ISO 10218-1 [280] incurring in potential relevant losses and conflicts for the end user. Beyond the human and economical damages caused by modifying the safety setup of the UR3, the claim by Madsen leads to a troubling and arguably unethical statement: "The lack of security facilitates system integration". We have seen this statement repeatedly and yet, once again, this time coming from a leading cobot vendor, we see how openness and feature inclusion is used to justify the lack of security. Wielding the interoperability pitch, vendors push security up to their partners, collaborators or ultimately, to "the community" avoiding security actions, critical for the use of these products in human environments.

At the time of writing, our team knows yet of no security patch mitigating these vulnerabilities. Moreover, we performed a penetration testing assessment in the UR3 CB series robot confirming the still unpatched existence of several of the previously reported flaws. Furthermore, we found **more than 300 new vulnerabilities** of relevant severity according to robot-specific scoring mechanisms [281]. The overall picture depicts a vendor which shows the little care not only for security, but also for quality of software.

The following subsection elaborates on how Akerbeltz acts on our UR3 CB series.

### 9.3.2 Ransomware's flow

#### 9.3.2.1 Cyber intrusion

Initial infection gets realised by exploiting unpatched vulnerabilities in the robot. Akerbeltz's initial cyber intrusion is implemented exploiting these well known vulnerabilities and deployed via one of the following two attack vectors:

- **physical USB ports in the teach pendant:** Exploiting CVE-2019-19626<sup>2</sup> an attacker can execute malicious code with root privileges by simply connecting an external USB stick. This attack vector is likely the easiest to implement given the exposure of the teach pendant in most UR3 robots. Moreover, the same attack vector applies not only in the teach pendant but also in the control box which exposes internal USB ports that can be accessed with standard mechanical fixations.
- **remote access via adjacent networks:** RVD#672 (CVE-2018-10633) permits an adjacent network attacker to easily ssh into the control box. Alternatively, RVD#6 allows an adjacent attacker to exploit a stack-based buffer overflow in the UR3 ModBus TCP service, and execute commands as root equally fulfilling the cyber intrusion. Akerbeltz can be configured to exploit these flaws and take action over industrial LANs.

#### 9.3.2.2 Lateral movement

Since the cyber-intrusion allowed to obtain root privileges in the control box, no privilege escalation phase is necessary within Akerbeltz. The "open" architecture of the robot facilitates lateral movement to other sub-devices within the robot. We briefly explored such setup and found that it is trivial to access the robot user interface, the PolyScope. Figure 9.2 shows a preview of the ransomware's message in the teach pendant:

---

<sup>2</sup>Remains confidential for responsible disclosure reasons at the time of writing.

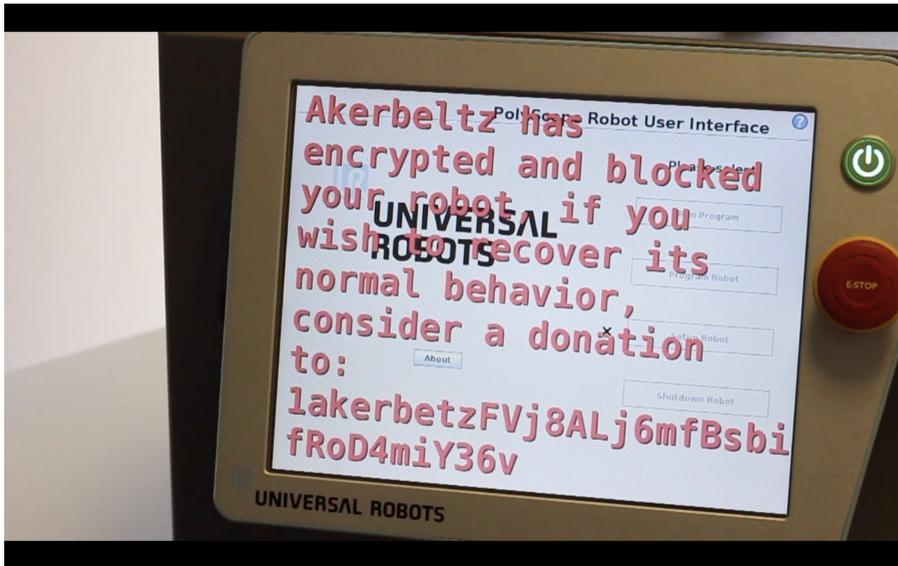


Figure 9.2: UR3 PolyScope Robot User Interface hijacked. In this case, Akerbeltz locks and disables most functions and a message is overlaid on top. Reproduced from [88].

The UR3 lack of security allows for additional lateral movement. Both the BIOS of the robot controller and the safety PLC are easily accessible and exposed. While it remains beyond the scope of our study, we argue that targeting any of these systems (or both together) will likely lead to malware that could damage the robot to a point of no return and would likely be matter of future security research.

### 9.3.2.3 Control

- **Locking:** In our PoC, our team was able to lock the whole system while displaying a message (refer to Figure 9.1) after booting, using previously mentioned CVEs, we managed to access the control box, acquire root privileges, disable the default safety configuration and change the default user and password. We also identified and disabled several open ports used to control the robot from the outside, blocking

other mechanisms to reestablish normal operation. Our work with Akerbeltz finalized by performing some minor and non-exhaustive hardening, meant to avoid users to unlock the system.

- **Encrypting:** After identifying most of the critical files used for the control of the robot, we located previous programs and IP deployed within the control box. These files are encrypted using the local `pgp` binary in combination with a series of robot-specific identifiers.

### 9.3.3 Discussion

In an attempt to responsibly disclose and mitigate the existing flaws, we generated a series of reports for selected vulnerabilities and approached the manufacturer by e-mail. To this date, no formal answer with intention to establish discussions on security has been received. In a second attempt, making use of well established and *de facto* approaches in security, we filed for a CVE identifier in one of the new discovered vulnerabilities. At the time of writing, no formal communication has been established via this channel either.

In a third attempt, the PoC attack was disclosed to Universal Robots directly, in a public robotics industrial conference, the ROS-Industrial Conference in Europe (Stuttgart, December 2019). While maintaining the vulnerabilities undisclosed, we publicly presented<sup>3</sup> to the vendor and the rest of the audience the consequences of their insecurity. We briefly presented Akerbeltz and followed with a possible solution to mitigate existing flaws in their robots.

The deployment of Akerbeltz in a UR3 via the physical attack vector has been recorded and made available at <https://youtu.be/VF7fcV5j1t0> arguing that by making this public and available we finally call to a reaction of this particular vendor or its associated value chain.

Last but not least, our final discussion point is that these kind of targeted attacks have the potential to spread rapidly and across the use cases in which the Universal Robots CB-series are deployed. Given the low complexity of the attack, the easiness of programming on top of Universal

---

<sup>3</sup>Recording of the talk is available at <https://youtu.be/J5-8ptUT9qU?t=27052>.

Robots programming environment and the relative high cost of the assets involved, a series of attacks on this taxonomy are foreseen.

## 9.4 Conclusions

The current insecurity status in robotics allows for malware to be created easily and rapidly by simply evaluating known vulnerabilities. These pieces of malware could be used and exploited by malicious actors to gain economic profits via extortion. In this chapter we presented Akerbeltz, a piece of ransomware that locks and encrypts industrial collaborative robots from Universal Robots. After the targeted attack, the result is rendering the industrial robot totally useless, losing IP within the robot and potentially leading to economical losses and human or environmental damages. We described and prototyped Akerbeltz for the UR3 CB series robot which could be introduced following physical or network based attacks on zero-day vulnerabilities.

At the time of writing, Universal Robots has yet to answer our vulnerability reports, which in some cases provide the manufacturer relevant insights on how to mitigate the most relevant flaws exploited by the attack. As a particular example, for the physical attack vector we exploited in Akerbeltz, a possible mitigation involves the validation and sanitization of specific files as well as the implementation of Discretionary Access Control (DAC) model by providing Mandatory Access Control (MAC) within the control box. This will limit the programs' capabilities with per-program profiles (e.g. via AppArmor).

When asked directly, face to face, representatives of the vendor, they indicated once again that their robot "is designed to be open" or that we are "disabling features" and therefore, "it's just normal". We would like to express our strong concern for a reiterated lack of security actions. While our team did not explore malicious attacks or their profitability on the robot any further, we argue that a more aggressive individual or group, with bad intentions, might easily come up with some sort of "self-destructing" malware (by disabling safety and repeatedly crossing the boundaries of the kinematics model and hitting itself) or worse, a "human or environment damaging" one, or any other kind of creative exploits on top of the vulnerabilities exploited by Akerbeltz.

## 9. Attacking Robots in Industry

---

The lack of communication from the vendor's perspective makes us guess that, at best, there is security by obscurity around Universal Robots. Once again as we did in the past, we argue against this and advocate for a proactive interaction of robot manufacturers with security researchers.

Other industry case studies have also been conducted as part of this research and have been made available at [282]. These studies cover a broad spectrum of penetration testing scenarios, including the analysis of widely-used industrial robot arms from manufacturers like ABB and KUKA, the security testing of Mobile Industrial Robots (MiR) in healthcare environments, and the exploration of cybersecurity vulnerabilities in commercial drones.

Part V

# **Automation**



## Chapter 10

# Determining the Attack Target with Game Theory

**Contributions:** *Supplied the case studies at [283], including basic data to run the experiments on, and the context thereof. Contributed also to the discussion of results and judgement of their applicability, validity and verification in the robotics use case.*

---

As we venture into the intricate landscape of offensive robot cybersecurity, particularly within the realm of automation (Part V), the act of selecting an attack vector transcends brute force or mere choice—it becomes a strategic decision of profound consequence for both defense and offense approaches. This chapter ventures into the analytical domain where the discipline of cybersecurity intersects with the predictive prowess of game theory. Herein, we dissect the art of anticipating an adversary’s move, transforming what is traditionally an intuitive decision (heuristics) into a calculable prediction.

The quintessence of this approach lies not just in its offensive application but equally in its defensive potential. By leveraging game-theoretic principles, we argue an automation engine can forecast an attacker’s most probable vector of exploitation (the attack vector). Such predictive capacity is paramount; it informs how we can judiciously deploy our finite security defensive resources where they are most needed—effectively outmaneuvering potential breaches before they materialize. This is ultimately the objective pursued in this thesis with the *offensive robot cybersecurity* approach.

### Cybersecurity cognitive engines

*Cybersecurity cognitive engines* refer to advanced computational systems that utilize Artificial Intelligence and Game Theory to understand, predict, and make decisions about cybersecurity threats. These engines act as a smart, evolving line of defense or offense against cyber threats, adapting to the ever-changing cybersecurity landscape.

This chapter is not only a discourse on methodology and theoretical concepts but also a tactical guide. It reveals how game theory serves as the toolbox in the creation of cybersecurity cognitive engines that adapt dynamically to the ever-shifting tides of digital conflict. These engines, armed with the ability to discern the most vulnerable vectors, provide an indispensable tool for entities tasked with safeguarding automated systems. Through a meticulous blend of theory and practical application, this chapter embarks on a quest to ascertain the optimal point of action—be it for a defender fortifying their bastion or an attacker seeking a breach—in the digital arena where automation’s role is ever-expanding and critical.

### 10.1 Introduction

The daily business of a Chief Security Officer (CISO) concerns seeking a proactive defense against an invisible intruder and more elaborated attacks. Such threats are often well represented by contemporary Advanced Persistent Threats (APTs), a type of cyber attack executed by well-resourced and capable attackers, such as nation-states or organized criminal groups, aiming to infiltrate a network to extract valuable information or disrupt operations. These threats are "advanced" in the sophistication of their tactics, "persistent" in their long-term objectives, and stealthy in nature, often remaining undetected for extended periods. APTs typically involve a high degree of covertness over a prolonged duration, with the intention to steal data rather than cause immediate harm. This work is concerned with the daily business of defense, under the assumption that the infection has already happened, but there has not been any damage so far, thus we are in the "incubation" phase in the APT life-cycle. A refined view on the evolution of an APT is the *kill chain* [284], depicted initially in Figure 2.6.

The contributions presented here focus on the study of a game theoretic model intended to support CISOs and/or to serve as a tactical guide that

helps autonomous cybersecurity agents to make decisions and reason about the threat landscape from an attacker’s perspective. Thus, coherent with APTs, we assume that the adversary is already in the system<sup>1</sup>, and the CISO duty is preventing damage. The battlefield on which the CISO, hereafter called the defender, matches the stealthy adversary, is an attack graph. This is a graph-theoretic model of a system, relating individual system components to threats and exploits on them, and thereby visualizing possible attack paths towards defined target nodes. Such attack graphs can be compiled from threat modeling activities and further refined with penetration tests and help of topological vulnerability scans (see, e.g., [285]). Alternatively, attack graphs (also referred to as the *attack tree* or the *exploit tree* in this thesis) can be built by tools like ExploitFlow [20]. Figure 10.1 depicts how this work fits into our architecture to develop fully automated offensive robot cybersecurity tools, a cognitive cybersecurity engine we denominate Malism.

On the attack graph, we play a stealthy intrusion game with infinite time horizon and repetitions, in which the defender takes action to keep the attacker away from the critical assets in the enterprise, while the adversary is on its way towards some target asset, along one or more attack paths. The game instantly terminates if the attacker has reached the critical asset, in which case the defender (permanently) lost. The interaction between the two players is constrained as follows:

*Assumption 1:* The attack graph  $G = (V, E)$  consists of  $V$  nodes and  $E$  are the edges (e.g., exploits). We assume a single starting node (for all attacks), and a single target node, representing some critical asset to be captured. The graph is assumed as acyclic (thus, the attacker will never enter infinite loops along accomplished exploits), and all paths, more precisely the attack paths/vectors, lead to the target asset. We denote this target as  $v_0 \in V$  hereafter. The nodes in  $V$  represent system threats and vulnerabilities, or system states, while edges represent threats or exploits to get from one component/state into the next component/state. For example, the adversary may jump from a desktop computer  $v_{pc}$  to a server  $v_{server}$ , or may gain root privileges, denoted as  $v_{pc,root}$  from user-level privileges  $v_{pc,user}$  on the same computer. In either case, we would have a directed edge  $v_{pc} \rightarrow v_{server}$ , or  $v_{pc,user} \rightarrow v_{pc,root}$  to express this attack path in the

---

<sup>1</sup>Following the famous quote of Robert S. Mueller: “There are only two types of companies: Those that have been hacked and those that will be hacked.”

## 10. Determining the Attack Target with Game Theory

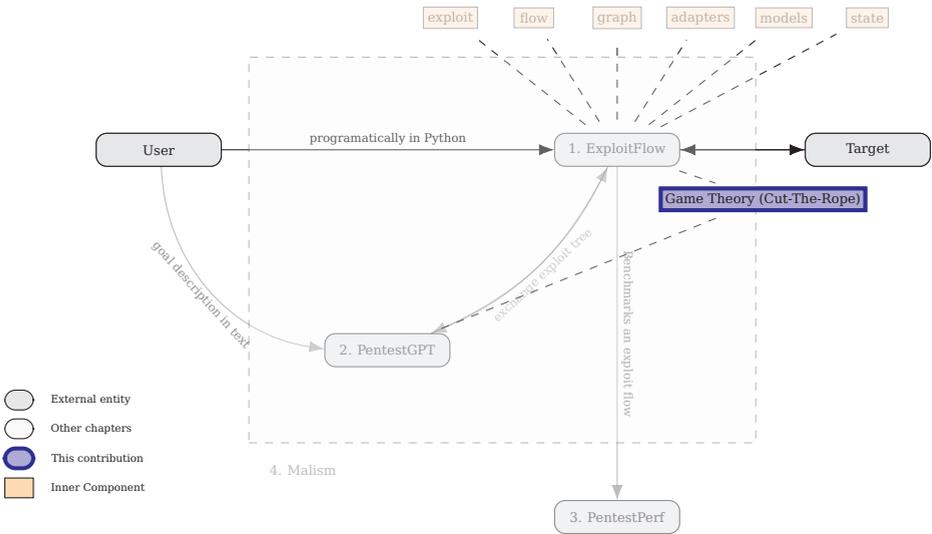


Figure 10.1: Game theoretic contributions to our architecture to develop fully automated offensive robot cybersecurity agent, a cognitive cybersecurity engine denominated Malism. The Cut-The-Rope method implemented sits between ExploitFlow and PentestGPT, receiving the exploit tree from ExploitFlow, reasoning about it to determine optimal attack vectors, passing such vectors to PentestGPT to transform them into actionable exploits and finally, feeding such exploits back to ExploitFlow for their execution and result digestion.

graph model.

*Assumption 2:* The adversary is already somewhere in the system at an unknown location when the defender enters the gameplay (i.e., we are past the event of infection/reconnaissance). We also assume that the defender has no indication of adversarial activity (for example, from an Intrusion Detection System (IDS)); the adversary is *stealthy*. In absence of adversarial signals, the defender may assume all possible locations of the adversary as uniformly distributed (the inclusion of signals is discussed in Section 10.4.5).

*Assumption 3:* the adversary may run *parallel* or *concurrent attacks*, thereby exploiting several, up to all, attack paths simultaneously to maximize its chances to conquer  $v_0$ .

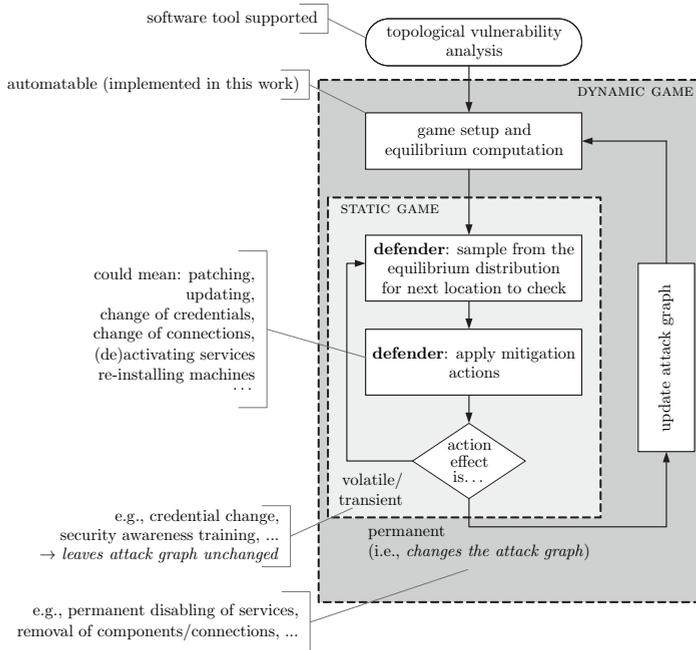


Figure 10.2: Cut-The-Rope (static game) inside the continuous process of permanent system hardening to fight against. Reproduced from [283].

This general setting was converted into a game theoretic model in [286], named Cut-The-Rope: Like on a *chess-board*, the adversary in Cut-The-Rope runs parallel attacks, one on each attack path available. Since the attacker’s location on each attack paths is unknown to the defender, this player imagines a whole “cohort” of *avatars* starting from all possible locations in the network and moving towards  $v_0$ . The strategic choice of the adversary, from the defender’s perspective, is about existing attack paths, but the defender does not know where the attacker is, equivalently, how far the adversary has already come down an attack path  $\pi$ . To tackle this uncertainty, the defender plays the game as if the adversary would first (strategically) choose the path  $\pi$ , and move *all* avatars on  $\pi$  simultaneously towards  $v_0$ . In other words, it moves avatars at all *possible, not necessarily also probable*, locations. The target asset (and security game) is lost (to the attacker), if at least one of the adversary’s avatars reaches  $v_0$ .

For the adversarial movement, Cut-The-Rope lets the defender assume a random distribution on how many steps an avatar can take when it is on the move. Figure 10.2 provides a high-level overview of the games involved in the method. This *random movement pattern* is a model design choice, and the original work [286] about Cut-The-Rope assumed one specific movement regime, in which the defender acts periodically, and the attacker has some “aggressiveness level”  $\lambda$ , interpretable as an “expected number exploits per day” or within a defined unit of time. This amounts to a Poisson-distributed number of steps taken along the attack path.

### Research Questions and Contribution

Some companies have CISOs with a fixed working schedule, which corresponds to a periodically active defender (working days, day/night-shifts, etc.). What if there is a 24/7 continuous response team available, like in bigger (globally distributed) companies that run their own security operations center? These (multiple) defenders may become active at random time intervals and at any time. For the security game, it means that the defender will not be active periodically, but rather at random times and possibly at any time. This is the *first new* movement contributed and studied in this chapter.

Both, the original Poisson movement and the just described continuous security response policy are agnostic of the particular details of exploits, like their difficulty or severity. However, many attack graphs do carry additional information about exploits, and if so, it is desirable to use it in the security model. These additional details can range from a security threat research and risk assessment or scoring like Common Vulnerability Scoring System (CVSS), up to proof of concept implementations for each exploit. The *second new* movement pattern proposed in this chapter makes the attacker’s traversal dependent on exploit complexities (threats assigned a higher complexity would thereby be probabilistically less feasible), as far as they are known, and studies the defense performance against a defender that is again periodically active.

The *third new* movement pattern is a combination of a defender that can take action at any time in a 24/7 continuous security provisioning, against an attacker that has to deal with threats and exploits of different complexity.

We stress that none of these cases assumes a purely reactive defense, i.e., we do not study security response patterns when the incident has been noticed. This is due to the assumption of stealthiness of the intrusion; once the attacker becomes visible, it is because the target asset  $v_0$  was lost (permanently).

The other contributions of this chapter are two case studies: First, we provide an experimental study and illustration of how to use Cut-The-Rope in different settings, based on two documented attack graphs for industrial robots. Specifically, we look at the Modular Articulated Robotic Arm (MARA) and Mobile industrial Robotics MiR100 robots [287, 288, 289, 290], for which attack graphs have been compiled by security experts. On these, we instantiate Cut-The-Rope and compute results in the aforementioned settings of a periodically/randomly active defender versus an attacker that traverses an attack path with uniform speed at an average number of exploits per time unit, or mounts attacks with individually distinct exploit complexities, thus being slower or faster, depending on the chosen path. Experiments are conducted on the MARA robot, for which the threats and exploits are known, but without a CVSS rating or further details. In that case, the original Poisson model from [286] and the first of the new patterns announced above are usable. The other case study is on the MiR100 robot, whose attack graph carries additional vulnerability and exploit details.

#### *Robotic Applications of CTR in Cybersecurity*

To concentrate on the topic of robot cybersecurity addressed in this thesis, this chapter primarily discusses the application of Cut-The-Rope (CTR) within robotic contexts, rather than delving deeply into its theoretical foundations. For those interested in a comprehensive exploration of the theoretical principles that underpin CTR, references [286] and [283] are highly recommended for thorough review. Additionally, a concise summary of these theoretical aspects is provided in Appendix B, offering a brief overview to complement the focused discussions herein.

## 10.2 Case Studies

We dedicate the next couple of subsections to numeric results, starting with a brief correction to past calculations in the literature, and then moving onward to the new case studies and the comparison of defense policies

optimized with Cut-The-Rope, versus a heuristic common-sense defense policy.

To assess the game w.r.t. a real-life application, we conducted two case studies on the industrial robots in Section 10.2.2. The game is similar to capture-the-flag competitions known in ethical hacking, since there and also here, the goal is to “capture” a target asset  $v_0$ . Our analysis, different from ethical hacking, is purely game-theoretic and optimization-based here.

### 10.2.1 Implementation Remarks

We adapted the implementation from [286] and thereby discovered a few bugs in this older code that we corrected in our version. The original code used fictitious play on the full distribution  $U = (u_1, u_2, \dots, u_n = \Pr(\text{adversary reaches } v_0))$  obtained from equation (B.5).

We compute an optimum w.r.t. a lexicographic order from right to left, first minimizing the last coordinate  $u_n$ , and breaking ties by continuing to minimize  $u_{n-1}$  while keeping  $u_n$  at minimum. The next tie is broken using  $u_{n-2}$ , while keeping the so-far optimized coordinates at their minima and so on. This introduces a dependence on the ordering of the coordinates, corresponding to a likewise ordering of locations in the attack graph. Therefore, the solution returned by the implementation from [286] is ambiguous in the sense of depending on the node ordering.

The optimization, however, independently of the node order, always minimizes the chances to reach the target asset, and hence provides a valid defense policy w.r.t. the targets of the defender. Our implementation inherits this dependence on the node order, but since our sole interest is reaching or avoiding to reach  $v_0$  anyway as (B.7) and (B.8) define, this ambiguity is not a limitation. In light of this, we chose the graph-topological sorting to order the probabilities in  $U'$  other than for  $v_0$ , which is the last element in this vector.

The attack graphs for our robot case studies have several entry points for the attacker and also several targets to reach. To handle them all in a single run of the analysis, we *added an artificial* (virtual) *entry node* from which all (real) entry nodes are trivially reachable (with probability 1). Since Cut-The-Rope in the original version, analyzed here, assumes only one target, we *contracted the multitude of target nodes* into a single “compound” target node. This corresponds to the target being to reach any of the possible target nodes, not distinguishing which in particular. A *target*

node is, by default in our implementation, any node that does not have descendants (zero out-degree in the attack graph). Consequently, all inner nodes, except the virtual start, are possible defense spot-check locations. The technical simplification towards having one target (only) is to avoid multi-criteria optimization, which is theoretically possible (even supported by the packages to run the optimization practically), but is more involved to interpret for a defense policy.

### 10.2.2 Robot Case Studies

For both of the robot cases to follow, we give computational results and a discussion of their practicality. To avoid confusion between the attack graphs appearing here and those found in the cited literature, we use the original versions thereof to visualize the battlefield and results. The actual simulation was done on an attack graph with added virtual starting and a single compound target node (if more than one exists).

It is perhaps practically interesting to remark that both attack graphs have inner nodes that classify as attack targets, but have descendant nodes as subsequent attack targets. With the convention of taking nodes with zero out-degree in the graph as targets (see Section 10.2.1), the simulation will include all “inner” nodes as defense locations even though they may be attack targets too. This is not precluded by the game design, and may be interpreted as considering inner nodes as “intermediate targets” whose prevention may avoid subsequent final, perhaps more dangerous, attack targets. The game’s defense policy would then advise to prevent a certain attack sub-target in the attack graph, with a certain level of effort (expressed as likelihood). From a simulation perspective, including or excluding any node from the defense policy is a simple matter of defining the action set for the defender accordingly.

**Case #1: Modular Articulated Robotic Arm (MARA)** MARA is a collaborative robotic arm with ROS 2.0 in each actuator, sensor or any other representative module. Each module has native ROS 2.0 support, can be physically extended in a seamless manner and delivers industrial-grade features including synchronization, deterministic communication latencies, a ROS 2.0 software and hardware component life-cycle and more. Altogether, MARA empowers new possibilities and applications in the professional landscape of robotics. The use case considered contemplates the MARA

## 10. Determining the Attack Target with Game Theory

---

modular robot operating in an industrial environment while performing a pick & place activity. Details about MARA for this case study can be found in [287, 288].

**Case #2: MiR100 - Mobile industrial Robotics** The MiR100 autonomous mobile robot is advertised as a safe and cost-effective mobile robot that quickly automates your internal transportation and logistics. The robot claims to optimize workflows, freeing staff resources so you can increase productivity and reduce costs. A case study analyzing the cyber-resilience of MiR100 robots was conducted and documented at [289] and [290], which considered a single robot operating in a structured environment while connected to a local area network that gets compromised. Through the local area network, prior work demonstrated how an attacker could exploit vulnerabilities, pivoting across subsystems in the robot all the way into its safety system, disabling it fully in a remote manner.

For both robots, we took attack graphs out of industrial security assessments, which, in the particular case of MiR100, were also annotated with Common Vulnerabilities and Exposures (CVE) and CVSS information, which allows an assessment of the “hardness” of vulnerabilities along the attack path. Such annotations were not available for the MARA use case, which, in lack of such details, suggests an application of the Poissonian movement pattern of Section B.3. The more detailed attack graph for the MiR100 robot enables the consideration of probabilistic success on exploits as Section B.3.2 described.

### 10.3 Results and Comparison

To evaluate how much a game-theoretic defense may add to the security, we do not only give the absolute results from the simulations, but also compare them to a heuristic best-effort defense policy, described in Section 10.3.1. Its simulation is run likewise with each of the four movement patterns from Sections B.3, with the probability to reach  $v_0$  given for each case as (i) optimized by Cut-The-Rope versus (ii) according to a best-effort defense.

#### 10.3.1 Baseline Comparison: A Best-Effort defense Policy

For an assessment of the quality of the game theoretic defense, let us use the following heuristic defense policy to compare:

- We assume that an adversarial avatar will always follow the shortest, or “easiest” attack path towards  $v_0$ . The distinction between shortest and easiest is made in dependence of how much is known about exploit complexities. In the MARA use case, the path choice will be for shortest, in terms of the number of exploits, since there is no further detail given about the exploit complexities. In the more detailed MiR100 use case, we have attack complexities and can likewise apply a shortest path algorithm to guide the attacker to the path whose success probability (as the product of all exploit success probabilities) is maximal <sup>2</sup>.
- The defender, unbeknownst of where the attacker is, and unable to actively detect it, applies a uniformly random defense strategy. That is, if the attacker is equally likely to be anywhere in the system, the defense policy would likewise be a uniformly random spot checking.

Under these hypotheses, we apply the same mechanism as in Cut-The-Rope, i.e., we let the attacker follow its chosen (shortest/easiest) path, and be occasionally sent back by the defender upon a coincidental cut of the path equivalently, closure of any backdoor. If so, then the avatar will keep retrying, until it hits the final target  $v_0$ . Note that this regime also includes lateral movement, since we still have a multitude of avatars attacking in parallel, each on its individually optimal route from its starting location  $\theta$  towards  $v_0$ .

We implemented this defense policy simulation by adapting the code from the implementations of Cut-The-Rope accordingly, to implement the heuristic defense and attack policy of above. Like for the game optimization, the heuristic defense implementation outputs the probability to reach  $v_0$  by simulating this defender-attacker interaction. We remark that this heuristic defense may still be overly optimistic relative to real life situations, in which defense teams may have only an incomplete view on the attack graph  $G = (V, E)$ . The defender would thus only be active on a subset  $D \subset V$ , so that all nodes in  $V \setminus D$  would be zero-day exploits.

---

<sup>2</sup>The usual trick of assigning the negative logarithm of probabilities as edge weights and computing a shortest path in the well known way

### 10.3.2 Overview of Experiments

In total, comparing the periodic/exponential defense strategy against a randomly moving adversary in two use cases, gives a total of 4 evaluation scenarios, each accompanied with its own comparison to the baseline heuristic of Section 10.3.1. Table 10.1 relates the sections and figures in the following to these four configurations.

<b>use case</b>	defender's policy: <b>periodic</b>	defender's policy: <b>exponential strategy</b>
MARA (no particular exploit hardness annotations), Sec. 10.3.3	attacker movement model: B.3.1 results shown in: Fig. 10.3	attacker movement model: B.3.3 results shown in: Fig. 10.4
MiR100 (known exploit complexities to consider), Sec. 10.3.4	attacker movement model: Sec. B.3.2 results shown in: Fig. 10.5	attacker movement model: Sec. B.3.4 results shown in: Fig. 10.6

Table 10.1: Overview of Experiments

### 10.3.3 MARA: Results

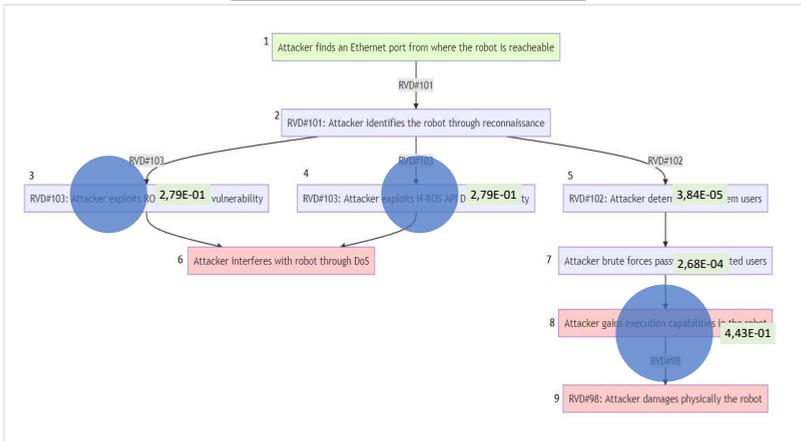
The attack graph for the MARA robot is taken from [288] and shown in Figures 10.3 and 10.4. This graph has 11 nodes and 10 edges in total, among them one entry point (node ①) for the attacker, and two targets (nodes ⑥, and ⑨).

We played Cut-The-Rope on this graph with a periodic defender versus an attacker that takes an average of 2 moves per time unit (i.e., in-between two appearances of the defender, e.g., per day). Figure 10.3 shows a table with the probabilities to spot-check each node on the attack graph. For the visualization, we have put bubbles on the attack graph, whose size corresponds to the probability of spot-checking there. That is, the larger the bubble, the more effort should be out on defending at this point.

Turning to the case of the defender coming back in random time intervals, we let the game run in three configurations, with the defender moving slower ( $\lambda_D = 1 < \lambda$ ), at equal speed ( $\lambda_D = \lambda = 2$ ) and faster than the attacker ( $\lambda_D = 3 > \lambda$ ) in (B.12). The resulting spot checking

Optimal defense policy:

Node	Probability
2	0
3	0,279
4	0,279
5	0,0000384
7	0,000268
8	0,443



Efficacy of the optimal vs. heuristic defense:

defense policy	chance to hit $v_0$
Cut-The-Rope	12,8%
heuristic	30,6%

Figure 10.3: MARA use case results for periodic spot checks (Section B.3.1). Adapted from [283].

probabilities are again displayed as bubbles located at the respective nodes in the attack graph, and put over one another in Figure 10.4.

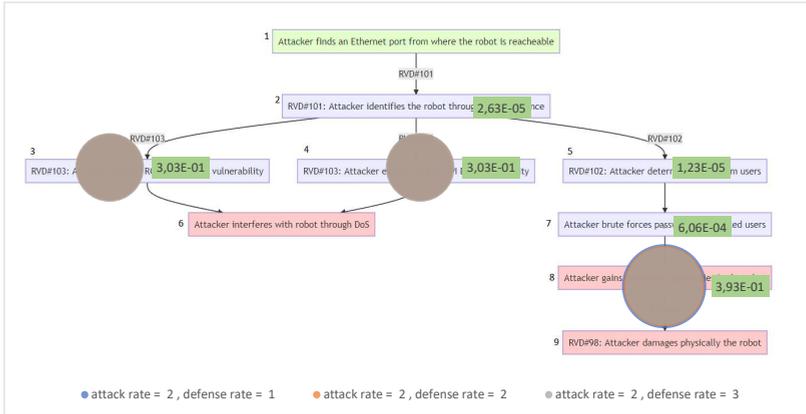
The numbers and bubbles are almost of the same size, showing that for the defense locations, the speed of spot checking has only a negligible impact, while the performance of the defense accordingly becomes better if the defender is “more active”. The performances of the defense policy as displayed in the bottom table of Figure 10.4 show that the optimized defense pays over the heuristic “blind” spot checking policy.

The takeaways from these findings is not that a more intense defense activity will reduce the chances of the attacker (this would be obviously

## 10. Determining the Attack Target with Game Theory

Optimal defense policy:

Node	probability for $\lambda_A = 2$ and		
	$\lambda_D = 1$	$\lambda_D = 2$	$\lambda_D = 3$
2	0,000147	0,0000502	0,0000263
3	0,272	0,291	0,303
4	0,272	0,291	0,303
5	0	0	0,0000123
7	0,000473	0,000548	0,000606
8	0,455	0,417	0,393



Efficacy of the optimal vs. heuristic defense:

defense policy	chances to hit $v_0$		
	$\lambda_D = 1$	$\lambda_D = 2$	$\lambda_D = 3$
Cut-The-Rope	7,4%	5,7%	4,6%
heuristic	17,8%	13,8%	11,2%

Figure 10.4: MARA use case results for spot checks at random intervals. Adapted from [283].

the case), but rather giving the defender an indication of where to allocate its (limited) resources to gain the best possible effect. Without signaling and without additional information in the attack graphs, the results are necessarily a crude approximation of reality, and Cut-The-Rope has been designed to be workable in such a situation of limited information, as well as with cases when more details are available, such as for the MiR100 robot following next. The results in the rather little detailed MARA use case are quite evident but therefore also plausible (“guard the closest graph cut between the asset and the defender”). The nontrivial indication here is the advice to let the attacker come “close” to the asset, while

a defender would perhaps otherwise try to guard the outer perimeter of the system to keep the intruder out in first place. The optimum to be at the closest graph-cut towards the asset is here explainable by our assumption that the attacker is stealthy and can start from anywhere, and in a practical situation, the defender may indeed have no reliable information about infected parts (otherwise, it would be trivial to disconnect and repair/replace the malfunctioning component). The defense policy that Cut-The-Rope computes is for practitioners operating blue teams that need to protect a large attack surface with no monitoring or signaling. A game-theoretic defense can help prioritize resources.

### 10.3.4 MiR100: Results

Similarly as for MARA, we used an attack graph for the MiR100 robot as shown in Figures 10.5 and 10.6. The attack graph has 16 nodes and 24 edges. The attacker can enter at four points (nodes ①...④), and four targets (⑫, ⑬, ⑭ and ⑯).

We conducted the likewise experiments under the same configurations as for the MARA use case, but this time making use of the CVE annotations to give information on how hard it is for the attacker to mount an exploit. For the defender, we again assume this one to be periodically active (as in Section B.3.1) and to randomly spot check (as in Section B.3.4). Note that in this case we do not have an attack rate  $\lambda$  as for the MARA use case before, since the movement of the attacker is solely governed by the difficulty to mount exploits.

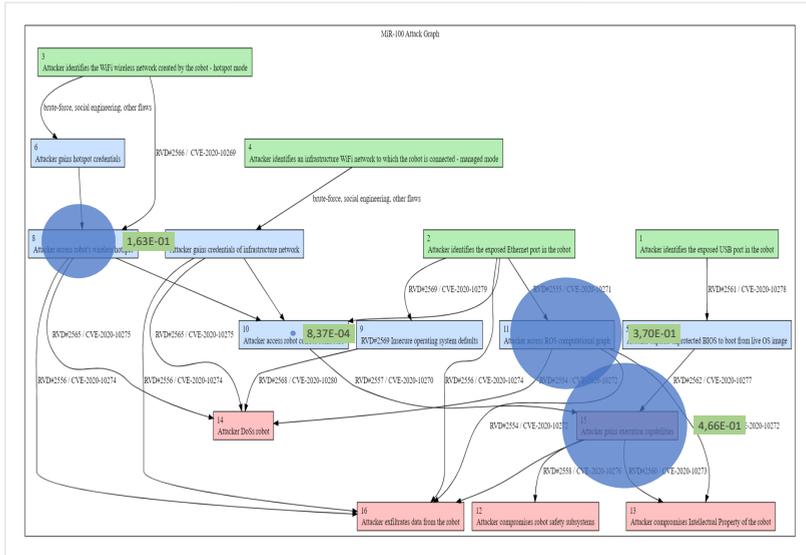
It is interesting to note that the optimal defense policy does not advise to guard node ⑦ or ⑨, which is a way towards reaching goal node ⑭. This may be assumption of the game, of the attacker already being somewhere in the network. The defense policy accounts for this and hence does not put more weight on lower nodes with higher incidence index. This way, the model accounts for defense in depth rather than entry prevention.

Finally, let us turn to the case of the adversary working towards  $v_0$  only in random time intervals between two appearances of the defender. This time, the defender's parameter  $\lambda_D$  is the average "window size"  $W$  (see Appendix B section B.6), measured in units of time, e.g., days. It is the time that we give the attacker to mount activities in the game. The results are shown in Figure 10.6.

## 10. Determining the Attack Target with Game Theory

Optimal defense policy:

Node	Probability
5	0
6	0
7	0
8	0,163
9	0
10	0,000837
11	0,37
15	0,466



Efficacy of the optimal vs. heuristic defense:

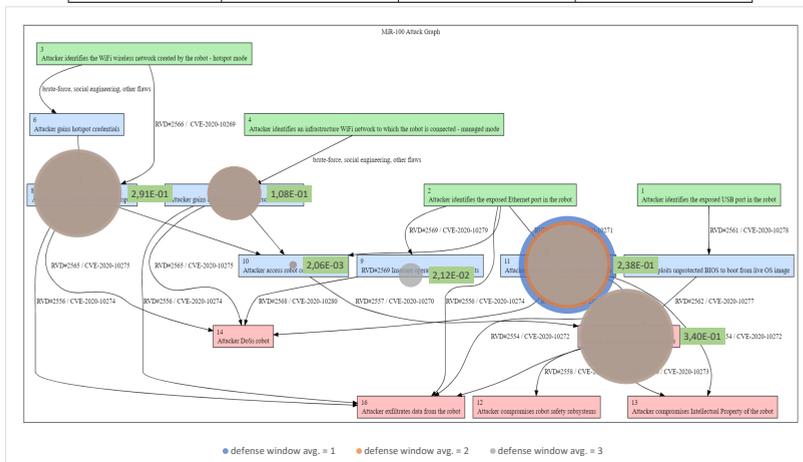
defense policy	chance to hit $v_0$
Cut-The-Rope	7,7%
heuristic	29,8%

Figure 10.5: MiR100 use case results for periodic spot checks (Section B.3.1). Adapted from [283].

Similar as for the MARA use case, the defense locations are the same in all cases, with the defense efforts only slightly differing according to how large the window is for the attacker, respectively, how frequently the defender comes back. The performance of the defense is shown in the bottom table of Figure 10.6. Consistent with the intuition, the

Optimal defense policy:

Node	probability for		
	$\lambda_D = 1$	$\lambda_D = 2$	$\lambda_D = 3$
5	0	0	0
6	0	0	0
7	0,101	0,108	0,108
8	0,249	0,281	0,291
9	0	0	0,0212
10	0,00194	0,00161	0,00206
11	0,354	0,28	0,238
15	0,294	0,33	0,34



Efficacy of the optimal vs. heuristic defense:

defense policy	chances to hit $v_0$		
	$\lambda_D = 1$	$\lambda_D = 2$	$\lambda_D = 3$
Cut-The-Rope	2.4%	3.5%	4.1%
heuristic	11%	16.1%	19%

Figure 10.6: MiR100 use case results for spot checks at random intervals (Section B.3.4). Adapted from [283].

attacker's chances to reach  $v_0$  become larger if the defense window is made larger. In both, the experiments with the heuristic defense and optimized under Cut-The-Rope, the value  $\lambda_D$  gives the average number of time units before the defender comes back. That is, larger  $\lambda_D$  give the attacker more time to exploit (conversely to the interpretation of  $\lambda_D$  in the other experiments, where it was the frequency of the defender's return). Again, the experiments show that Cut-The-Rope outperforms the heuristic defense considerably.

### 10.4 Discussion

The experimental findings suggest that the apparent optimal defense strategy delivered by Cut-The-Rope is to guard the immediate neighborhood of the target asset, so as to cover cases where the attacker has already deeply penetrated the system when the game begins. Indeed, an analytic characterization of the optimal defense under Cut-The-Rope is obtained in Appendix B section B.4 as Proposition B.4.3. It confirms a certain graph cut to be optimal under certain assumptions, but not in all of our test cases. For this reason, we leave the discussion of analytic results as an appended remark here, and continue the discussion with more practical aspects.

#### 10.4.1 Incomplete Attack Graphs and Zero-Day Exploits

The heuristic defense of Section 10.3.1 may in reality be still over-optimistic, in its assumption of complete knowledge about the attack graph. Practical defense teams may only have a limited knowledge or possibility to construct the entire attack graph, and it is generally unaccomplishable for the defender to get exactly the same attack graph as the adversary has. The simulations implemented in this chapter have been made with the possibility to include only a randomly chosen subset of nodes in the defender's possibilities to spot check, so include such incomplete knowledge in the analysis. Concretely, the code was made to randomly reduce the defender's spot check locations to, for example, only 75% of the nodes in the attack graph. Under such reduced possibilities, the game runs against an attacker with more, i.e., full, knowledge about the attack graph. We confine ourselves here to reporting that the defense policies performed worse than under full knowledge (not surprisingly), but both policies (Cut-The-Rope

and the heuristic) lost performance at approximately equal magnitudes, leaving their relative quality over one another without substantial changes.

We emphasize that a simulation under such reduced knowledge for the defender, whereas giving the attacker full knowledge, can be viewed as a study of the impact of zero-day exploits used by the attacker. That is, any node excluded from the defense, but used by the attacker is nothing else than a zero-day vulnerability. Since a systematic account for this would be beyond the scope (and space limits) of this work, we will explore this route along future work.

#### **10.4.2 Cutting the rope vs. Changing the Attack Graph**

In our experimental instance of the game, we let the attack graph remain unchanged over time. In particular, we assume that none of the defender's actions causes a permanent removal of a certain backdoor. This is practically motivated by the fact that spot checking may remove some, but not all vulnerabilities, so that, for example, one buffer overflow vulnerability in a secure shell implementation might get fixed, but other exploits of the same kind remain open, making the respective nodes remain unchanged in the graph after an inspection. Likewise, remote shell access may be required for the business workflow and hence cannot be deactivated, but only the access credentials might be updated. In that case, the remote shell access exists before and after an inspection. Even though the game model itself uses a static attack graph (see Figure 10.2), this one may itself require an update from time to time upon changes in the infrastructure. This is part of the business continuity management related to security, and accordingly changes the action sets for the defender and attacker. The implementation of the game, however, remains unaffected, except for the specification/input of the attack graph. We close the discussion at this point, referring to Appendix B section B.5 for a continuation of this discussion.

#### **10.4.3 Further Generalizations**

The movement patterns as studied admit further modifications and generalizations, yet to be explored, such as:

### Probabilistic Success on Spot Checks

First, to the advantage of the attacker, suppose that the defender is not necessarily successful on wiping out the adversary inside a node  $c$ , which may be the more “probable” case in an enterprise or embedded network. It is not difficult to generalize the model towards this: If we write  $p_c$  for the likelihood to actually cut the rope at  $c \in V$  upon trying so, (B.5) becomes a mix of cut and uncut paths,

$$\begin{aligned} \Pr(\text{adversary's location} = v) = & \\ & p_c \cdot \Pr(\text{adversary's location} = v | V(\pi|_c)) \\ & + (1 - p_c) \cdot \Pr(\text{adversary's location} = v | V(\pi)), \end{aligned}$$

and (B.5) is defined alike by the entirety of all these values for all  $v \in V$ . Thus, the computation as such does not change, only the code needs to use the above formulas to compute the payoffs. If the probabilities are made conditional on the system state, the analysis can be made to account for changing system conditions too.

### Multiple Adversarial Targets

Cut-The-Rope may be modified towards a multi-criteria game, treating all target nodes as individual targets in the game. The concept of a security strategy has a multi-goal counterpart, which the software used for the experimental implementation already supports. The experiments reported here could, possibly, be re-conducted without the merge of targets, i.e., the graph-theoretic contraction.

#### 10.4.4 Complexity and Scalability

The complexity of the analysis is governed by the time to solve a sequence of linear optimizations. The dimensions of these problems depend on the number of strategies for both players. Using interior point methods, the computational complexity is  $O(|AS_1| \cdot p(|AS_1| \cdot |AS_2|))$  for a polynomial  $p$  that depends on the chosen optimization algorithm. The need for an exhaustive enumeration of attack paths can raise scalability issues, since the number of attack paths is worst-case exponential. However, the number of paths in an attack *graph* may become large only because many paths overlap in large portions, and the defender may consider using only a subset of paths that

cover all edges in the attack graph, so as to cover all known exploits (which is a polynomial number), rather than all possible paths (whose number is exponential). We did not implement such a dimensionality reduction, this modification of the model is a possible aisle of future studies. For the use cases in our research, the number of paths was sufficiently small to admit an exhaustive enumeration. Likewise is the number of paths feasibly small if the battlefield is an attack tree, rather than an attack graph. In any case, Cut-The-Rope itself does not conceptually change if the restriction is imposed only on the cardinality of the strategy sets to be polynomial in the number of nodes in the attack graph.

#### 10.4.5 Including Signals about Adversarial Activities

The model assumes zero information for the defender about where the adversary is located. Many real-life systems use intrusion detection, and other signaling means (here explicitly not to be understood in the game-theoretic sense of signaling games). If we can compile the entirety of indications about the adversary's activity into a weight  $\Pr(\theta) \neq 1/|AS_2|$  for location  $\theta$  to possibly start from. Higher values may be assigned where we have stronger indication of recent adversarial activity at location  $\theta$ .

### 10.5 Conclusions and Outlook

Cut-The-Rope has been designed for ease of use in applications with little information (such as exemplified with the MARA robot use case), but also situations where there is detailed information encoded in the attack graph (such as for the MiR100 use case), or even when adversarial indications are available from auxiliary security systems, such as intrusion detection or others (including is possible as outlined in Section 10.4.5). The accuracy hence depends on how much information we can bring into the game, being a rather crude approximation for MARA, but much more fine-grained for the MiR100 use case. In both cases, however, the defender gets nontrivial advice on where to allocate its typically scarce resources for a best defense, beyond just guarding a graph-cut or choke point towards the critical asset  $v_0$ . When there are several such (evident) critical regions in the network to defend, different choke points may be of different criticality, depending on how many attack scenarios (each executed by another avatar in Cut-The-Rope) actually make use of this area in the attack graph.

The results obtained show some limitations for the practical use. First, and most substantially, the results depend on the ordering of the nodes, and – in addition to the general non-uniqueness of equilibria – hence may be ambiguous for the defender, leaving a residual chance of there being other defense possibilities. The algorithms applied in this work give only one solution, among perhaps many others. Second, the setting of probabilities from CVE, CVSS or likewise annotations is a nontrivial matter on its own, with only few first steps towards a systematic and sound derivation available in the literature [291]. The assumption of invisibility of the intruder can perhaps be weakened by including signals from intrusion detection or other side-information in the defender’s policy. The model simulated here does not include this possibility. Finally, the condensation of several attack goals into a single target node comes with the price of losing accuracy and some information about which attack goal may be more likely to be reached, thus making multi criteria optimization an interesting generalization to study.

Generally, Cut-The-Rope opens up an interesting class of games of mixed timing of moves between the actors, unlike as in extensive or normal form games, where players usually take actions in a fixed order. Likewise, and also different to many other game models, Cut-The-Rope has no defined start or finish for the defender (“security is never done”), while only one of the players knows when the game starts and ends, and the attacker can send its avatars from all possible locations in the network. The model is thus complementary to FlipIt, while it allows the attacker to spend any amount of time in the system, as long as the vital asset remains out of reach. This is actually to reflect the reality of security management: we cannot keep the adversary out, we can only try keeping as far away as possible.

In conclusion, Cut-The-Rope work presented in this chapter epitomizes the dynamic interplay between attack and defense in the realm of robotic cybersecurity, embodying a nuanced approach to resource allocation and threat anticipation. Its adaptability across various information landscapes—from sparse to rich data scenarios—provides a robust framework for decision-making in cybersecurity. While acknowledging its inherent limitations and the non-uniqueness of solutions, Cut-The-Rope underscores the perpetual nature of cyber defense. It represents a significant stride in the evolution of cybersecurity models, transitioning from static defense strategies to a more proactive, predictive stance. This model not only facilitates effective defense posturing but also enriches our understanding of the complex cyber battlefields, filled with automation, and where the

security landscape is constantly reshaped by emerging technologies and sophisticated adversaries.



## Chapter 11

# Guiding Attacks with Machine Learning

**Contributions:** *One of the top two contributors at [292]. Co-led software contributions and security methodology. All artifacts have been open sourced and results reproducible. Note various LLM models used within this work are not deterministic. This may difficult obtaining identical results.*

---

In the rapidly evolving field of offensive robot cybersecurity, particularly in automated systems (referenced in Part V), the challenge lies in effectively comprehending and navigating the complex threat landscape. Following the previous Chapter 10, which focused on automating attack vector selection through game theory, we now turn to the task of guiding an autonomous agent. This involves choosing the correct sequence of actions (like reconnaissance, exploitation, and exfiltration) and appropriate payloads to effectively utilize the chosen attack vector. This necessitates a fusion of in-depth cybersecurity knowledge with specialized domain expertise, such as robotics, each bringing unique challenges and technological nuances.

In this chapter, we delve into the sophisticated realm of guiding autonomous cybersecurity agents, whether for offensive or defensive purposes. The challenge lies in the enormity and obscurity of the cybersecurity state and action spaces, coupled with the unpredictability of the environment. While traditional machine learning approaches, including Reinforcement Learning, have demonstrated proficiency in learning basic policies, their effectiveness is often limited to the data sets they are

trained on, lacking the capacity to generalize across diverse cybersecurity scenarios.

This chapter introduces a novel method for autonomously steering cybersecurity agents towards specific objectives. Our approach harnesses the power of foundational models and Large Language Models (LLMs), which encompass a broad spectrum of general-purpose knowledge. These models are particularly adept at parsing, reasoning, and generating security-related artifacts, thereby playing a pivotal role in guiding cybersecurity exercises. We specifically focus on the application of Generative Pre-trained Transformer (GPT) models [293, 294], with our prototype, PentestGPT, demonstrating remarkable capabilities. PentestGPT leverages the extensive knowledge base and analytical prowess of LLMs to offer innovative solutions and insights in the field of robotic cybersecurity.

Through this exploration, we aim to showcase how leveraging advanced machine learning techniques can profoundly impact the field of cybersecurity, particularly in automating and refining attack strategies. This chapter not only highlights the potential of these technologies but also sets the stage for future developments in the automated orchestration of cybersecurity tactics.

### 11.1 Introduction

Guaranteeing a system's immunity to potential attacks is a formidable challenge. Offensive security methods, such as penetration testing (pen-testing) or red teaming, have become essential in the security lifecycle. As detailed by Applebaum [14], these methods require security teams to attempt breaches of an organization's defenses to uncover vulnerabilities. They offer marked advantages over traditional defensive mechanisms, reliant on incomplete system knowledge and modeling. Guided by the principle "*the best defense is a good offense*", this study focuses on offensive strategies, particularly *penetration testing*.

Penetration testing [295] is a proactive offensive technique aiming at identifying, assessing, and mitigating as many security vulnerabilities as possible. This involves executing targeted attacks to confirm diverse flaws (e.g., erratic behaviors) and is efficacious in creating a comprehensive inventory of vulnerabilities complemented by actionable enhancement recommendations. As a widely-employed practice for security appraisal, penetration testing empowers organizations to discern and neutralize

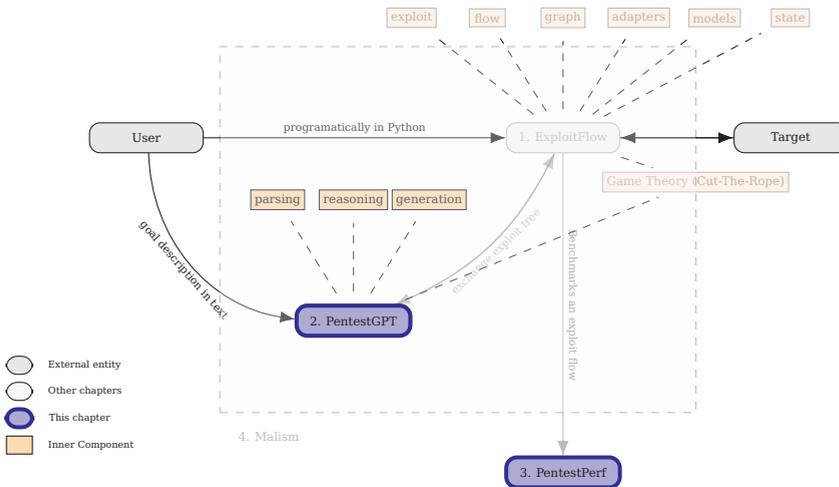


Figure 11.1: Machine Learning contributions to our architecture to develop fully automated offensive robot cybersecurity agent, a cognitive cybersecurity engine denominated Malism.

potential vulnerabilities in their networks and systems before exploitation by malicious entities. Despite its significance, the industry often leans on manual techniques and specialized knowledge [296], making it labor-intensive. This has generated a gap in responding to the escalating demand for adept and efficient security evaluations.

Recently Large Language Models (LLMs) [297, 298] are making striking progress, exhibiting an increasingly nuanced understanding of human-like text and effectively executing various tasks across diverse domains. One intriguing aspect of LLMs is their emergent abilities [299], which are not explicitly programmed but arise during the training process. These abilities enable LLMs to perform complex tasks such as reasoning, summarization, question-answering, and domain-specific problem-solving without requiring specialized training. Such capabilities indicate the transformative potential of LLMs across various sectors, including cybersecurity. A critical question thus emerges: can LLMs be leveraged in cybersecurity, particularly for performing automated penetration testing?

Motivated by this question, we set out to evaluate the capabilities of LLMs on real-world penetration testing tasks. Unfortunately, the current

benchmarks for penetration testing [300, 301] are not comprehensive and fail to assess progressive accomplishments fairly during the process. To address this limitation, we construct a robust benchmark that includes test machines from HackTheBox [302] and VulnHub [303]—two leading platforms for penetration testing challenges. Comprising 13 targets with 182 sub-tasks, our benchmark encompasses all vulnerabilities appearing in OWASP’s top 10 vulnerability list [304]. Also, it offers a more detailed evaluation of the tester’s performance by monitoring the completion status for each sub-task.

Armed with this benchmark, we conduct an exploratory study using GPT-3.5 [305], GPT-4 [306], and Bard [307] as representative LLMs. We interactively test these models by guiding them to complete the penetration tasks against our benchmark targets. This interaction involves setting a penetration testing goal for the LLM, soliciting it for the appropriate operation to execute, implementing it in the testing environment, and feeding the test outputs back to the LLM for next-step reasoning (Figure 11.2). By repeating this cycle, we derive the final penetration testing results. To evaluate the performance of the LLMs, we compare their results against baseline solutions provided by official walkthroughs and solutions from certified penetration testers. By analyzing similarities and differences in their problem-solving approaches, we aim to better understand LLMs’ penetration testing capabilities and discern how their problem-solving strategies diverge from those of human experts.

Our investigation yields intriguing insights into the capabilities and limitations of LLMs in penetration testing. We discover that LLMs demonstrate proficiency in managing specific sub-tasks within the testing process, such as utilizing testing tools, interpreting their outputs, and suggesting subsequent actions. Compared to human experts, LLMs are especially adept at executing complex commands and options with testing tools, while models like GPT-4 excel in comprehending source code and pinpointing vulnerabilities. Furthermore, LLMs can craft appropriate test commands and accurately describe graphical user-interface operations needed for specific tasks. Leveraging their vast knowledge base, they can design inventive testing procedures to unveil potential vulnerabilities in real-world systems and CTF challenges. However, we also note that LLMs have difficulty in maintaining a coherent grasp of the overarching testing scenario, a vital aspect for attaining the testing goal. As the dialogue advances, they may lose sight of earlier discoveries and struggle to apply

their reasoning consistently toward the final objective. Additionally, LLMs might overemphasize recent tasks in the conversation history, regardless of their vulnerability status. As a result, they tend to neglect other potential attack surfaces exposed in prior tests and fail to complete the penetration testing task.

The outcomes of our empirical study are promising, revealing that LLMs possess the necessary domain knowledge to perform penetration testing tasks. In particular, they are great at providing *an intuition* of what to do in a given networking scenario. However, what they lack is effective guidance to carry out these tasks independently and maintain a cohesive grasp of the testing scenario. On the other hand, as investigated in a prior research publication [20] focused on capturing the exploitation route (or flow) for automation. Given the complexity of the (network) state space, the state itself is not enough to reason about what are the best actions to pentest. It rapidly becomes evident that a heuristic is needed to support autonomous pentesting which helps pick actions to achieve given goals. With this understanding, we aim to contribute unlocking the potential of modern machine learning approaches and develop a fully automated penetration testing framework that helps produce cybersecurity cognitive engines. Our overall architecture is depicted in Figure 11.1, which shows our current work so far and near future planned contributions. Our proposed framework, Malism, is designed to enable a user without in-depth security domain knowledge to produce its own cybersecurity cognitive engine that helps conduct penetration testing over an extensive range of targets. This framework comprises three primary components:

1. ExploitFlow [20]: A modular library to produce cyber security exploitation routes (*exploit flows*). ExploitFlow aims to combine and compose exploits from different sources and frameworks, capturing the state of the system being tested in a flow after every discrete action which allows learning attack trees that affect a given system. ExploitFlow's main motivation is to facilitate and empower Game Theory and Artificial Intelligence (AI) research in cyber security. It provides a unique representation of the exploitation process that encodes every facet within it. Its representation can be effectively integrated with various penetration testing tools and scripts, such as Metasploit [308] to perform end-to-end penetration testing. Such representation can be further visualized to guide the human experts

for the reproduction of the testing process.

2. PentestGPT (**this chapter**): An automated penetration testing system that leverages the power of LLMs to produce testing guidance and intuition at every given discrete state. It functions as the core component of the Malism framework, guiding the LLMs to efficiently utilize their domain knowledge in real-world testing scenarios.
3. PentestPerf: A comprehensive penetration testing benchmark developed to evaluate the performances of penetration testers and automated tools across a wide array of testing targets. It offers a fair and robust platform for performance comparison.

The harmonious integration of these three components forms an automated, self-evolving penetration testing framework capable of executing penetration tests over various targets, Malism. This framework to develop fully automated penetration testing tools, which we named back in Chapter 11 *cybersecurity cognitive engines*, aims to revolutionize the field of penetration testing by significantly reducing the need for domain expertise and enabling more comprehensive and reliable testing.

Building on our insights into LLMs' capabilities in penetration testing, we present PentestGPT, an interactive system designed to enhance the application of LLMs in this domain. Drawing inspiration from the collaborative dynamics commonly observed in real-world human penetration testing teams, PentestGPT is particularly tailored to manage large and intricate projects. It features a tripartite architecture comprising Reasoning, Generation, and Parsing Modules, each reflecting specific roles within penetration testing teams. The Reasoning Module emulates the function of a lead tester, focusing on maintaining a high-level overview of the penetration testing status. We introduce a novel representation, the Pentesting Task Tree (PTT), based on the cybersecurity attack tree [309]. This structure encodes the testing process's ongoing status and steers subsequent actions. Uniquely, this representation can be translated into natural language and interpreted by the LLM, thereby comprehended by the Generation Module and directing the testing procedure. The Generation Module, mirroring a junior tester's role, is responsible for constructing detailed procedures for specific sub-tasks. Translating these into exact testing operations augments the generation process's accuracy. Meanwhile, the Parsing Module deals with diverse text data encountered

during penetration testing, such as tool outputs, source codes, and HTTP web pages. It condenses and emphasizes these texts, extracting essential information. Collectively, these modules function as an integrated system. PentestGPT completes a complex penetration testing task by bridging high-level strategies with precise execution and intelligent data interpretation, thereby maintaining a coherent and effective testing process.

We evaluate PentestGPT using our benchmark to showcase its efficacy. Specifically, our system achieves remarkable performance gains, with 228.6% and 58.6% increases in sub-task completion compared to the direct usage of GPT-3.5 and GPT-4, respectively. We also apply PentestGPT to the HackTheBox active penetration testing machines challenge [310], completing 4 out of the 10 selected targets at a total OpenAI API cost of 131.5 US Dollars, ranking among the top 1% players in a community of over 670,000 members. This evaluation underscores PentestGPT's practical value in enhancing penetration testing tasks' efficiency and precision. The solution has been made publicly available on GitHub <sup>1</sup>, receiving widespread acclaim with over 4,700 stars to the date of writing, active community engagement, and ongoing collaboration with multiple industrial partners. In summary, we make the following contributions:

- **Development of a Comprehensive Penetration Testing Benchmark.** We craft a robust and representative penetration testing benchmark, encompassing a multitude of test machines from leading platforms such as HackTheBox and VulnHub. This benchmark includes 182 sub-tasks covering OWASP's top 10 vulnerabilities, offering fair and comprehensive evaluation of penetration testing.
- **Empirical Evaluation of LLMs for Penetration Testing Tasks.** By employing models like GPT-3.5, GPT-4, and Bard, our exploratory study rigorously investigates the strengths and limitations of LLMs in penetration testing. The insights gleaned from this analysis shed valuable light on the capabilities and challenges faced by LLMs, enriching our understanding of their applicability in this specialized domain.
- **Development of an Innovative LLM-powered Penetration Testing System.** We engineer PentestGPT, a novel interactive system that leverages the strengths of LLMs to carry out penetration testing tasks au-

---

<sup>1</sup>For anonymity during the review process, we have created an anonymous repository to open-source our solution [311].

tomatically. Drawing inspiration from real-world human penetration testing teams, PentestGPT integrates a tripartite design that mirrors the collaborative dynamics between senior and junior testers. This architecture optimizes LLMs' usage, significantly enhancing the efficiency and effectiveness of automated penetration testing.

### 11.2 Background

#### 11.2.1 Penetration Testing

Penetration testing, or “pentesting”, is a critical practice to enhance organizational systems' security. In a typical penetration test, security professionals, known as penetration testers, analyze the target system, often leveraging automated tools. The standard process is divided into seven phases [312]: Reconnaissance, Scanning, Vulnerability Assessment, Exploitation, and Post Exploitation (including reporting). These phases enable testers to understand the target system, identify vulnerabilities, and exploit them to gain access.

Despite substantial efforts [12, 301, 313] in the field, a fully automated penetration testing pipeline remains elusive. The challenges in automating the process arise from the comprehensive knowledge needed to understand and manipulate various vulnerabilities and the demand for a strategic plan to guide subsequent actions. In practice, penetration testers often use a combined approach integrating depth-first and breadth-first search techniques [312]. They begin by obtaining an overarching understanding of the target environment (utilizing a breadth-first approach) before focusing on specific services and vulnerabilities (employing a depth-first approach). This strategy ensures a thorough system analysis while prioritizing promising attack vectors, relying heavily on individual experience and domain expertise. Additionally, penetration testing requires many specialized tools with unique features and functions. This diversity adds complexity to the automation process. Therefore, even with the support of artificial intelligence, creating a fully unified solution for automated penetration testing remains a formidable challenge.

## 11.2.2 Large Language Models

Large Language Models (LLMs), including OpenAI's GPT-3.5 and GPT-4, are prominent tools with applications extending to various cybersecurity-related fields, such as code analysis [314] and vulnerability repairment [315]. These models are equipped with wide-ranging general knowledge and the capacity for elementary reasoning. They can comprehend, infer, and produce text resembling human communication, aided by a training corpus encompassing diverse domains like computer science and cybersecurity. Their ability to interpret context and recognize patterns enables them to adapt knowledge to new scenarios. This adaptability, coupled with their proficiency in interacting with systems in a human-like way, positions them as valuable assets in enhancing penetration testing processes. Despite inherent limitations, LLMs offer distinct attributes that can substantially aid in the automation and improvement of penetration testing tasks. The realization of this potential, however, requires the creation and application of a specialized and rigorous benchmark.

## 11.3 Penetration Testing Benchmark

### 11.3.1 Motivation

The fair evaluation of Large Language Models (LLMs) in penetration testing necessitates a robust and representative benchmark. Existing benchmarks in this domain [300, 301] have several limitations. First, they are often restricted in scope, focusing on a narrow range of potential vulnerabilities, and thus fail to capture the complexity and diversity of real-world cyber threats. For instance, the OWASP benchmark *juiceshop* [316] is commonly adopted for evaluating web vulnerability testing. However, it does not touch the concept of privilege escalation, which is an essential aspect of penetration testing. Second, existing benchmarks may not recognize the cumulative value of progress through the different stages of penetration testing, as they tend to evaluate only the final exploitation success. This approach overlooks the nuanced value each step contributes to the overall process, resulting in metrics that might not accurately represent actual performance in real-world scenarios.

To address these concerns, we propose the construction of a comprehensive penetration testing benchmark that meets the following criteria:

**Task Variety.** The benchmark must encompass diverse tasks, reflecting various operating systems and emulating the diversity of scenarios encountered in real-world penetration testing.

**Challenge Levels.** To ensure broad applicability, the benchmark must include tasks of varying difficulty levels suitable for challenging novice and expert testers.

**Progress Tracking.** Beyond mere success or failure metrics, the benchmark must facilitate tracking of incremental progress, thereby recognizing and scoring the value added at each stage of the penetration testing process.

### 11.3.2 Benchmark Design

Following the criteria outlined previously, we develop a comprehensive benchmark that closely reflects real-world penetration testing tasks. The design process progresses through several stages.

**Task Selection.** Our first step is to meticulously select tasks from HackTheBox [302] (HTB) and VulnHub [303]. These platforms are widely recognized and frequently utilized for penetration testing practice. Our selection process is guided by a desire to incorporate a diverse and challenging set of tasks. Capture The Flag (CTF) exercises and real-world testing scenarios have been included. The targets are drawn from various operating systems and encompass a broad spectrum of vulnerabilities. This approach ensures a wide representation of real-world penetration testing tasks. To account for different skill levels, the selected tasks cover a broad range of difficulty. While HTB and VulnHub offer reference difficulty levels, we further validate these with input from three certified penetration testers<sup>2</sup>, including the authors of this work. This collaborative process yields a consensus on the final difficulty rating for each target, aligning with the conventional categorization [303] of penetration testing machines into *easy*, *medium*, and *hard* levels. It is worth noting that our benchmark does not explicitly include benign targets for evaluating false positives. This is because the iterative and exploratory nature of penetration testing inherently involves investigating services within the target that may ultimately be deemed benign. In this process, our primary focus is successfully identifying genuine vulnerabilities.

---

<sup>2</sup>Our penetration testers are all Offensive Security Certified Professionals (OSCP).

**Task Decomposition.** We further parse the testing process of each target into a series of sub-tasks, following the standard solution commonly referred to as the “walkthrough” in penetration testing. Each sub-task corresponds to a unique step in the overall process. Specifically, a sub-task may represent a micro-step involving the use of a particular penetration testing tool (e.g., performing port scanning with nmap [317]) or exploiting a unique vulnerability identified in the Common Weakness Enumeration (CWE) [318] (e.g., exploiting SQL injection). To standardize decomposition, we arrange the sub-tasks into a two-layer structure. Initially, we categorize each sub-task according to the five phases of penetration testing, as described in Section 11.2. Then, we label the sub-task with either the corresponding CWE item it targets or the specific tools employed. These two steps enable us to formulate an exhaustive list of sub-tasks for every benchmark target. We include this list in Appendix C, and the complete sub-task information is accessible on our anonymous open-source project [311].

**Benchmark Validation.** The final stage of our benchmark development involves rigorous validation. This step ensures that our benchmark accurately reflects real-world penetration testing scenarios and offers reproducibility. During validation, three certified penetration testers independently attempt the penetration testing targets, refining the sub-tasks as needed. We adjust our task decomposition accordingly because some targets may have multiple valid solutions.

By the end, we compile a benchmark of 13 penetration testing targets with 182 sub-tasks in 25 categories. The benchmark includes all types of vulnerabilities as listed in the OWASP [304] Top 10 Project. Detailed information on the included categories is listed in the Appendix C. To contribute to community development, we have made this benchmark publicly available online at our anonymous project website [311].

## 11.4 Exploratory Study

We conduct an exploratory study to assess the capabilities of LLMs in penetration testing. Our primary objective is determining how well LLMs can adapt to the real-world complexities and challenges associated with penetration testing tasks. Specifically, we aim to address the following two research questions:

**RQ1 (Capability):** To what extent can LLMs perform penetration testing tasks?

**RQ2 (Comparative Analysis):** How do the problem-solving strategies of human penetration testers and LLMs differ?

We utilize the benchmark described in Section 11.3 to evaluate the performance of LLMs on penetration testing tasks. In the following, we first delineate our testing strategy for this study. Subsequently, we present the testing results and an analytical discussion to address the above research questions.

### 11.4.1 Testing Strategy

LLMs cannot perform penetration tests directly. Their capabilities are primarily text-based, responding to queries and providing suggestions. However, penetration testing often involves operations with user interfaces (UI) and understanding graphical information, such as website images. This necessitates a bridge between the test machine and the LLM to facilitate task completion.

We introduce an interactive loop structure to evaluate the LLM's abilities in penetration testing within our benchmark. This process, depicted in Figure 11.2, consists of the following stages: ❶ We present the target information to the LLM and request recommendations for penetration testing actions. This initiates a looped testing procedure. ❷ We implement the actions suggested by the LLM, which encompass both terminal commands and graphical interactions. ❸ We gather the results of the actions. Text-based output, such as terminal responses or source code, is recorded directly. Human penetration testers provide concise summaries and descriptions for non-textual results (e.g., images). The summarized information is returned to the LLM to inform subsequent actions. ❹ This cycle continues until we identify a solution or reach a standstill. We compile a record of the testing procedures, encompassing successful tasks, ineffective actions, and any reasons for failure, if applicable.

### 11.4.2 Evaluation Settings

We proceed to assess the performances of various LLMs in penetration testing tasks using the strategy mentioned above.

**Model Selection.** Our study focuses on three cutting-edge LLMs that are currently accessible: GPT-3.5 and GPT-4 from OpenAI and LaMDA [319] from Google. These models are selected based on their prominence in

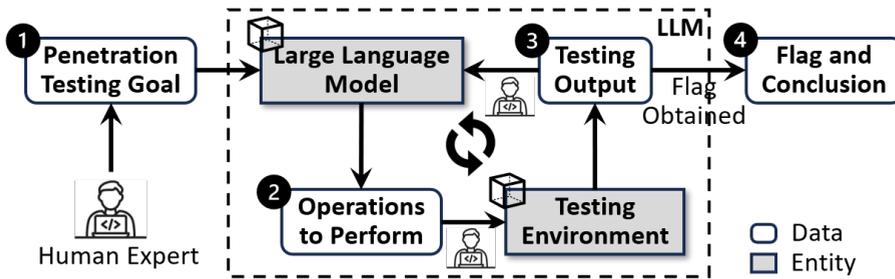


Figure 11.2: Overview of strategy to use LLMs for penetration testing. Reproduced from [292].

the research community and consistent availability. To interact with the LLMs mentioned above, we utilize chatbot services provided by OpenAI and Google, namely ChatGPT [320] and Bard [307]. For this chapter, the terms GPT-3.5, GPT-4, and Bard represent these three LLMs.

**Experimental Setup.** We conduct our experiments in a local environment where the target and testing machines are part of the same private network. The testing machine operates on Kali Linux [321], version 2023.1. Several measures are implemented to validate the effectiveness of our testing procedures. First, we repeat the tests to account for inherent variability in the LLM outputs. In particular, we test each target with each LLM five times. We performed 195 tests in total, i.e., 5 repetitions \* 3 models \* 13 targets, wherein the 13 targets correspond to the *Easy*, *Medium* and *Hard* target scenarios<sup>3</sup>. In this process, a sub-task is considered successful if it succeeds in at least one trial, and a penetration task is considered successful as long as one trial succeeds. Second, we make the best efforts to translate UI operations and graphical information into natural languages accurately. In addition, we ensure the precise execution of the instructions provided by the LLMs. Third, we maintain the integrity of the testing process by strictly limiting the tester’s role to executing actions and reporting results without adding expert knowledge or guidance. Finally, the testing and target machines are rebooted after each test to reset their states, ensuring a consistent starting point for each test.

**Tool Usage.** Our study aims to assess the innate capabilities of LLMs

<sup>3</sup>*Easy*, *Medium* and *Hard* is the categorization convention used in the security CTF community to represent the difficulty for a human to solve such challenge.

## 11. Guiding Attacks with Machine Learning

without reliance on automated vulnerability scanners such as Nexus [322] and OpenVAS [323]. Consequently, we explicitly instruct the LLMs to refrain from using these tools. However, we follow the LLMs’ recommendations for utilizing other tools designed to validate specific vulnerability types (e.g., sqlmap [324] for SQL injections). Occasionally, versioning discrepancies may lead the LLMs to provide incorrect instructions for tool usage. In such instances, our penetration testing experts evaluate whether the instructions would have been valid for a previous version of the tool. They then make any necessary adjustments to ensure the tool’s correct operation.

### 11.4.3 Capability Evaluation (RQ1)

Tools	Easy		Medium		Hard		Total	
	Overall (7)	Sub-task (77)	Overall (4)	Sub-task (71)	Overall (2)	Sub-task (34)	Overall (13)	Sub-task (182)
GPT-3.5	1 (14.29%)	24 (31.17%)	0 (0.00%)	13 (18.31%)	0 (0.00%)	5 (14.71%)	1 (7.69%)	42 (23.07%)
GPT-4	4 (57.14%)	52 (67.53%)	1 (25.00%)	27 (38.03%)	0 (0.00%)	8 (23.53%)	5 (38.46%)	87 (47.80%)
Bard	2 (28.57%)	29 (37.66%)	0 (0.00%)	16 (22.54%)	0 (0.00%)	5 (14.71%)	2 (15.38%)	50 (27.47%)
Average	2.3 (33.33%)	35 (45.45%)	0.33 (8.33%)	18.7 (26.29%)	0 (0.00%)	6 (17.64%)	2.7 (20.5%)	59.7 (32.78%)

Table 11.1: Overall performance of LLMs on Penetration Testing Benchmark.

To study **RQ1**, we begin by assessing the overall performance of three prominent LLMs: GPT-4, Bard, and GPT-3.5. The results of these evaluations are compiled in Table 11.1. The experimental results show that the three LLMs completed at least one end-to-end penetration testing task. This achievement underscores their ability to conduct a broad spectrum of testing operations, particularly within environments of less complexity. Among the models, GPT-4 stands out with superior performance, achieving success with 4 targets of easy difficulty and 1 of medium difficulty. Bard and GPT-3.5 also demonstrate commendable capabilities, completing 2 and 1 targets of easy difficulty, respectively. When examining sub-tasks, GPT-4 accomplishes 52 of 77 on easy difficulty targets and 27 out of 71 on medium ones, underlining its potential for significant contributions to more complex penetration testing scenarios. Though not as proficient as GPT-4, GPT-3.5 and Bard still show promise, completing 13 (18.31%) and 16 (22.54%) of sub-tasks on medium difficulty targets, respectively. However, the performance of all three models noticeably diminishes when challenged with hard difficulty targets. While each model can complete the initial reconnaissance phase on these targets, they fall short in exploiting the identified vulnerability. This outcome is not entirely unexpected, as the

hard difficulty machines are deliberately crafted to be exceedingly difficult. They often include services that appear vulnerable but are, in fact, non-exploitable—a trait commonly referred to as *rabbit holes* [325]. Additionally, the routes to successfully exploiting these machines are typically inventive and unforeseeable, making them resistant to straightforward replication by automated tools. For instance, the benchmark target *Falafel* involves deliberately crafted SQL injection vulnerabilities, which are resistant to *sqlmap* and can only be exploited through manually designed payloads. Existing LLMs do not exhibit the capability to solve them solely without the guidance of human experts.

**Finding 1:** Large Language Models (LLMs) have shown proficiency in conducting end-to-end penetration testing tasks but struggle to overcome challenges presented by more difficult targets.

Sub-Tasks	Walkthrough	GPT-3.5	GPT-4	Bard
General Tool Usage	25	4	10	7
Port Scanning	9	9	9	9
Web Enumeration	18	4	8	4
Code Analysis	18	4	5	4
Shell Construction	11	3	7	4
Directory Exploitation	11	1	7	1
General Privilege Escalation	8	2	4	3
Flag Capture	8	1	5	2
Password/Hash Cracking	8	2	4	2
Network Exploitation	7	1	3	2

Table 11.2: Top 10 Types of Sub-tasks completed by each tool.

We further examine the detailed sub-task completion performances of the three LLMs, as presented in Table 11.2. Analyzing the completion status, we identify several areas where LLMs excel. First, they adeptly utilize common penetration testing tools to interpret the corresponding outputs, especially in enumeration tasks correctly. For example, all three evaluated LLMs successfully perform all nine *Port Scanning* sub-tasks. They can configure the widely-used port scanning tool, *nmap* [317], comprehend the scan results, and formulate subsequent actions. Second, the LLMs reveal a deep understanding of prevalent vulnerability types, connecting them to the services on the target system. This understanding is evidenced by the successful completion of sub-tasks related to various vulnerability

## 11. Guiding Attacks with Machine Learning

---

Unnecessary Operations	GPT-3.5	GPT-4	Bard	Total
Brute-Force	75	92	68	235
CVE Study	29	24	28	81
SQL Injection	14	21	16	51
Command Injection	18	7	12	37

Table 11.3: Top Unnecessary Operations Prompted by LLMs on the Benchmark Targets. Experiments conducted on June 2023.

types. Finally, LLMs demonstrate their effectiveness in code analysis and generation, particularly in the tasks of *Code Analysis* and *Shell Construction*. These tasks require the models to read and generate codes in different programming languages, essential in penetration testing. This often culminates in identifying potential vulnerabilities from code snippets and crafting the corresponding exploits. Notably, GPT-4 outperforms the other two models regarding code interpretation and generation, marking it the most suitable candidate for penetration testing tasks.

**Finding 2:** LLMs can efficiently use penetration testing tools, identify common vulnerabilities, and interpret source codes to identify vulnerabilities.

### 11.4.4 Comparative Analysis (RQ2)

To address **RQ2**, we examine the problem-solving strategies that LLMs employ, contrasting them with human penetration testers. In each penetration testing trial, we concentrate on two main aspects: (1) Identifying the unnecessary operations that LLMs prompt, which are not conducive to successful penetration testing, as compared to a standard walkthrough; and (2) Understanding the specific factors that prevent LLMs from successfully executing penetration tests.

We analyze the unnecessary operations prompted by LLMs by breaking down the recorded testing procedures into sub-tasks. We employ the same method to formulate benchmark sub-tasks, as Section 11.3 outlines. By comparing this to a standard walkthrough, we identify the primary sub-task trials that fall outside the standard walkthrough and are thus irrelevant to the penetration testing process. The results are summarized in Table 11.3. We find that the most prevalent unnecessary operation prompted by LLMs is brute force. For all services requiring password authentication,

Failure Reasons	GPT3.5	GPT4	Bard	Total
Session context lost	25	18	31	74
False Command Generation	23	12	20	55
Deadlock operations	19	10	16	45
False Scanning Output Interpretation	13	9	18	40
False Source Code Interpretation	16	11	10	37
Cannot craft valid exploit	11	15	8	34

Table 11.4: Top causes for failed penetration testing trials

LLMs typically advise brute-forcing it. This is an ineffective strategy in penetration testing. We surmise that many hacking incidents in enterprises involve password cracking and brute force. LLMs learn these reports from accident reports and are consequently considered viable solutions. Besides brute force, LLMs suggest that testers engage in CVE studies, SQL injections, and command injections. These recommendations are common, as real-world penetration testers often prioritize these techniques, even though they may not always provide the exact solution.

We further investigate the reasons behind the failure of penetration testing trials. We manually categorize the causes of failure for the 195 penetration testing trials, with the results documented in Table 11.4. This table reveals that the predominant cause of failure is the loss of session context. The three examined models face difficulties in maintaining long-term conversational memory uniformly, frequently forgetting previous test results as the dialogue progresses. This lack of retention may be attributable to the limited token size within the LLM conversation context. Given the intricate nature of penetration testing—where a tester must skillfully link minor vulnerabilities across different services to develop a coherent exploitation strategy—this loss of context substantially undermines the models’ effectiveness.

**Finding 3:** LLMs struggle to maintain long-term memory, which is vital to link vulnerabilities and develop exploitation strategies effectively.

Secondly, LLMs strongly prefer the most recent tasks, adhering rigorously to a depth-first search approach. They concentrate on exploiting the immediate service, rarely deviating to a new target until all potential paths for the current one have been pursued. This can be attributed to the attention of LLMs focusing more on the beginning and end of the prompt,

## 11. Guiding Attacks with Machine Learning

---

as revealed in [326]. Experienced penetration testers generally assess the system from a broader standpoint, strategizing the subsequent steps likely to provide the most substantial results. When combined with the aforementioned memory loss issue, this tendency causes LLMs to become overly fixated on a specific service. As the test progresses, the models completely forget previous findings and reach a deadlock.

**Finding 4:** LLMs strongly prefer recent tasks and a depth-first search approach, often resulting in an over-focus on one service and forgetting previous findings.

Lastly, LLMs have inaccurate result generation and hallucination issues, as noted in [327]. This phenomenon ranks as the second most frequent cause of failures and is characterized by the generation of false commands. In our study, we observe that LLMs frequently identify the appropriate tool for the task but stumble in configuring the tools with the correct settings. In some cases, they even concoct non-existent testing tools or tool modules.

**Finding 5:** LLMs may generate inaccurate operations or commands, often stemming from inherent inaccuracies and hallucinations.

Our exploratory study of three LLMs within penetration testing reveals their potential for executing end-to-end tasks. Nevertheless, challenges arise in maintaining long-term memory, devising a testing strategy beyond a depth-first approach, and generating accurate operations. In the following section, we elucidate how we address these challenges and outline our strategy for designing our LLM-powered penetration testing tool.

## 11.5 Methodology

### 11.5.1 Overview

In light of the challenges identified in the preceding section, we present our proposed solution, PentestGPT, which leverages the synergistic interplay of three LLM-powered modules. As illustrated in Figure 11.3, PentestGPT incorporates three core modules: the **Reasoning Module**, the **Generation Module**, and the **Parsing Module**. Each module reserves one LLM session with its conversation and context. The user interacts seamlessly with PentestGPT, where distinct modules process different types of messages.

This interaction culminates in a final decision, suggesting the subsequent step of the penetration testing process that the user should undertake. In the following sections, we elucidate our design reasoning and provide a detailed breakdown of the engineering processes behind PentestGPT.

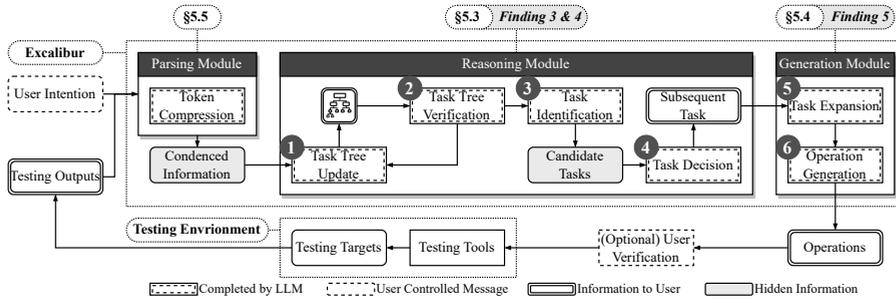


Figure 11.3: Overview of PentestGPT. Reproduced from [292].

### 11.5.2 Design Rationale

Our central design considerations emerged from the three challenges observed in the previous Exploratory Study (Section 11.4): The first challenge (*Finding 3*) pertains to the issue of penetration testing context loss due to memory retention. LLMs in their original form struggle to maintain such long-term memory due to token size limits. The second obstacle (*Finding 4*) arises from the LLM chatbots' tendency to emphasize recent conversation content. In penetration testing tasks, this focuses on optimizing the immediate task. This approach falls short in the complex, interconnected task environment of penetration testing. The third obstacle (*Finding 5*) is tied to the inaccurate results generation by LLMs. When tasked to produce specific operations for a step in penetration testing directly, the outputs are often imprecise, sometimes even leading to

PentestGPT has been engineered to address these challenges, rendering it more apt for penetration testing tasks. We drew inspiration from the methodologies employed by real-world penetration testing teams, where a director plans overarching procedures, subdividing them into subtasks for individual testers. Each tester independently performs their task, reporting results without an exhaustive understanding of the broader context. The director then determines the following steps, possibly redefining tasks, and

triggers the subsequent round of testing. Essentially, the director manages the overall strategy without becoming entrenched in the minutiae of the tests. This approach is mirrored in PentestGPT’s functionality, enhancing its efficiency and adaptability in conducting penetration tests. Our strategy divides penetration testing into two processes: identifying the next task and generating the concrete operation to complete the task. Each process is powered by one LLM session. In this setup, the LLM session responsible for task identification retains the complete context of the ongoing penetration testing status. At the same time, the generation of detailed operations and parsing of information is managed by other sessions. This division of responsibilities fosters effective task execution while preserving the overarching context.

To assist LLMs in effectively carrying out penetration testing tasks, we design a series of prompts that align with user inputs. We utilize the Chain-of-Thought (CoT) [328] methodology during this process. As CoT reveals, LLMs’ performance and reasoning capabilities can be significantly enhanced using the *input, chain-of-thought, output* prompting format. Here, the *chain-of-thought* represents a series of intermediate natural language reasoning steps leading to the outcome. We dissect the penetration testing tasks into micro-steps and design prompts with examples to guide LLMs through processing penetration testing information step-by-step, ultimately leading to the desired outcomes. The complete prompts are available at our anonymized open-source project [311].

### 11.5.3 Reasoning Module

The **Reasoning Module** plays a pivotal role in our system, analogous to a team lead overseeing the penetration testing task from a macro perspective. It obtains testing results or intentions from the user and prepares the testing strategy for the next step. This testing strategy is passed to the generation module for further planning.

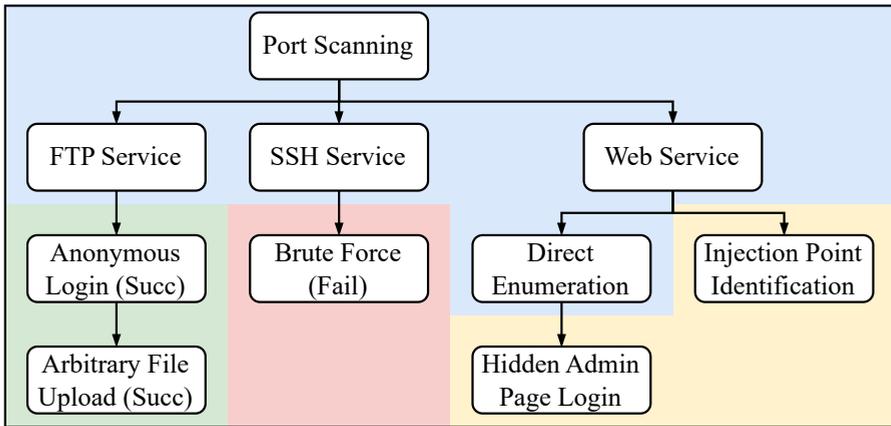
To effectively supervise the penetration testing process and provide precise guidance, it is crucial to translate the testing procedures and outcomes into a natural language format. Drawing inspiration from the concept of an attack tree [329], which is often used to outline penetration testing procedures, we introduce the notion of a *pentesting task tree* (PTT). This novel approach to testing status representation is rooted in the concept of an *attributed tree* [330]:

**Definition 1** (Attributed Tree). A *attributed tree* is an edge-labeled, attributed polytree  $G = (V, E, \lambda, \mu)$  where  $V$  is a set of nodes (or vertices),  $E$  is a set of directed edges,  $\lambda : E \rightarrow \Sigma$  is an edge labeling function assigning a label from the alphabet  $\Sigma$  to each edge and  $\mu : (V \cup E) \times K \rightarrow S$  is a function assigning key(from  $K$ )-value(from  $S$ ) pairs of properties to the edges and nodes.

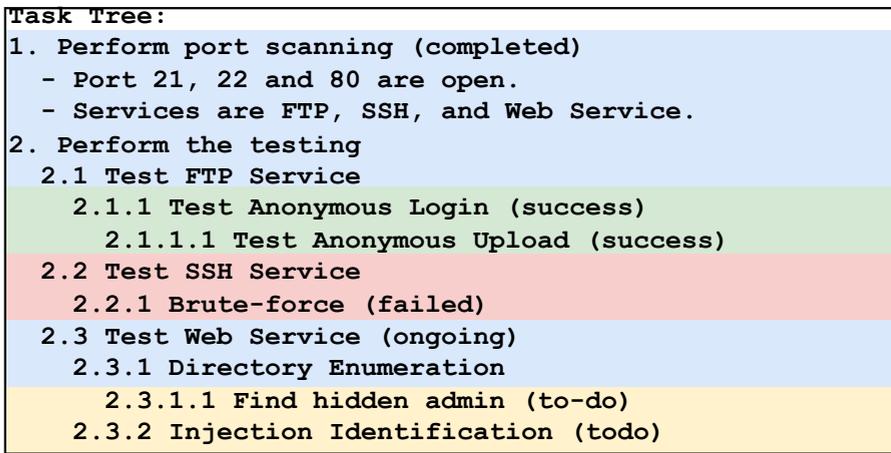
Given the definition of *attributed tree*, PTT is defined as follows:

**Definition 2** (Pentesting Task Tree). An *PTT*  $T$  is a pair  $(N, A)$ , wherein the correspond with: (1) a set of nodes  $N$  organized in a tree structure. Each node has a unique identifier, and there is a special node called the root that has no parent. Each node, other than the root, has exactly one parent and zero or more children. (2) the function  $A$  that assigns to each node  $n \in N$  a set of attributes  $A(n)$ . The function  $A(n)$  outputs a set of attribute pairs  $(a, v)$ , where  $a$  is the attribute name and  $v$  is the attribute value. The set of attributes can be different for each node.

As outlined in Figure 11.3, the Reasoning Module’s operation unfolds over four key steps operating over the PTT. ❶ Initially, the module absorbs the user’s intentions to construct an initial PTT in the form of natural language. This is achieved by carefully instructing the LLM with examples and definitions of PPT using meticulously crafted prompts. The LLM outputs are parsed to confirm that the tree structure is accurately formatted. Note that due to the nature of the tree structure, it can be represented in the natural language format through layered bullets, as illustrated in Figure 11.4. The Reasoning Module effectively overcomes the memory-loss issue by maintaining a task tree that encompasses the entire penetration testing process. ❷ After updating the tree information, a verification step is conducted on the newly updated PTT to ascertain its correctness. This process checks explicitly that only the leaf nodes of the PTT have been modified, aligning with the principle that atomic operations in the penetration testing process should only influence the status of the lowest-level sub-tasks. This step confirms the correctness of the reasoning process, safeguarding against any potential alterations to the overall tree structure due to hallucination by the LLM. If discrepancies arise, the information is reverted to the LLM for correction and regeneration. ❸ With the updated PTT, the Reasoning Module evaluates the current tree state and pinpoints viable sub-tasks that can serve as candidate steps for further testing. ❹



a) PTT Representatoin



b) PTT Representation in Natural Language

Figure 11.4: Pentesting Task Tree in a) visualized tree format, and b) natural language format encoded in LLM. Reproduced from [292].

Finally, the module evaluates the likelihood of these sub-tasks leading to successful penetration testing outcomes. It then recommends the top task as the output. The expected results of this task are subsequently forwarded to the Generation Module for an in-depth analysis. This is feasible, as

demonstrated in the exploratory study, since LLMs, particularly GPT-4, can identify potential vulnerabilities when provided with system status information. This procedural approach enables the Reasoning Module to address one of the inherent limitations of LLMs, precisely their tendency to concentrate solely on the most recent task. Note that in cases where the tester identifies that the correct task is incorrect or not completed in a preferred way, he could also manually revise the PTT through the interactive handle further discussed in Section 11.5.6.

We devise four sets of prompts to sequentially guide the Reasoning Module through the completion of each stage. To bolster the reproducibility of our results, we optimize these prompts further with a technique known as hint generation [331]. From our practical experience, we observe that LLMs are adept at interpreting the tree-structured information pertinent to penetration testing and can update it accurately in response to test outputs.

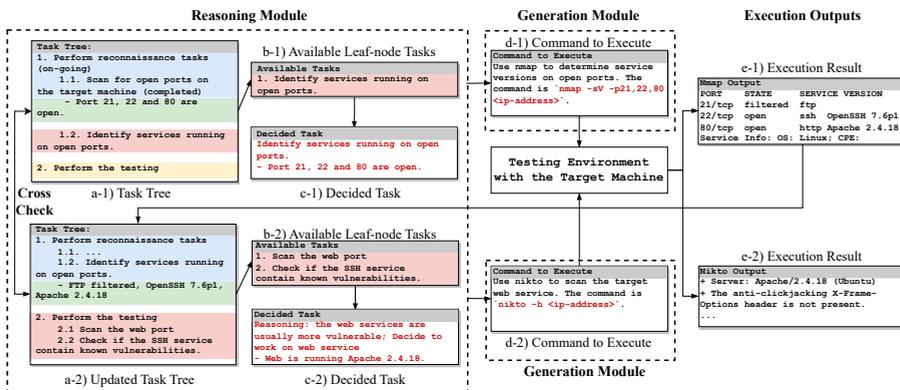


Figure 11.5: A demonstration of the task-tree update process on the testing target *HTB-Carrier*. Reproduced from [292].

## 11.5.4 Generation Module

The Generation Module translates specific sub-tasks from the Reasoning Module into concrete commands or instructions. Each time a new sub-task is received, a fresh session is initiated in the Generation Module. This strategy effectively isolates the context of the overarching penetration

task from the immediate task under execution, enabling the LLM to focus entirely on generating specific commands.

Instead of directly transforming the received sub-task into specific operations, our design employs the CoT strategy [328] to partition this process into two sequential steps. This design decision directly addresses the challenges associated with model inaccuracy and hallucination by enhancing the model's reasoning capability. In particular, ⑤ upon the receipt of a concise sub-task from the Reasoning Module, the Generation Module begins by expanding it into a sequence of detailed steps. Notably, the prompt associated with this sub-task requires the LLM to consider the possible tools and operations available within the testing environment. ⑥ Subsequently, the Generation Module transforms each of these expanded steps into precise terminal commands ready for execution or into detailed descriptions of specific Graphical User Interface (GUI) operations to be carried out. This stage-by-stage translation eliminates potential ambiguities, enabling testers to follow the instructions directly and seamlessly. Implementing this two-step process effectively precludes the LLM from generating operations that may not be feasible in real-world scenarios, thereby improving the success rate of the penetration testing procedure.

By acting as a bridge between the strategic insights provided by the Reasoning Module and the actionable steps required for conducting a penetration test, the Generation Module ensures that high-level plans are converted into precise and actionable steps. This transformation process significantly bolsters the overall efficiency of the penetration testing procedure.

**An Illustrative Example.** We utilize a real-world running example to illuminate how the Reasoning Module and the Generation Module collaboratively operate to complete penetration testing tasks. Figure 11.5 illustrates a single iteration of PentestGPT working on the HackTheBox machine Carrier [332], a medium-difficulty target. As depicted in a-1), the PTT, in natural language format, encodes the testing status, revealing the open ports (21, 22,80) on the target machine. The Reasoning Module is subsequently instructed to identify the available tasks. As highlighted in red, service scanning is the only available task on the leaf node of the PTT. This task is therefore chosen and forwarded to the Generation Module for command generation. The generated command is executed in the testing environment, and the execution result is conveyed to the Reasoning Module

to update the PTT. In a-2), the Reasoning Module integrates the previous scanning result into the PTT, cross-referencing it with the earlier PTT to update only the leaf nodes. It then looks for the available tasks to execute. In this case, two tasks emerge: scanning the web service on port 80 and checking the SSH service for known vulnerabilities. The LLM evaluates which task is more promising and chooses to investigate the web service, often seen as more vulnerable. This task is passed to the Generation Module. The Generation Module turns this general task into a detailed process, employing *nikto* [333], a commonly used web scanning script. The iterative process continues until the tester completes the penetration testing task.

### 11.5.5 Parsing Module

The **Parsing Module** operates as a supportive interface, enabling effective processing of the natural language information exchanged between the user and the other two core modules. Two needs can primarily justify the existence of this module. First, security testing tool outputs are typically verbose, laden with extraneous details, making it computationally expensive and unnecessarily redundant to feed these extended outputs directly into the LLMs. Second, users without specialized knowledge in the security domain may struggle to extract key insights from security testing outputs, presenting challenges in summarizing crucial testing information. Consequently, the Parsing Module is essential in streamlining and condensing this information.

In PentestGPT, the Parsing Module is devised to handle four distinct types of information: (1) *user intentions*, which are directives provided by the user to dictate the next course of action, (2) *security testing tool outputs*, which represent the raw outputs generated by an array of security testing tools, (3) *raw HTTP web information*, which encompasses all raw information derived from HTTP web interfaces, and (4) *source codes* extracted during the penetration testing process. Users must specify the category of the information they provide, and each category is paired with a set of carefully designed prompts. For source code analysis, we integrate the GPT-4 code interpreter [334] to execute the task.

### 11.5.6 Active Feedback

While LLMs can produce insightful outputs, their outcomes may sometimes require revisions. To facilitate this, we introduce an interactive handle in PentestGPT, known as active feedback, which allows the user to interact directly with the Reasoning Module. A vital feature of this process is that it does not alter the context within the Reasoning Module unless the user explicitly desires to update some information. The reasoning context, including the PTT, is stored as a fixed chunk of tokens. This chunk of tokens is provided to a new LLM session during an active feedback interaction, and users can pose questions regarding them. This ensures that the original session remains unaffected, and users can always query the reasoning context without making unnecessary changes. If the user believes it necessary to update the PTT, they can explicitly instruct the model to update the reasoning context history accordingly. This provides a robust and flexible framework for the user to participate in the decision-making process actively.

### 11.5.7 Discussion

We explore various design alternatives for PentestGPT to tackle the challenges identified in Exploratory Study. We have experimented with different designs, and here we discuss some key decisions.

**Addressing Context Loss with Token Size:** a straightforward solution to alleviate context loss is the employment of LLM models with an extended token size. For instance, GPT-4 provides versions with 8k and 32k token size limits. This approach, however, confronts two substantial challenges. First, even a 32k token size might be inadequate for penetration testing scenarios, as the output of a single testing tool like *dirbuster* [335] may comprise thousands of tokens. Consequently, GPT-4 with a 32k limit cannot retain the entire testing context. Second, even when the entire conversation history fits within the 32k token boundary, the API may still skew towards recent content, focusing on local tasks and overlooking broader context. These issues guided us in formulating the design for the Reasoning Module and the Parsing Module.

**Vector Database to Improve Context Length:** Another technique to enhance the context length of LLMs involves a vector database [336, 337]. By transmuting data into vector embeddings, LLMs can efficiently store and

retrieve information, practically creating long-term memory. Theoretically, penetration testing tool outputs could be archived in the vector database. In practice, though, we observe that many results closely resemble and vary in only nuanced ways. This similarity often leads to confused information retrieval. Solely relying on a vector database fails to overcome context loss in penetration testing tasks. Integrating the vector database into the design of PentestGPT is an avenue for future research.

**Precision in Information Extraction:** Precise information extraction is crucial for conserving token usage and avoiding verbosity in LLMs. Rule-based methods are commonly employed to extract diverse information. However, rule-based techniques are engineeringly expensive given natural language’s inherent complexity and the variety of information types in penetration testing. We devise the Parsing Module to manage several general input information types, a strategy found to be both feasible and efficient.

**Limitations of LLMs:** LLMs are not an all-encompassing solution. Present LLMs exhibit flaws, including hallucination [338] and outdated knowledge. Our mitigation efforts, such as implementing task tree verification to ward off hallucination, might not completely prevent the Reasoning Module from producing erroneous outcomes. Thus, a human-in-the-loop strategy becomes vital, facilitating the input of necessary expertise and guidance to steer LLMs effectively. Moreover, it must be highlighted that reproducibility of results is an issue here, as LLMs are in continuous evolution. In our case, given the novelty of the technology and lack of availability at the time of starting this work, we did not host the LLM deployments ourselves, but instead used third party providers. This setup is sub-optimal from a scientific experiment point of view. For future work, we foresee building our own LLMs by performing post-training phases on top of existing open source models and *freezing* them after a certain stage of learning. This will lead to results being reproducible given the right input context, and fully reproducible on statistical average. This topic is further discussed in Section 11.7.

## 11.6 Evaluation

In this section, we assess the performance of PentestGPT, focusing on the following four research questions:

**RQ3 (Performance):** How does the performance of PentestGPT compare with that of native LLM models and human experts?

**RQ4 (Strategy):** Does PentestGPT employ different problem-solving strategies compared to those utilized by LLMs or human experts?

**RQ5 (Ablation):** How does each module within PentestGPT contribute to the overall penetration testing performance?

**RQ6 (Practicality):** Is PentestGPT practical and effective in real-world penetration testing tasks?

### 11.6.1 Evaluation Settings

We implement PentestGPT with 1,700 lines of Python3 code and 740 prompts, available at our anonymized project website [311]. We evaluate its performance over the benchmark constructed in Section 11.3. In this evaluation, we integrate PentestGPT with GPT-3.5 and GPT-4 to form two working versions: PentestGPT-GPT-3.5 and PentestGPT-GPT-4. Due to the lack of API access, we do not select other LLM models, such as Bard. In line with our previous experiments, we use the same experiment environment setting and instruct PentestGPT to only use the non-automated penetration testing tools.

### 11.6.2 Performance Evaluation (RQ3)

The overall task completion status of PentestGPT-GPT-3.5, PentestGPT-GPT-4, and the naive usage of LLMs is illustrated in Figure 11.6a. As the Figure shows, our solutions powered by LLMs demonstrate superior penetration testing capabilities compared to the naive application of LLMs. Specifically, PentestGPT-GPT-4 surpasses the other three solutions, successfully solving 6 out of 7 easy difficulty targets and 2 out of 4 medium difficulty targets. This performance indicates that PentestGPT-GPT-4 can handle penetration testing targets ranging from easy to medium difficulty levels. Meanwhile, PentestGPT-GPT-3.5 manages to solve only two challenges of easy difficulty, a discrepancy that can be attributed to GPT-3.5 lacking the knowledge related to penetration testing found in GPT-4.

The sub-task completion status of PentestGPT-GPT-3.5, PentestGPT-GPT-4, and the naive usage of LLM is shown in Figure 11.6b. As the Figure illustrates, both PentestGPT-GPT-3.5 and PentestGPT-GPT-4 perform better than the standard utilization of LLMs. It is noteworthy that PentestGPT-GPT-

4 not only solves one more medium difficulty target compared to naive GPT-4 but also accomplishes 111% more sub-tasks (57 vs. 27). This highlights that our design effectively addresses context loss challenges and leads to more promising testing results. Nevertheless, all the solutions struggle with hard difficulty testing targets. As elaborated in Section 11.4, hard difficulty targets typically demand a deep understanding from the penetration tester. To reach testing objectives, they may require modifications to existing penetration testing tools or scripts. Our design does not expand the LLMs' knowledge of vulnerabilities, so it does not notably enhance performance on these more complex targets.

### 11.6.3 Strategy Evaluation (RQ4)

We then investigate the problem-solving strategies employed by PentestGPT, contrasting them with those of LLMs and human experts. By manually analyzing the penetration testing process of PentestGPT, we synthesize its underlying approaches to problem-solving. We surprisingly find that PentestGPT decomposes the penetration testing task in a manner akin to human experts, successfully achieving the overall goal. Instead of focusing solely on the most recently discovered task, PentestGPT can pinpoint potential sub-tasks likely to lead to successful outcomes.

Figure 11.7 provides an illustrative example, demonstrating the strategic differences between GPT-4 and PentestGPT while handling the VulnHub machine, *Hackable II* [339]. This target comprises two vulnerable services: an FTP service allowing arbitrary file uploads and a web service enabling file viewing through FTP. A successful exploit necessitates exploiting both services by uploading a malicious PHP shell via the FTP service and triggering it through the web service. As depicted in the figure, GPT-4 begins by enumerating the FTP service and successfully identifies the file upload vulnerability (❶-❷). However, it fails to correlate this with the web service, resulting in an incomplete exploit in the following steps. Conversely, PentestGPT follows a more holistic approach, toggling between enumerating the FTP service and browsing the web service. In particular, PentestGPT firstly ❶ enumerates the FTP service and ❷ web service to understand the general situation. It then ❸ prioritizes the FTP service, and ❹ eventually discovers the file upload vulnerability. More importantly, in this process, PentestGPT identifies that files available on FTP are the same as those on the web service. By connecting these findings, PentestGPT

guides the tester to ⑤ perform a shell upload, ⑥ leading to a successful reverse shell. This strategy aligns with the walkthrough solution and highlights PentestGPT's comprehensive understanding of the penetration testing process and its ability to make effective decisions on the optimal sub-task to pursue next. This reveals PentestGPT's strategic thinking and ability to integrate different aspects of the testing process.

Our second observation is that although PentestGPT behaves more similarly to human experts, it still exhibits some strategies that humans will not apply. For instance, PentestGPT still prioritizes brute-force attacks before vulnerability scanning. This is obvious in cases where PentestGPT always tries to brute-force the SSH service on target machines.

We then analyze the failed penetration testing cases to understand the limitations of PentestGPT. Beyond the absence of some advanced penetration testing techniques, two primary issues emerge. First, PentestGPT struggles to interpret images. LLMs are limited to text comprehension, so they cannot accurately process images. This issue might be addressed by developing large multimodal models to understand text and visual data. Second, it cannot grasp certain social engineering tricks and subtle cues. For instance, real-world penetration testers often create brute-force wordlists using information gathered from the target service. Though PentestGPT can retrieve a list of names from a web service, it fails to instruct the use of tools to create a wordlist from those names. These limitations underline the necessity for improvement in areas where human insight and intricate reasoning are still more proficient than automated solutions.

### 11.6.4 Ablation Study (RQ5)

We perform an ablation study on how the three modules: Reasoning Module, Generation Module, and Parsing Module, contribute to the performance of PentestGPT. We implement three variants:

1. PentestGPT-no-Parsing: the Parsing Module is deactivated, causing all data to be directly fed into the system.
2. PentestGPT-no-Generation: the Generation Module is deactivated, leading to the completion of task generation within the Reasoning Module itself. The prompts for task generation remain consistent.

3. PentestGPT-no-Reasoning: the Reasoning Module is disabled. Instead of PTT, this variant adopts the same methodology utilized with LLMs for penetration testing, as delineated in the Exploratory Study.

All the variants are integrated with GPT-4 API for testing.

The results of the three variants tested on our penetration testing benchmarks are depicted in Figure 11.8. In general, PentestGPT demonstrates superiority over the three ablation baselines regarding overall target and sub-task completion. Our key findings are as follows: (1) In the absence of the Parsing Module, PentestGPT-no-Parsing attains marginally lower performance in overall task and sub-task completion relative to the full configuration. While parsing information is advantageous in penetration testing, the 32k token size limit often suffices for various outputs. Given the Reasoning Module’s inherent design to maintain the entire testing context, the lack of the Parsing Module does not substantially impair the tool’s performance. (2) PentestGPT-no-Reasoning fares the worst, completing only 53.6% of the sub-tasks achieved by the full solution, an outcome even inferior to the naive application of GPT-4 in testing. We attribute this to the Generation Module adding supplementary sub-tasks to the LLM context. Since the prompts are not tailored for scenarios without the Reasoning Module, the resulting outputs are irrelevant for the naive LLM without the Generation Module. Furthermore, the extended generation output obscures the original context, hindering the LLM’s ability to concentrate on the task, thus failing the test. (3) PentestGPT-no-Generation realizes performance slightly above that of GPT-4 employed naively. This occurs because, without the Generation Module, the testing procedure closely resembles the usage of LLMs. Notably, the Generation Module is principally intended to guide the tester in executing precise penetration testing operations. Without this module, the tester may depend on supplementary information to operate the tools or scripts essential for completing the test.

### 11.6.5 Practicality Study (RQ6)

We demonstrate that PentestGPT exhibits practicality for real-world penetration testing beyond the crafted benchmark. For this purpose, we engage PentestGPT in the HackTheBox active machine challenges, a series of penetration testing objectives open to global testers. Each challenge consists of two components: a user flag, retrievable upon initial

## 11. Guiding Attacks with Machine Learning

---

Machine	Difficulty	Completion	Completed Users	Cost (USD)
Sau	Easy	✓	4798	15.2
Pilgrimage	Easy	✓	5474	12.6
Topology	Easy	✗	4500	8.3
PC	Easy	✓	6061	16.1
MonitorsTwo	Easy	✓	8684	9.2
Authority	Medium	✗	1209	11.5
Sandworm	Medium	✗	2106	10.2
Jupiter	Medium	✗	1494	6.6
Agile	Medium	✓	4395	22.5
OnlyForYou	Medium	✗	2296	19.3
Total	-	6	-	131.5

Table 11.5: PentestGPT performance over the active HackTheBox Challenges.

user access, and a root flag, obtainable after gaining root access. Our evaluation encompasses five targets of easy difficulty and five of medium difficulty. During this exercise, PentestGPT, utilizing GPT-4’s 32k token API, conducts up to five tests on each target. Success is defined solely by the capture of the root flag. Table 11.5 details the performance of PentestGPT in these challenges<sup>4</sup>. Ultimately, PentestGPT completes three easy and five medium challenges. The total expenditure for this exercise amounts to 131.5 USD, averaging 21.92 USD per target. This cost is markedly lower than employing human penetration testers and falls within an acceptable range. Our evaluation, therefore, underscores PentestGPT’s capability to yield viable penetration testing results in real-world settings at an efficient cost, thereby highlighting its potential as a practical tool in the cybersecurity domain.

### 11.7 Discussion

We recognize that the penetration testing walkthrough might have been part of the training material for the tested LLMs, potentially biasing the results. To mitigate this, we take two measures. First, we manually verify that the LLM does not have prior knowledge of the target machine. We do this by prompting the LLMs if the tested machine is within their knowledge

---

<sup>4</sup>**Completed Users** denotes the number of users globally who have completed the target as of the manuscript submission time. Note that HackTheBox boasts over 670,000 active users.

base. Second, we include penetration testing target machines released after 2021 in our benchmark, which falls outside the training data of OpenAI models. The practicality study on the most recent HackTheBox challenges also demonstrates that PentestGPT can solve challenges without prior knowledge of the target.

The rapidly evolving nature of LLMs and inconsistencies in available APIs could invalidate PentestGPT's designed prompts. We strive to make prompts general and suitable for various LLMs. However, due to their hacking nature, some LLMs resist generating specific penetration testing content, such as concrete reverse shell scripts. Our prompts include jailbreak techniques [340] to guide the LLM to generate penetration-testing-related information. How to generate reproducible outcomes is an important direction we are working towards.

We identify hallucination in Large Language Models [338] as a significant challenge where the model's outputs diverge from its training data. This affects the reliability of our automatic penetration testing tool. We are actively exploring various techniques [341] to reduce hallucination and enhance our tool's performance. As an ongoing work, we believe such an attempt will lead to a more robust and effective automatic penetration testing tool.

## 11.8 Conclusion

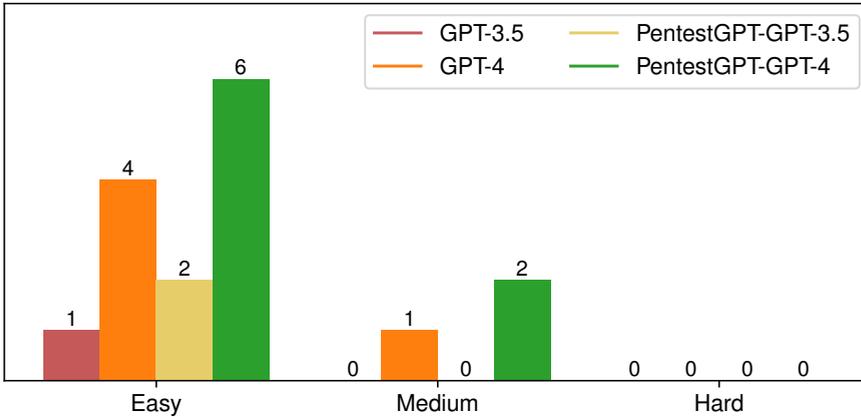
This chapter paved the way towards integrating Large Language Models (LLMs) with the operational processes of cybersecurity, particularly in guiding cybersecurity software agents to perform autonomously and more specifically, automating and refining penetration testing strategies. The implementation of PentestGPT, our innovative prototype, showcases the potential of LLMs to emulate human-like decision-making in cyber-attacks. While PentestGPT demonstrates adeptness in fundamental cybersecurity tasks, it also reveals the limitations of current models, particularly in maintaining context and focusing attention over prolonged interactions.

PentestGPT's architecture, inspired by the dynamics of professional penetration testing teams, divides complex tasks into manageable segments, enabling a collaborative approach to threat analysis and response. This modular structure of Reasoning, Generation, and Parsing replicates the cognitive synergy found in human teams, ensuring a comprehensive and nuanced approach to cybersecurity challenges.

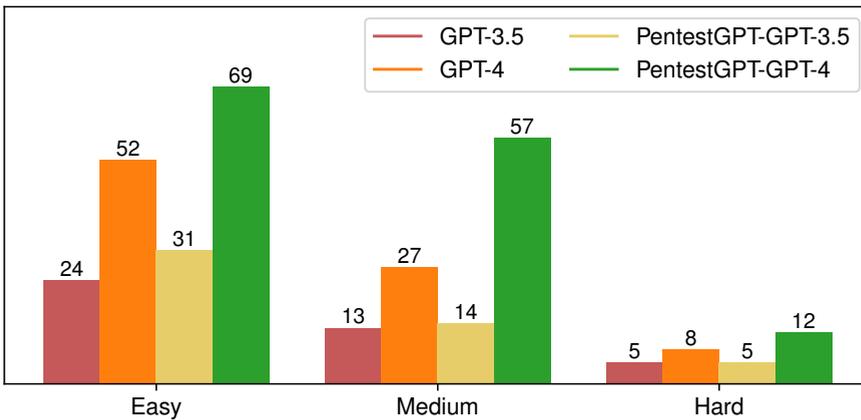
## 11. Guiding Attacks with Machine Learning

---

The insights gleaned from our evaluation of PentestGPT suggest that while LLMs offer significant advantages, they are not a panacea. There remains a clear need for human intuition and expertise, particularly in complex and novel scenarios that exceed the training parameters of existing models. As we look to the future, the integration of LLMs in cybersecurity presents an exciting frontier with the potential to revolutionize the field. The continuous development of PentestGPT and similar tools promises to enhance our defensive and offensive capabilities, making our digital infrastructures more resilient against the evolving threats they face. Our work lays a foundation for future research, where the symbiosis of human and machine intelligence will lead to unprecedented advancements in maintaining cybersecurity.



(a) Overall completion status.



(b) Subtask completion status.

Figure 11.6: The performance of GPT-3.5, GPT-4, PentestGPT-GPT-3.5, and PentestGPT-GPT-4 on overall target completion and sub-task completion. Reproduced from [292].

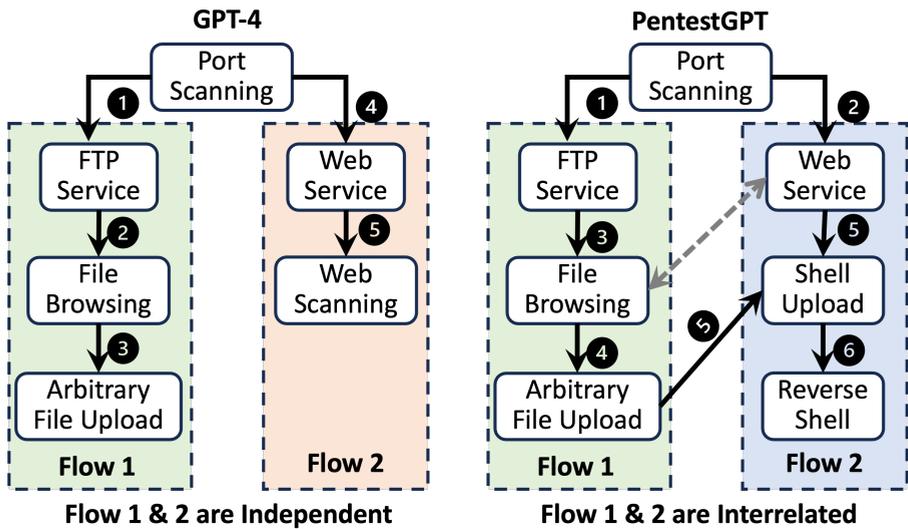
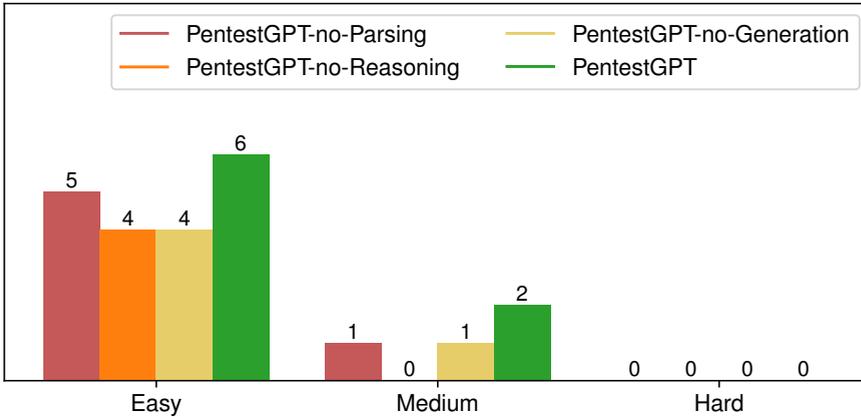
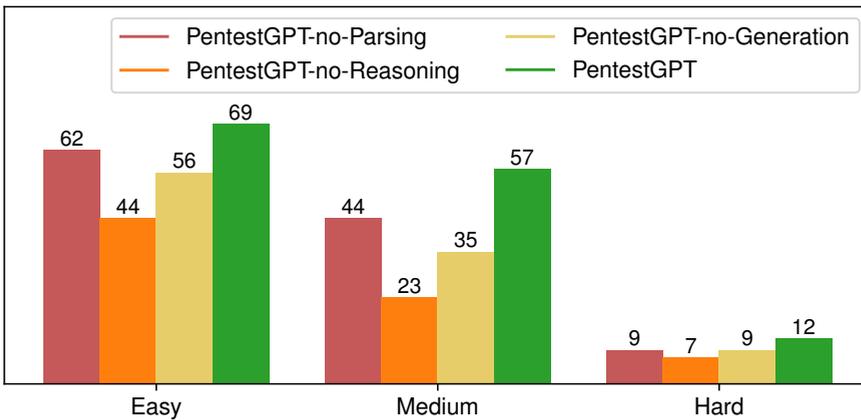


Figure 11.7: Penetration testing strategy comparison between GPT-4 and PentestGPT on *VulnHub-Hackable II*. Reproduced from [292].



(a) Overall completion status



(b) Sub-task completion status

Figure 11.8: The performance of PentestGPT, PentestGPT-No-Annotation, PentestGPT-Operation-Only, and PentestGPT-Parameter-Only on both normalized average code coverage ( $\mu LOC$ ) and bug detection. Reproduced from [292].



## Chapter 12

# Conclusion and future work

This thesis, titled "Offensive Robot Cybersecurity," presents a multifaceted exploration of safeguarding robots by adopting a preemptive hacking stance, using Game Theory and Machine Learning as primary tools for automation. The journey began with a motivation (**Part I**), which provided a general introduction to the field and a description of the thesis structure and framework (Chapter 1). This is followed by some general background (Chapter 2) which provided more context to the intersection of cybersecurity, robotics and artificial intelligence involved in this work.

The content then continues with an in-depth ethical investigation (**Part II**), setting the landscape (Chapter 3) for responsible and ethically-aligned cybersecurity practices in robotics.

The development of fundamental blueprints (**Part III**) introduced a suite of core know-how, tools and frameworks, fostering a secure robotics research environment and empowering both defensive and offensive research. This includes DevSecOps for Robotics and ROS 2 security mechanisms (Chapter 4), benchmarking robots and their security (Chapter 5) and reducing the timing overhead and computational bottlenecks introduced by security layers in robotics through hardware acceleration (Chapter 6).

The heart of the thesis lies in the meticulous cyber-attacking of robots (**Part IV**), a necessary evil that unveiled a plethora of robot vulnerabilities. This research was conducted attacking robot software (Chapter 7), hardware (Chapter 8) and industry deployments (Chapter 9), providing invaluable insights into the current state of robot cybersecurity.

The finale (**Part V**) of the thesis ventured into the realm of autonomous offensive cybersecurity strategies for robots, harnessing the analytical prowess of Game Theory (Chapter 10) to determine the most vulnerable attack targets and vectors, followed by the use of modern Machine Learning approaches to guide the offensive behavior. In particular, Large Language Models helped bring up adaptive intelligence to our *cybersecurity cognitive engines* (Chapter 11).

## 12. Conclusion and future work

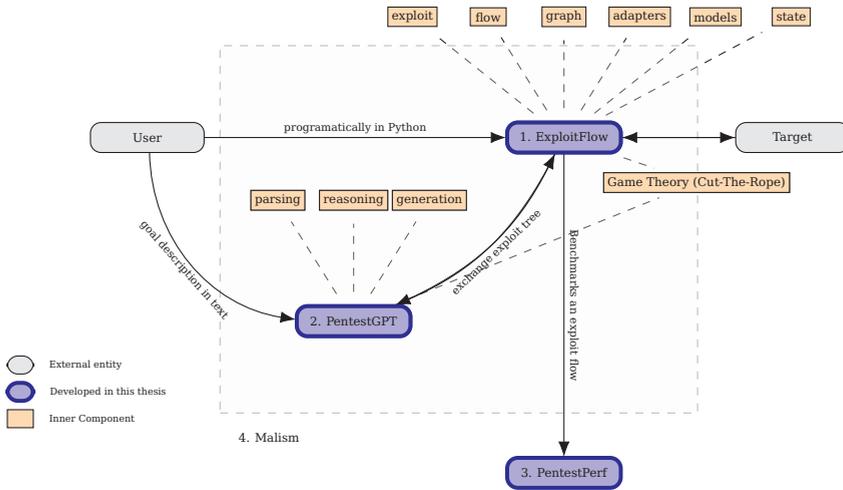


Figure 12.1: Our architecture and framework to develop fully automated offensive robot cybersecurity agents, which name *cybersecurity cognitive engines* in Chapter 10: Malism.

The resulting framework and architecture for our *cybersecurity cognitive engines* (introduced in Chapter 10) is again presented above in Figure 12.1 and captures the results obtained in this thesis. In relation to the content of this thesis, Parts I and II helped understand better the User and Target entities. Part III helped developed tools such as *ExploitFlow*, which sits at the core of our architecture providing capabilities for composing exploitation flows to interact with the robotic security environment and achieve desired goals. Part IV helped populate tools like *ExploitFlow* with actions that corresponded with both exploitation and mitigation of robot security flaws. Finally, Part V completes and connects the architecture altogether with three final contributions: a game theoretic extension of *ExploitFlow* (through implementing Cut-The-Rope) that allows to reason about attack trees and identify the best attack vectors to achieve desired goals, b) a security reasoning engine (*PentestGPT*) that digests such annotated attack trees, the attack vectors and the ultimate goal and guides the process and c) a comprehensive penetration testing benchmark (*PentestPerf*) that helps compare various implementations for the penetration testing task.

---

In conclusion, this thesis successfully demonstrates the critical role of offensive measures in enhancing robot cybersecurity, charting a path towards a future where robots are not only resilient to cyber threats but also equipped to autonomously defend themselves by hacking-themselves-first with Game Theory and Machine Learning. In a soon-to-come autonomous ever-evolving digital battleground, *Malism* aims to provide a reference guideline for future implementations. In summary, the work presented in this thesis lays a solid foundation for the future of offensive cybersecurity in robotics, paving the way for innovative solutions that can autonomously protect and secure robotic systems in an increasingly interconnected world. The advancements achieved through this research not only contribute to the academic field by demonstrating how robots can be made more secure with an offensive stance, but also have far-reaching implications for the safety and security of next-generation robotic applications.

Future work will focus on implementing *Malism* in a scalable form factor, demonstrating fully autonomous offensive and defensive cybersecurity behaviors, and extending its concepts to industrial setups and OT-machinery beyond robotics. Achieving embedded form factors for *Malism*-like cybersecurity cognitive engines is critical for real-world applicability, necessitating the integration of efficient Machine Learning (ML) and Game Theory (GT) algorithms within limited computational resources. Leveraging hardware accelerators like FPGAs can empower these deployments, ensuring real-time threat detection and response without compromising primary robotic functions. Embedded systems will enable localized, on-device processing, reducing dependency on vulnerable external cloud services and enhancing security response times. Moreover, distributed security frameworks facilitated by embedded *Malism*-like systems will enhance resilience and coordination in networked environments. This approach is crucial for dynamic, time-sensitive applications in autonomous vehicles, smart manufacturing, and critical infrastructure, where robust, real-time security solutions are essential.

In reflecting on this research journey, we find ourselves at the exhilarating intersection of robotics and cybersecurity, where offensive tactics become the ultimate defense. This thesis has danced through the realms of ethical hacking, intricate game theory strategies, and the boundless potential of machine learning, culminating in the creation of

## 12. Conclusion and future work

---

autonomous guardians named *cybersecurity cognitive engines* for our robotic systems. As we look to the future, *Malism* stands as a reference concept that deserves further development and time. Such a concept may eventually lead to a world where robots not only outsmart cyber adversaries but evolve with every challenge they face.

Part VI

# **Appendices**



# Appendix A

## Research Results

### A.1 Thesis framework

The topics of interest for the purpose of this thesis are presented as building blocks. These blocks represent a framework for the development of this dissertation. Results are categorized accordingly, with each research product aligning to one of these *Blocks* as hinted in the tables below. From bottom to top: (0) Ethics and law, (1) Offensive security blueprints for robots and their components, (2) Cyber-attacking robots and (3) Cyber-attacking robots with Machine Learning and Game Theory. See Figure 1.2.

### A.2 Past work

Tables A.1 and A.2 present a summary of the work performed prior to engaging formally with the PhD. Several articles were produced focusing mostly in blocks 0 (Ethics and law) and 1 (Blueprints). Since building a solid base was instrumental for the PhD research, these grounding publications capture the robotics know-how acquired after multiple years in industry and represent the baseline from wherein the research started.

## A. Research Results

Title	Summary	Year	Block
Extending the OpenAI gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo [342]	This paper introduces an extension of the OpenAI Gym for robotics, integrating ROS and Gazebo simulator. It elaborates on the software architecture and benchmarks two Reinforcement Learning techniques: Q-Learning and Sarsa. The work establishes a system for comparing various techniques and algorithms under consistent virtual conditions in robotics.	2016	1
The Hardware Robot Operating System (H-ROS); an infrastructure to create interoperable robot components [228]	This article presents the Hardware Robot Operating System (H-ROS), a hybrid hardware-software infrastructure for creating adaptable, interchangeable robot components. H-ROS enables dynamic modification of a robot's internal representation, allowing selective activation and exchange of hardware parts, thus simplifying robot assembly. The study validates the concept's feasibility and discusses future enhancements for the H-ROS platform."	2017	1
Time-sensitive networking for robotics [190]	This paper posits that Time-Sensitive Networking (TSN) will emerge as the standard for real-time robotic communications, reviewing relevant communication standards and highlighting issues with traditional Ethernet networks. It evaluates TSN's deterministic communication features through experimental tests in a robotic scenario. The findings suggest a gradual shift from existing real-time solutions to TSN, paving the way for a unified, interoperable landscape in robotics.	2018	1
Real-time Linux communications: an evaluation of the Linux communication stack for real-time robotic applications [191]	This paper examines the Linux communication stack's suitability for real-time robotic applications, focusing on UDP-based communications in multi-core embedded devices. The study confirms that with proper configuration, the Linux kernel significantly improves UDP communication determinism. It also reveals that concurrent traffic affects latency bounds, suggesting isolation of real-time applications and corresponding interrupts on a separate CPU as a solution	2018	1
Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications [192]	This study evaluates ROS 2.0 for real-time robotic applications, focusing on inter-component communication on Linux systems. It benchmarks worst-case latencies and missed deadlines to assess ROS 2.0's performance in real-time contexts. The research experimentally shows the impact of computational and network congestion on communication latencies and proposes a setup that achieves bounded traffic under specific conditions.	2018	1
Time synchronization in modular collaborative robots [193]	The study emphasizes synchronization in modular robot systems, addressing a key challenge in the next generation of collaborative robots. It delves into synchronization issues, demonstrating that with an optimally configured system, remarkable synchronization precision is attainable. Results include distributed sub-microsecond clock synchronization among modules, ROS 2.0 message timestamping accuracy below 100 microseconds, and stable millisecond-level end-to-end communication latencies, even under heavy network loads of up to 90% capacity.	2018	1

Table A.1: Past work group 1

Title	Summary	Year	Block
Introducing the Robot Security Framework (RSF), a standardized methodology to perform security assessments in robotics [89]	A methodology to perform systematic security assessments in robots proposing a checklist-like approach that reviews most relevant aspects in a robot	2018	1
Robot hazards: from safety to security [2]	Discussion of the current status of insecurity in robotics and the relationship between safety and security, ignored by most vendors	2018	0
Towards an open standard for assessing the severity of robot security vulnerabilities, the Robot Vulnerability Scoring System (RVSS) [90]	Introduction of a new assessment scoring mechanisms for the severity of vulnerabilities in robotics that builds upon previous work and specializes it for robotics	2018	1
Robotics CTF (RCTF), a playground for robot hacking [91]	Docker-based CTF environment for robotics (RCTF) which bridges the gap between robotics and cybersecurity. The RCTF enables reproduction of virtual robotics scenarios and adjustment of network setups for realistic security challenges.	2018	1
Aztarna, a footprinting tool for robots [85]	Aztarna, a tool focused on robot reconnaissance and footprinting, addresses the security overlook in the Industry 4.0 era, where robots are increasingly network-connected. It enables identification of various robots, aiding in assessing the widespread lack of security practices in the robotics industry.	2018	1
Volatile memory forensics for the Robot Operating System [84]	General overview of forensic techniques in robotics and discussion of a robotics-specific Volatility plugin named linux_rosnode, packaged within the ros_volatility project and aimed to extract evidence from robot's volatile memory.	2018	1
Introducing the robot vulnerability database (RVD) [15]	The Robot Vulnerability Database (RVD) addresses the emerging importance of cybersecurity in robotics, a field where cyber attacks can lead to severe safety consequences. RVD serves as a comprehensive platform for responsible disclosure of vulnerabilities in robots, surpassing existing databases in detail and relevance. This paper outlines RVD's design, disclosure policy, and invites contributions to mitigate zero-day vulnerabilities in robotics, enhancing overall security	2019	1
ROS 2 Security Workshop - ROSCon 2019: Demonstrating vulnerabilities and ways to protect your robots [343]	This workshop demonstrated different real-world attacks performed on robots while teaching techniques on how to secure them against those attacks. This included teaching on how to design threat models for robotics, hardening the ROS ecosystem through SROS2 as well as using several penetration testing tools to verify that the robot is no longer vulnerable to those attacks.	2019	2
ROS 2 Real-Time Workshop - ROSCon 2019: Doing Real-Time with ROS 2: Capabilities and Challenges [344]	This workshop introduced participants to the on-going work to make ROS 2 usable as a foundation for soft, firm and hard real-time robot systems. Participants will learn what the current capabilities are of ROS 2 for real-time robotics, what still remains to be done, and how they can contribute to making ROS 2 usable in these applications.	2019	1

Table A.2: Past work group 2

### A.3 Peer-reviewed articles

Tables A.3 and A.4 show the peer-reviewed articles published within the PhD period:

Title	Summary	Year	Block	Type/Publisher
Industrial robot ransomware: Akerbeltz [88] (paper)	Introduce Akerbeltz, the first known instance of industrial robot ransomware targeting a well known collaborative robot.	2020	2	2020 Fourth IEEE International Conference on Robotic Computing (IRC)
Cybersecurity in Robotics: Challenges, Quantitative Modeling and Practice [345] (paper)	This book emphasizes the critical need for integrating security in robotics from the design phase, addressing the common neglect of security in the rapid development of robotic systems. It advocates for quantitative security approaches, including tailored vulnerability scoring and game theory, to enhance protection against various attacks. Targeting both theorists and practitioners, the book offers strategies to manage the complex nature of robotic systems effectively.	2021	1	Foundations and Trends® in Robotics: Vol. 9: No. 1, pp 1-129
Hacking planned obsolescence in robotics, towards security-oriented robot teardown [104] (paper)	This thought-provoking article introduces robot teardown as a method to explore robot hardware architectures and enhance security research. It highlights the increasing need for component updates in damaged or compromised robots and critiques manufacturers' use of planned obsolescence to hinder repairs. The approach is effective in exposing security vulnerabilities and evidencing manufacturers' strategies against competition.	2021	0	Conference on Networked Systems 2021 (NetSys 2021)
Robot teardown, stripping industrial robots for good [101] (paper)	This paper examines the crucial hardware/software nexus in robotics, emphasizing teardown as essential for cybersecurity research. It analyzes contemporary industrial robot hardware, critiquing manufacturers' practices of preprogrammed obsolescence and the consequent generation of robotics waste. The study highlights the manufacturers' restrictive networks that limit repair capabilities and stifle competition, underscoring the ongoing need for component updates in response to damage or security issues.	2022	2	Journal of Cyber Forensics and Advanced Threat Investigations (CFATI)
Robot cybersecurity, a review [83] (paper)	This article highlights the inadequate cybersecurity in robotics, attributed to underdeveloped defensive mechanisms, the complexity and cost of securing robots, and manufacturers' delayed responses to vulnerabilities. Drawing on literature, forum surveys, and recent research, it underscores the challenges in robotic security. The author, leveraging extensive experience and empirical data from security assessments, advocates for a proactive offensive strategy as an effective solution to strengthen robot cybersecurity.	2022	0	Journal of Cyber Forensics and Advanced Threat Investigations (CFATI)
SROS2: Usable Cyber Security Tools for ROS 2 [102] (paper)	SROS2 is presented as a user-friendly suite of tools and libraries to secure ROS 2 computational graphs, aligning with the growing use of ROS 2 in the robotics industry. The work emphasizes a systematic, DevSecOps-aligned approach to security, illustrated through a case study with TurtleBot3. The analysis of SROS2's capabilities underscores the essential role of usability in the effective implementation of security in robotics.	2022	1	2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)

Table A.3: Peer-reviewed articles group 1.

Title	Summary	Year	Block	Type/Publisher
RobotCore: An Open Architecture for Hardware Acceleration in ROS 2 [139] (paper)	RobotCore enhances robotics by integrating hardware acceleration into ROS 2, supporting diverse platforms and accelerators like FPGAs and GPUs. This adaptable architecture, complemented with a new firmware layer, facilitates efficient deployment across various setups. It utilizes the Linux Tracing Toolkit for real-time tracing and benchmarking, highlighting its effectiveness in a case study where RobotCore accelerates a ROS 2 computational graph on an FPGA, achieving a significant 24.42% speedup over CPU-based systems.	2022	1	2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
FogROS 2: An adaptive and extensible platform for cloud and fog robotics using ROS 2 [103] (paper)	FogROS2 is an advanced open-source platform designed for enhancing robotic computing by integrating with cloud services like AWS, GCP, and Azure, ensuring compatibility with ROS 2. With a focus on security and efficiency, it offers substantial improvements over FogROS1, including lower latency, faster startup, and better usability. Key enhancements include a 50% reduction in SLAM latency and a significant boost in planning speeds, alongside improved network efficiency. Available on the ROS 2 repository, FogROS2 addresses the computing limitations of robots securely and effectively.	2023	1	2023 Proceedings IEEE International Conference on Robotics and Automation (ICRA)
Threat modeling for robotic-based production plants [346] (paper)	This study highlights the growing cybersecurity challenges accompanying the integration of robotics and automation in production plants. It presents a comprehensive threat model for a production facility with diverse components like PLCs, machine tools, sensors, actuators, and robots. Recognizing the heterogeneity of these components and their protocols, the paper outlines potential threats to the factory's security. It also suggests a series of changes and mitigations to enhance the cybersecurity and resilience of these production systems.	2022	2	2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)
Game-theoretic APT defense: An experimental study on robotics [283] (paper)	This paper presents a novel game-theoretic approach to counter Advanced Persistent Threats (APTs) in cybersecurity. It expands the Cut-The-Rope model to simulate more realistic attack scenarios and employs this framework in an experimental APT defense game using attack graphs. This allows for optimized defense strategies without assuming a clean system or relying on adversarial activity indicators. The model effectively incorporates vulnerability data like CVEs but can also function without it. Tests on real-world attack graphs in robotics show that this optimized defense significantly outperforms basic heuristics, demonstrating its practical applicability and effectiveness in complex cybersecurity environments.	2023	3	Computers & Security (2023)
RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance [133] (paper)	RobotPerf is a vendor-neutral, open-source benchmarking suite using ROS 2, designed to evaluate computing performance in robotics across various hardware. It integrates black-box and grey-box testing methods for comprehensive performance assessment and is adaptable for custom ROS 2 computational graphs. Developed with expert inputs, it aims to standardize robotics benchmarking and evolve with community contributions.	2023	1	Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Methods for Objective Comparison of Results in Intelligent Robotics Research

Table A.4: Peer-reviewed articles group 2.

## A.4 Preprint articles (including *under review*)

Table A.5 summarizes other **non peer-reviewed** results while classifying them according to the four levels of Figure 1.2.

Title	Summary	Year	Block
DevSecOps in Robotics [17] (paper)	A set of best practices designed to help roboticists implant security deep in the heart of their development and operations processes.	2020	1
Alurity, a toolbox for robot cybersecurity [19] (paper)	Alurity is a modular and composable toolbox for robot cybersecurity. It ensures that both roboticists and security researchers working on a project, have a common, consistent and easily reproducible development environment facilitating the security process and the collaboration across teams	2020	1
Can ROS be used securely in industry? Red teaming ROS-Industrial [86] (paper)	Red team ROS in an industrial environment to attempt answering the question: Can ROS be used securely for industrial use cases even though its origins didn't consider it?	2020	2
An Introduction to Robot System Cybersecurity [92] (paper)	This article addresses the often overlooked aspect of security in rapidly developed robotic systems, advocating for its integration from the initial design phase. It proposes quantitative methods and game theory for robust security management, tailored to the complex, distributed nature of robotics. The focus is on balancing affordability with effective security, providing a comprehensive guide for practitioners and theorists to enhance protection against diverse cyber threats in robotics.	2021	1
A Security Analysis of the Data Distribution Service (DDS) Protocol [97] (paper)	This study revealed significant security vulnerabilities in the Data Distribution Service (DDS), a middleware used in critical systems globally, including ROS 2. We identified 13 new CVEs across DDS implementations and found hundreds of public-facing DDS services, posing serious risks like denial of service and loss of control. The findings underscore the urgent need for continuous security testing of DDS. The research was a collaborative effort involving experts from multiple organizations, highlighting the widespread concern for DDS security.	2022	2
ExploitFlow, cyber security exploitation routes for Game Theory and AI research in robotics [20] (paper)	ExploitFlow (EF) is introduced in this paper as a modular library integrating Game Theory and AI for cybersecurity, specifically targeting robotics. It automates attacks by combining various exploits and analyzing system states to identify potential attack paths.	2023	1
PentestGPT: An LLM-empowered Automatic Penetration Testing Tool [292] (paper)	This study examines the use of Large Language Models (LLMs) for automating penetration testing, revealing their strengths in individual tasks but challenges in overall scenario comprehension. To address this, the researchers introduce PentestGPT, an advanced LLM-based tool with specialized modules to enhance penetration testing efficiency. Outperforming standard LLMs, PentestGPT shows significant improvements in task completion and has gained notable attention in the cybersecurity community, demonstrating its practical applicability in both academic and industrial settings.	2023	3
RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance [347] (paper)	RobotPerf is a vendor-neutral, open-source benchmarking suite using ROS 2, designed to evaluate computing performance in robotics across various hardware. It integrates black-box and grey-box testing methods for comprehensive performance assessment and is adaptable for custom ROS 2 computational graphs. Developed with expert inputs, it aims to standardize robotics benchmarking and evolve with community contributions.	2023	1

Table A.5: Preprints, this includes *under review* documents

## **A.5 Tutorials and talks**

Table A.6 summarizes participation in peer-reviewed tutorials and talks across workshops and conferences while classifying them according to the four thesis building blocks of Figure 1.2.

## **A.6 Vulnerabilities found**

Table A.7 summarizes vulnerabilities found during the research period that have been awarded with a CVE ID:

## A. Research Results

Title	Summary	Year	Block	Type/Publisher
Cyber security for robotics workshop (part1, part2)	Workshop on robot cybersecurity where to argue about robot-related cybersecurity topics and challenges while providing a deep insight and discussion about potential cyber-risks the current robotics ecosystem is facing.	2020	0	European Robotics Forum (ERF) 2020
Workshop on Security and Privacy in Robotics Zero Trust Architecture in Robotics (site)	Introduced a security architecture for robots that makes no trust assumptions and demands strict identity verification for every person, device or sub-component trying to access resources on a robot network (internal or external), regardless of whether they are sitting inside or outside of the network perimeter.	2020	1	2020 International Conference on Robotics and Automation (ICRA)
Cybersecurity in Robotics Tutorial: Demonstrating vulnerabilities and methods to protect robots (site)	This tutorial spotted diverse cybersecurity vulnerabilities and attack patterns for robotic systems and demonstrated new quantitative methods of cybersecurity management and design. In particular, a model-based security approach with game theory was presented to address security issues related to the heterogeneity and complexity of robotic systems	2021	1	IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2021
Conference Talk: Small Wonder: Uncovering Planned Obsolescence Practices in Robotics and What This Means for Cybersecurity (site)	We introduce and promote systematic "robot teardown" as an approach to repair robots by understanding their internals (still obscure). We show several "tricks from the trade" and the legal implications learned by porting reverse-engineering practices into the less-explored field of robotics. We explain how we a) discovered more than 90 security vulnerabilities in robots from Teradyne (MiR and UR) over a period of two years (never discussed publicly before), b) gained repairing capabilities on these robots, c) show evidence of planned obsolescence by comparing two sequentially released robot controllers, and d) demonstrate how robot hacking leads us to repurpose an older controller (previous version) from Universal Robots with their newer robots (arms) maintaining full capabilities and demonstrating that there's no need to re-spend thousands of dollars again. Paper produced for the venue available here	2021	2	BlackHat 2021 USA
1st International Workshop on Cyber Forensics and Advanced Threat Investigations in Emerging Technologies: Hacking planned obsolescence in robotics, towards security-oriented robot teardown (site)	The main motivation for this Workshop is to bring together researchers and practitioners working on cyber forensics and threat investigations for emerging technologies to disseminate current research issues and advances. Original technical papers describing new, state-of-the-art research, will be considered. The Workshop welcomes submissions that evaluate existing research results by reproducing experiments. The aim of this workshop is to provide insight for the discussion of the major research challenges and achievements on various topics of interest.	2021	2	5th International Conference on Networked Systems (Netsys 2021)
Conference Talk: The Data Distribution Service (DDS) Protocol is Critical: Let us Use it Securely (site)	We discovered and disclosed vulnerabilities in most of the OMG Data Distribution Service (DDS) implementations. DDS enables crucial technologies like autonomous driving, healthcare machinery, military tactical systems, or missile launch stations. Notably, DDS is used by NASA at the KSC, by SIEMENS for smart grid applications, by Volkswagen and Bosch for autonomous valet parking systems, by NAV CANADA for ATC, and by the Robot Operating System 2 (ROS2) to control industrial and consumer robots. Presentation available at here	2021	2	BlackHat 2021 Europe

Table A.6: Workshops, tutorials and talks in peer-reviewed conferences.

CVE ID	Summary	Scope	CVSS	RVSS [90]
CVE-2019-19626	Bash scripts (magic UR files) get launched automatically with root privileges and without validation or sanitizing	Universal Robots CB-series UR3, UR5, UR10	6.8	10
CVE-2020-10267	Unprotected intellectual property in Universal Robots controller CB 3.1 across firmware versions	Universal Robots CB-series UR3, UR5, UR10 and e-Series UR3e, UR5e, UR10e	7.5	6.5
CVE-2020-10268	Terminate Critical Services in KUKA controller KR C4	Any KUKA robot driven by the KUKA KR C4 controller, e.g. the KUKA KR 3 R540	6.1	7.1
CVE-2020-10271	MiR ROS computational graph is exposed to all network interfaces, including poorly secured wireless networks and open wired ones	MiR-based fleet of AMRs and derivative robots including MiR100, MiR200, MiR250, MiR500, MiR1000, ER-Lite, ER-One, etc	8.0	10.0
CVE-2020-10278	Unprotected BIOS allows user to boot from live OS image. The BIOS onboard MiR's Computer is not protected by password, therefore, it allows a Bad Operator to modify settings such as boot order. This can be leveraged by a Malicious operator to boot from a Live Image.	MiR-based fleet of AMRs and derivative robots (MiR100, MiR200, MiR250, MiR500, MiR1000, ER-Lite, etc)	6.1	7.1
CVE-2020-10269	Hardcoded Credentials on MiRX00 wireless Access Point. MiR fleet vehicles comes pre-configured in WiFi Master (Access Point) mode. Credentials to such wireless Access Point default to well known and widely spread SSID (MiR_RXXXX) and passwords (omitted). This information is also available in past User Guides and manuals which the vendor distributed. We have confirmed this flaw in MiR100 and MiR200 but it might also apply to MiR250, MiR500 and MiR1000.	MiR-based fleet of AMRs and derivative robots (MiR100, MiR200, MiR250, MiR500, MiR1000, ER-Lite, etc)	9.8	9.8
CVE-2020-10272	MiR ROS computational graph presents no authentication mechanisms. MiR100, MiR200 and other MiR robots use the Robot Operating System (ROS) default packages exposing the computational graph without any sort of authentication. This allows attackers with access to the internal wireless and wired networks to take control of the robot seamlessly. In combination with CVE-2020-10269 and CVE-2020-10271, this flaw allows malicious actors to command the robot at desire.	MiR-based fleet of AMRs and derivative robots (MiR100, MiR200, MiR250, MiR500, MiR1000, ER-Lite, etc)	8.0	10.0
CVE-2020-10287	Hardcoded default credentials on IRC 5 OPC Server. The IRC5 family with UAS service enabled comes by default with credentialsthat can be found on publicly available manuals. ABB considers this a well documentedfunctionality that helps customer set up however, out of our research, we foundmultiple production systems running these exact default credentials and considerthereby this an exposure that should be mitigated. Moreover, future deploymentsshould consider that these defaults should be forbidden (user should be forced tochange them).	All robots powered by ABB IRC5 controller OPC Server (e.g. ABB IRB140)	9.1	10

Table A.7: Some of the most relevant robot vulnerabilities found during research.



## Appendix B

# Game Theoretic Appendix

### B.1 CUT-THE-ROPE Background

APTs, like most targeted attacks conducted by cybercriminals, due to their diverse combination of attacks, hardly admit a single model to capture them; rather, they call for a combination of models designed for different aspects or characteristics of the attack. Game theoretic defense models may be distinguished according to the nature of APT [348] that they cover: there is the parasitic type, in which the attacker tries to steal resources for as long and much as possible, but does not aim to kill its victim. Related models are FlipIt [349, 350] and its descendants. Minimizing the total time that the attacker spends in the system may not necessarily minimize damage too, since the attacker may entirely destroy the asset  $v_0$  even within a very short period of time. The defender may nonetheless suffer a permanent defeat (upon loss of  $v_0$ ). For example, if the attacker can gain access to the security controls of a nuclear power plant even for a very short time, this may be sufficient to cause an unstoppable meltdown. Conversely, the attacker may spend a considerably larger amount of time in other areas of the nuclear power plant's system; as long as there is no vital subsystem to fiddle with, the damage to the infrastructure may be bearable. This motivates the consideration of the second type of APT, for which the game model Cut-The-Rope is tailored to: there, the attacker aims to kill the victim and silently prepare the final blow. A documented case of this is Stuxnet [351], and Cut-The-Rope is a game model designed for this latter type.

Many other game models are aligned with the phases in the kill chain, and most related work [352] is specific for at least one of them. We note that the ADAPT project [353] covers a wide spectrum of aspects and phases here. Specific defense models include the detection of spying activities [354], tracing information flows [355], detection of malware [356], deception [357] also via honeypots [358], attack path prediction [359], path selection to support malware detection in distributed networks [360], and general network defense [361] to name only a few. Our game is in a way similar

to that of the seminal work [362], yet differs from this previous model in not being stochastic, and in using payoffs that are not real-valued. The stochastic element is included in a much simpler way in our model, yet preserving information about uncertainty in a full distribution, to avoid losing information by averaging out randomness (for example, replacing a random payoff by a real-valued expected payoff).

Since the methods applied here come from the risk management field, this relates our work to that of [363], who presents a framework to optimally respond to a detected APT. Their work is thus an *a posteriori* treatment after the APT succeeded, while ours complements the risk management here by an *a priori* treatment to prevent the APT from success. Likewise notable is also the work of [364, 365], who consider inter-dependency graphs in relation to attack graphs in a game-theoretic analysis of targeted attacks. Their work adds constraints to budgets or desirable risk levels, and is specifically about investments in defenses of nodes and edges, but also works with crisp payoff measures (such as, e.g., paths of maximal attack probability or similar).

A different classification of related work is based on the protection targets. defenses can be optimized for confidentiality [366], the monetary value of some asset upon theft or damage [367], or the time that an adversary has parts of the system under control [349]. This distinction can be important depending on the context, as industrial production typically puts priority on availability and integrity, with confidentiality as a secondary or tertiary interest. Conversely, whenever personal data is processed, confidentiality becomes the top priority, putting availability further down on the list.

The techniques applied to capture and defend against APTs are manifold, but in most of these (like in our work), the network graph is in the center of attention: it may define how an attack evolves as a dynamical system [368, 369] inside the graph topology, with the challenge of optimized orchestrated defense. A good defense design that needs to account for new vulnerabilities potentially being opened up when closing known security holes. The work of [370], in this regard, utilizes a game model for graph coloring for a systematic and optimized defense, applying these results to industrial bus systems. Another dynamic yet queuing-based model is that of [371], which like our model computes optimal resource allocations by the defender and attacker, as an aid for decision making. Tailoring the attack model more closely to the application domain for the sake of a more

accurate description, the work of [372] provides insightful connections of graph topological properties of a power grid, and how areas in danger of becoming attacked are identifiable from analyzing the graph.

The work of [373, 374] takes a more birds eye perspective on the domain of the internet of things (IoT), and applies it directly to or varies the FlipIt game (see the references above and [350]) to model individual parts of a cloud-based IoT infrastructure, combining these submodels into a larger hybrid game model that allows certain equilibria to play optimally against the adversary. Another cloud-related and -specific APT defense model is [375]. Like us, they adopt a leader-follower model, but different to our work, they use a Stackelberg equilibrium concept.

Taking the APT as a long term yet one-shot event, an attack graph can be treated as a (big) game in extensive form. From this point of view, it is possible to think of the APT as an instance of the induced gameplay, to which Bayesian or subgame perfect equilibria can be sought [376]. More similar to this work, we can treat the APT as a game of inspections, to discover optimal strategies of inspection in different depths of a shell-structured defense [367, 377]. An aspect of strong relevance concerns the use of probabilities: the work of [378, 379] are most interesting in its account for subjective probability and prospect theory, since this includes the way of how humans bring in their individual risk attitudes in decision making under uncertainty (especially about defenses). We avoid this conceptual and practical difficulty in the modeling by designing our game with as few probabilistic parameters as possible.

Cut-The-Rope is, in two ways, different from most other game theoretic models: first, it can let the players act in different time axes, meaning that the defender can be active in discrete or continuous time, while the attacker is (here always) acting in continuous time. This is in contrast to most other models in which both players act in fixed schedules (such as in extensive form games), or both can take actions continuously (such as in differential games). The second aspect is the added suggestion of tie-breaking if there are several equilibria. Cut-The-Rope implicitly addresses the equilibrium selection problem by refining the set of possibly many defense actions based on the probabilities to reach not only  $v_0$ , but also to get nearby it. Formally, the optimization, after having minimized the chance to conquer  $v_0$ , continues by minimizing the chances to reach a node close to  $v_0$ . As mentioned in Section 10.2.1 this induces a dependency on the ordering of nodes, but this ordering is up to the choice of the defender setting up the



attack graph with uniform probability. The game is round-based, where the exact meaning of a round depends on the moving patterns of the defender and the attacker:

- If the defender acts periodically in fixed intervals (e.g., daily), a round of the game is one period of activity for the defender (e.g., one day). During this period of time, the attacker can take a random (unlimited) number of steps along the attack path towards the goal.
- If the defender is taking action at random, e.g., taking exponentially distributed pause times, then a round of the game is, in each instant, the random idle time of the defender. Again, during these periods, the attacker can take any number of actions, depending on its “configuration” and/or the attack path. In the terminology of the FlipIt game [349], this is called an *exponential defense strategy*.

In both cases, we do not explicitly model the time to complete a spot-check and merely assume this completion to be possible within one unit of time. Including the defender’s costs for spot-checking as a separate goal (to minimize) makes the game multi-criteria and calls for Pareto-optimization, which we leave out of our scope in this work (and up to future considerations). We will come back to the exact meaning of a “round” or “unit of time” in Section B.3.1. Let us first complete the description of the gameplay: Figure B.1 displays two attack paths, with the lower path showing the step-by-step traversal of an avatar towards the goal  $v_0$ . Every possible action of the defender is here called a *spot-check* at any node in  $V \setminus \{v_0\}$ , where the target node is excluded to avoid trivialities<sup>1</sup>. A *spot-check* can mean any action that, for example, (i) cleans a component from malware, (ii) disables certain services that an exploit would rely on, (iii) changes in the security policy or implementation that invalidates the adversary’s knowledge (e.g., access control mechanisms), or similar. Common to all actions of the defender is their *transient* efficacy, which means that the effect of such an action is *not* permanent (the opposite case is discussed later in Section 10.4.2). After the action, and not necessarily known to the defender, the attacker is sent back on the attack path to

---

<sup>1</sup>If the defender would not move away from the target, there would be nothing to accomplish here for the adversary and there would be nothing to analyze.

an earlier position (upper part of Figure B.1). For example, if the so-far accomplished route has at some point used access credentials for a computer, and the defender has just changed them, the route is essentially closed at this point, and the attacker has to re-try just before this point<sup>2</sup>. The avatars can go unaffected by the defender's action in two cases: (i) if it travels on a different route that the defender did not inspect in this moment (e.g., lower attack path in Figure B.1), or (ii) the attacker started from a location below the cut point (e.g., if the attacker is left to the cut point  $\checkmark$  in Figure B.1). This assumption implicitly accounts for "out of attack graph" ways of the attacker having reached this location. In either case, the avatar's journey is not intercepted.

The attacker may at any point decide to try a different route instead. This is called *lateral movement*. It is naturally included in this modeling by having avatars on all attack routes, which makes lateral movement nothing else than moving other avatars on another route. Cut-The-Rope is played under the assumption that an avatar can be thrown back to an earlier point by the defender, but will in any case re-try its current attack path, until it (or any of its clones) has reached the goal.

The payoffs in the game are zero-sum, and come to the probability of reaching  $v_0$  in a single round of the game. This is the payoff to the attacker, and likewise the loss of the defender, who seeks to minimize this probability. Its computation depends on the probability distribution law that governs how many steps  $N$  can be taken during the defender's idle periods. This is the main ingredient whose influence is studied in this work, relative to a heuristic best-practice defense.

The *payoff* to the adversary is the chances for any of its avatars to reach, from its current position, the target  $v_0$  within  $N$  steps and within the time-limit  $W$ , during which the defender is idle<sup>3</sup>. This can be fixed (for a periodic defender) or random (for an exponential defense strategy). We collect all avatars in a set  $\Theta \subseteq V \setminus \{v_0\}$ , and denote individual avatars as  $\theta \in \Theta$ . The exclusion of  $v_0$  from this set is to avoid the trivial case where the attacker has already reached  $v_0$  before the defense game starts. The

---

<sup>2</sup>We herein assume that there is no direct way to just get back to the later point: if there would be such a shortcut route bypassing the just-closed backdoor, this would be another attack path, taken by a respectively other avatar.

<sup>3</sup>Here, we simplified the payoff representation from a vector-valued distribution in [286] over the attacker distance to the goal, to the probability of reaching the goal. This does not affect the solution of the model, but facilitates readability.

payoffs to both players are:

$$u_{\text{attacker}} = -u_{\text{defender}} = \Pr(\text{adversary reaches } v_0) \quad (\text{B.1})$$

We will formally determine this quantity in Section B.2.1 in expressions (B.7) and (B.8), which make the dependency on the strategic choices of the defender and attacker visible and explicit.

## Strategies

We now turn to the description of how the defender's and attacker's action determine the probability to reach  $v_0$ . The strategic choices of both players towards maximizing or minimizing  $\Pr(\text{adversary reaches } v_0)$  are the following:

- The defender has a choice from the set  $AS_1 := V \setminus \{v_0\}$  to spot-check, giving a total of  $n = |AS_1|$  actions. We will write  $\mathbf{x} \in \Delta(AS_1)$  for a randomized such spot-checking rule.
- The attacker can likewise use a total of  $m = |AS_2|$  attack paths in  $G$ , collected in the set  $AS_2$ . Each avatar starts from a different location  $\theta \in \Theta = V \setminus \{v_0\}$  and traverses one of the (perhaps many) routes from  $\theta$  towards  $v_0$ . The adversary solution in the game is the best choice of attack paths from  $AS_2$ . Likewise, we will write  $\mathbf{y} \in \Delta(AS_2)$  for a random choice from the set of attack paths.

Every avatar takes action by being moved forward along the attack path that it is on, and draws/samples a random number  $N$  from a fixed step-distribution  $f_N$ . This is *not* a strategic choice, but rather a part of the game's *payoff mechanism*. Low-level procedures of how the avatar technically mounts exploits are not expressed nor modeled in the game itself (due to the heterogeneity and sheer number of possibilities of exploits in a real-life attack graph).

### B.2.1 Definition of Payoffs

For the sake of rigor, let us concretize (B.1) by showing how it is practically obtained. This will also display the role of the movement patterns (periodic, exponential) in the experimental analysis. Working out the adversary's

utility is a matter of conditioning the attack step distribution  $F_N$  on the current situation in the network, i.e, the position of the avatar and where the defender took action.

Let  $\pi_1, \pi_2, \dots, \pi_m$  be an (exhaustive) enumeration of all attack paths, each starting from another location  $\theta \in \Theta \subseteq V \setminus \{v_0\}$ . Each starting location is thus identified with one avatar, and the adversary moves all of them towards  $v_0$ . Let  $m$  be the total number of all attack paths.

Each such path is again a sequence of nodes, written as  $\pi = (\theta, w_1, w_2, \dots, v_0)$  with all  $w_i \in \{v_1, v_2, \dots\} = V$  and  $\theta \in \Theta$  being the starting point of the route, one-to-one corresponding to an adversarial avatar. The set of nodes constituting  $\pi$  is  $V(\pi)$ . Furthermore, let  $d_\pi(u, v) \in \mathbb{N}$  count the edges on the path  $\pi$  from  $u$  to  $v$ . It is a graph-theoretic distance.

Then, the *location distribution* for the attacker assigns to each node  $v \in V$  the mass

$$\Pr(\text{avatar location} = v | V(\pi)) = \frac{f_N(d_\pi(\theta, v))}{\Pr_N(V(\pi))}, \quad (\text{B.2})$$

in which  $f_N(n) = \Pr(N = n)$ , where  $N \in \{0, 1, 2, 3, \dots\}$  is the random number of steps undertaken by the avatar, and

$$\Pr_N(V(\pi)) = \sum_{x \in V(\pi)} \Pr_N(d_\pi(\theta, x)) = \sum_{x \in V(\pi)} f_N(d_\pi(\theta, x)). \quad (\text{B.3})$$

The probability density  $f_N$  will be the main element to vary when describing different attacker-defender scenarios (such as announced in the introduction under the contributions). We will give various options to define  $f_N$  in equations (B.10), (B.11), (B.12) and (B.13).

Now, the defender attempts to break the attacker's chain of exploitation ("cut the rope" in the wording of [286]). Let  $c \in V$  be the checked node, then the possibly truncated path is

$$\pi|_c = \begin{cases} (\theta, w_1, w_2, \dots, w_{i-1}), & \text{if } c = w_i \text{ for some } w_i \text{ on } \pi \\ (\theta, w_1, \dots, v_0), & \text{otherwise.} \end{cases} \quad (\text{B.4})$$

The closing of a backdoor here becomes a conditioning of the distribution of the avatar's location on the shorter (cut) path  $\pi|_c$ . The formula is the same as (B.2), only with  $\pi$  replaced by  $\pi|_c$  now. Since  $c \sim \mathbf{x}$  follows the

defender’s mixed spot checking strategy (possibly degenerate), and the set of paths  $\pi$  along which avatars proceed, the defender can determine the possible locations of the attacker, based on the imagined avatars, as the vector of probabilities

$$U = (\Pr(\text{adversary's location} = v | V(\pi|_c)))_{v \in V}, \quad (\text{B.5})$$

which depends on the random choices of the defender (“where to cut?”) and the attacker (“which route to take?”). This is what the implementation of Cut-The-Rope computes.

The actual quantity of interest for the game, coming back to (B.1), is the mass that  $U$  assigns to  $v_0$ . This is the utility for the adversary and conversely the loss of the defender. Since the game is, from the attacker’s perspective, a strategic choice  $\mathbf{y} \in \Delta(AS_2)$  of an attack path, the payoffs in the game are obtained from the following consideration:

$$\begin{aligned} \Pr(\text{attacker reaches } v_0) &= \Pr(\text{at least one avatar reaches } v_0) \\ &= \sum_{\theta \in V \setminus \{v_0\}} \Pr(\text{avatar reaches } v_0 \text{ starting from } \theta) \cdot \Pr(\theta) \\ &= \sum_{c, \pi} \sum_{\theta \in V \setminus \{v_0\}} \underbrace{\Pr[\text{avatar } \theta \text{ has location } v_0 | V(\pi|_c)]}_{\text{from eq. (B.2) and (B.4)}} \cdot \underbrace{\Pr[\text{path } \pi \text{ is chosen and defender cuts at } c]}_{\text{strategic choices to optimize}} \cdot \underbrace{\Pr(\theta)}_{=1/|AS_2|} \quad (\text{B.6}) \\ &= \sum_{c, \pi} \sum_{\theta \in V \setminus \{v_0\}} \Pr \left[ \text{avatar } \theta \text{ has location } \right. \\ &\quad \left. v_0 | V(\pi|_c) \right] \cdot \Pr_{\mathbf{x}}(c) \Pr_{\mathbf{y}}(\pi) \cdot \Pr(\theta) =: u_{\text{attacker}}(\mathbf{x}, \mathbf{y}) \quad (\text{B.7}) \\ &= -u_{\text{defender}}(\mathbf{x}, \mathbf{y}) \quad (\text{B.8}) \end{aligned}$$

The equality in the second line herein follows from the fact that the attacker will move one avatar at a time, so that no two avatars will simultaneously reach  $v_0$ . The first avatar to reach  $v_0$  will make all others stop, so that the respective events become disjoint.

### B.2.2 Solution Concept

An *instance* of Cut-The-Rope is a quintuple  $(G, v_0, AS_1, AS_2, f_N)$ , containing: the attack graph  $G = (V, E)$ , the target node  $v_0 \in V$ , the defender's possible spot check locations  $AS_1 \subseteq V \setminus \{v_0\}$ , the possible locations  $AS_2 \subseteq V \setminus \{v_0\}$ , for the attacker's avatars. These avatars will move towards  $v_0$  along the attack paths encoded in  $G$ , taking a random number  $N$  of steps distributed according to the probability density  $f_N$ . This density determines the particular behavior of the attacker, relative to the defender's actions, and will be generally given in Section B.3, and instantiated for the two real-life use-cases in Section 10.3.

A *solution* for a given instance is obtained with standard techniques to compute Nash equilibria: With both players having a finite set of choices, and the utility  $\Pr(\text{adversary's location} = v \mid V(\pi|_c))$  derived from the location distribution (B.5) that depends on the attack path  $\pi$ , movement pattern  $f_N$  and spot-check location  $c$ , we end up with a (normal-form) matrix game that we can analyze for an equilibrium using known techniques. The solution concept used in this work is a *security strategy* for the defender, having the following (informal) semantics: it is the best randomized choice rule  $\mathbf{x}^* \in \Delta(AS_1)$  such that

$$u_{\text{defender}}(\mathbf{x}^*, \mathbf{y}^*) \leq u_{\text{defender}}(\mathbf{x}^*, \mathbf{y}) \quad \text{for all } \mathbf{y} \in \Delta(AS_2) \quad (\text{B.9})$$

That is, the defender can, upon playing the optimal spot checking strategy  $\mathbf{x}^*$ , enforce the worst-case minimal likelihood for the attacker to reach  $v_0$ , for all choice rules  $\mathbf{y} \in \Delta(AS_2)$ , i.e., irrespectively of what the attacker actually does.

The security strategy is computable by solving a conventional matrix game, which is finite since there are only finitely many spot check locations, and likewise finitely many attack paths. The game matrix is thus computable by evaluating formula (B.7), for all locations  $c \in AS_1$  and all paths  $\pi \in AS_2$ . The Nash equilibrium of this game is  $(\mathbf{x}^*, \mathbf{y}^*)$ , in which  $\mathbf{x}^*$  is the sought security strategy, and  $\mathbf{y}^*$  is the optimal choice rule for the attack paths towards  $v_0$ .

The latter information is, however, of limited use for the defender, since equilibria are generally not unique. Therefore, taking  $\mathbf{y}^*$  as a guidance on where to find for the invisible intruder with highest probability can be misleading, since there may be (plenty of) other equilibria giving entirely different advice.

On the contrary, since the saddle point value giving the lower bound value in (B.9) is invariant w.r.t. different equilibria  $(\mathbf{x}^*, \mathbf{y}^*)$  any alternative defense advice cannot accomplish any better lower bound for the defender. Hence,  $\mathbf{x}^*$  is in fact useful as optimal advice.

*Remark B.2.1.* The original solution concept proposed in [286] has been a perfect Bayesian equilibrium, but this raises issues with the interpretation of the results. While the game's setting formally *fits* into the definition of a perfect Bayesian equilibrium as given by [380], it *does not fit* equally well into the interpretation thereof: the game is not about signaling, while the solution concept in [286] took an equilibrium designed for signaling games. Also, there is no random conditioning on adversary types, which a Bayesian equilibrium would require. Rather, Cut-The-Rope is – from the defender's point – played with avatars, all of which concurrently move on their routes, without a particular type choice made by nature. Therefore, a security strategy (computed as a Nash equilibrium) is the more suitable solution concept.

## B.3 Movement Patterns

In lack of any particular knowledge about the difficulty of the attack path, a simple heuristic is to just use the *shortest path*, in a graph theoretic sense. This will later also be the intuitive benchmark (see Section 10.3.1) to compare the defense obtained from Cut-The-Rope to a defense based on the (plausible) assumption that the attacker takes the shortest/easiest route towards  $v_0$ .

### B.3.1 Periodically Active Defender

In the simplest case, originally proposed in [286], we assume that the defender becomes active in fixed time intervals that are known to the attacker. The *unit of time* (see the previous section) is herein the period in which the defender becomes active (each day, each week, or similar). Furthermore, we assume no particular cost for the attacker to penetrate (this case is covered in Section B.3.2). This corresponds to the situation of having a “just conceptual” attack graph, displaying general strategies to penetrate, but without reference to concrete exploits, CVE numbers or similar.

During the defender’s idle times, we assume an average number  $N \sim \mathcal{Pois}(\lambda)$  of steps towards its target at “average speed  $\lambda$ ”. This analytical choice is common in related literature (see, e.g., the FlipIt Game [349] to describe APTs, calling this strategy “exponential”). Empirically estimating the rate parameter from data, for example, taken from intrusion detection or other monitoring systems is an interesting challenge of independent research.

The function  $f_N$  for a periodic defender and attacker with average speed  $\lambda$  is the Poisson distribution density

$$f_N(n) = f_{\mathcal{Pois}(\lambda)}(n) = \frac{\lambda^n}{n!} e^{-\lambda}, \quad (\text{B.10})$$

which would be substituted into (B.2) and (B.3) to set up the game.

The value  $\lambda$  must be set relative to the frequency at which the defender takes actions. For example, if the attacker makes two attempts per day, and the defender does one spot check per week, then we have  $\lambda = 2 \times 7 = 14$ . If the defender checks twice per day, then the attack rate is  $\lambda = 2 \times \frac{1}{2} = 1$ . The actual choice of  $\lambda$  was, experimentally, found to mostly impact the likelihood to hit  $v_0$ . The defense advice, however, did not significantly change (see Appendix B, section B.4), meaning that an inaccurate choice of  $\lambda$  in practice will deliver a respectively inaccurate estimate on how likely  $v_0$  will fall, but can nonetheless deliver a valid defense recommendations.

We assume that the defender has knowledge (or a reasonable assumption) about  $\lambda$ , so that he is able to adapt the defense to it accordingly, as the security resources permit. The choice of  $\lambda$  itself can be considered as a strategic decision for the attacker too, knowing the defender’s behavior. However, we do not explore this variation here any further, as it leads to a different game, but point out this investigation as a separate research question. We refer to the work of [379] as being a game about computing the optimal check-intervals explicitly with help from prospect theory, and under some assumptions on the attacker’s attitude (risk appetite, etc.), but not considering attack graphs.

### B.3.2 Probabilistic Success on Exploits

The attacker may not necessarily succeed in every penetration that it attempts. As before, if we assign probabilities <sup>4</sup>  $q(e)$  to express the chance

---

<sup>4</sup>For example, using subjective probability, prospect theory and generally empirical studies on human risk perception and subjective assessments, CVSS ratings to derive probabilities

of a successful exploit  $e$  on the respective attack path. Formally,  $q(e)$  could be equated to the likelihood of meeting some precondition to penetrate a node. Let us slightly change the view to think of an attack path  $\pi$  as a sequence of exploits  $\pi = (e_1, e_2, \dots)$  (instead of nodes). Then, the chances to progress forward by a lot of  $n = 0, 1, 2, \dots$  steps is no longer Poisson distributed; rather, assuming stochastic independence of exploits, the chances to take  $n = 0, 1, 2, \dots$  steps are

$$f_N(n) = (1 - q(e_{n+1})) \cdot \prod_{k=1}^n q(e_k), \quad (\text{B.11})$$

i.e., the probability to succeed with exactly  $n$  exploits, and to fail on the  $(n + 1)$ st step on the attack path. This function then goes into (B.2) and (B.3) to instantiate the game under the setting described here. A *unit of time* is, again, the period between two appearances of the defender in the system, again taken as fixed and constant over time (e.g., one day, one week, etc.).

### B.3.3 Checks with Random Intervals (“Exponential Strategy”)

If the defender becomes active at its own random (Poisson) rate  $\lambda_D$ , the attacker will be able to take a  $\mathcal{Pois}(\lambda)$ -distributed number of steps in an exponentially distributed pause time controlled by the defense intensity  $\lambda_D$ . This defense regime defines a *random unit of time*, whose long run average is exactly  $\lambda_D$ .

This change of the setting amounts to a humble change of the Poisson distribution into a geometric distribution, because: we now have two types of events to consider, which are activity of the attacker at rate  $\lambda$  and activity of the defender, at rate  $\lambda_D$ . Within a unit of time, we will thus have a number  $k_A$  of attack events, vs. a number  $k_D$  of defense actions. So, the likelihood of the defender to become active is (frequentistically) approximated as  $p = k_D / (k_A + k_D) = \frac{1/n \cdot k_D}{1/n \cdot (k_A + k_D)}$  for all  $n > 0$ . The last term, however is the average number of events per  $n$  time units, which upon  $n \rightarrow \infty$  converges to  $\lambda$  for  $k_A/n$  and to  $\lambda_D$  for  $k_D/n$ . Thus, the probability for an action to be taken by the defender is  $p = \lambda_D / (\lambda_D + \lambda)$ , and the number of trials that the attacker can take until the defender becomes active again is a geometric distribution with that parameter  $p$ . Conceptually, the model

---

from, and others. Helpful related work hereto was done by [291, 378, 379].

thus remains unchanged, except that the attacker's step number is now computed using the geometric distribution density with the given rate parameter. Consequently, we have

$$f_N(n) = p \cdot (1 - p)^n \quad \text{with} \quad p = \frac{\lambda_D}{\lambda_D + \lambda} \quad (\text{B.12})$$

in (B.2) and (B.3).

### B.3.4 Spot Checks with Random Intervals and Probabilistic Success on Exploit

Unlike before, we now consider a *fixed unit of time*, in which an exploit for a given vulnerability can be tried. The defender comes back in random intervals, measured in the this (fixed) unit of time, and has an average return time of  $\lambda_D$ . Consequently, the time window for the attacker to run exploits is an exponentially distributed random variable  $W \sim \text{Exp}(\lambda_D)$ . Within this time window  $W$ , the attacker ought to accomplish  $n$  exploits, along an attack path  $\pi = \theta \rightarrow w_1 \rightarrow w_2 \rightarrow \dots \rightarrow v_0$ , in the notation of Section B.2.1. Like in Section B.3.2, let us call  $e_k$  the edge into node  $v_k$ , which carries a known exploit complexity as the quantity  $q(e_i) = \Pr(\text{exploit on } e_i \text{ is successful within a (fixed) unit of time})$ . Then, an exploit on edge  $e_i$  takes an exponentially distributed time  $T_i \sim \text{Exp}(1/q(e_i))$ . The total time for  $n$  exploits is thus  $T_1 + T_2 + \dots + T_n$ , which, unfortunately, does not admit a closed analytical expression for its distribution, since the values can be assumed independent, but not identically distributed. To escape the issue, we simplify matters by assuming the avatar to move at a uniform velocity along the attack path, instead of being faster and slower depending on the attack complexities. We believe this assumption to be mild, since our main concern is the time it takes to reach the end  $v_0$  anyway, and we are not as much interested in determining the avatar's location anywhere in the middle of the attack path.

This simplification comes to a geometric mean of the probabilities

$$\bar{q} = \text{geomean}\{q(e_i) \mid e_i \text{ is on the chosen attack path}\}.$$

The point is that the product of the actual probabilities, i.e., the chance to hit  $v_0$ , remains unchanged hereby, since  $\prod_i q(e_i) = \bar{q}^{|V(\pi)|}$  where  $|V(\pi)|$  is the length of the attack path. Let us put  $\lambda_\pi := 1/\bar{q}$  to bring the notation closer to that of Section B.3.3, since the result (to come later) will also be

close to this previous finding. The subscript  $\pi$  to  $\lambda_\pi$  herein reminds about the attack rate now to depend on the chosen path.

Under this simplification, the time for  $n$  exploits is the sum of all identically  $\mathcal{Exp}(1/\bar{q})$ -distributed random variables  $E_n := T_1 + T_2 + \dots + T_n \sim \mathcal{Erl}(\lambda_\pi, n)$  that is Erlang distributed. We are interested in the probability of  $T_1 + T_2 + \dots + T_n \leq W$ , which is a matter of computing a convolution integral. We shift the algebraic details to the appendix B, and directly give the result here:

$$f_N(n) = \begin{cases} \Pr(E_N \leq W) = \left(\frac{\lambda_\pi}{\lambda_\pi + \lambda_D}\right)^n & n \geq 1; \\ \Pr(E_1 > W) = 1 - \Pr(E_1 \leq W) & n = 0. \end{cases} \quad (\text{B.13})$$

Observe that this is movement pattern is like in Section B.3.3, which is yet another geometric distribution, only with the different parameterization.

The approach of geometric averaging over the entire attack path deserves a bit of discussion: we could equally well average only across the segment of length  $n$  that the attacker targets to overcome, and/or exclude all exploits with  $q(e_k) = 1$  from the averaging. We refrained from both these options for two reasons: first, removing the 1es from the averaging would unrealistically shorten the attack path to less than its physical reality. Even if an exploit has a 100% chance to be used within short time, there is nonetheless a time step necessary to do it, so including it in the geometric mean seems plausible. An attack path that is longer will, despite the same product probability of accomplishing it, take a proportionally longer time to traverse. Second, concerning the focus on only a segment, this may miss the actual intention of the adversary, since it targets the end of the attack path, and not only a specific segment. In other words, geometrically averaging only over the first  $k$  exploits would be the assumption that the adversary would stop at the  $k$ -th step, even if there is time left before the defender comes back. Since the target is getting to the end of  $\pi$ , it appears plausible to include all exploits towards this end.

## B.4 Analytic Results

For a plausibility assessment of the game's results, we analytically study the results on attack graphs with no information at all, so that we can intuitively (and without any model) determine a best defense. The analytic results to follow are consistent with the numeric findings for MARA, and,

more importantly, are independent of the attack rate  $\lambda$  (see Proposition B.4.3).

Given a set  $AS_2$  of attack paths, the defender's best strategy in Cut-The-Rope is to find and guard a minimal (in a sense to be defined later) graph-theoretic cut  $C \subseteq V$  between the starting node (set) of all attack paths, and the target asset  $v_0$ . Towards proving this claim, suppose that the defender would focus on a set  $S \subset V$  that is not a cut. Then, there is a path  $\pi$  that bypasses  $S$ , i.e.,  $S \cap V(\pi) = \emptyset$ , but this makes  $\pi$  a winning strategy for the attacker (since the defender can never catch the attacker on that route). Suppose that the defender's guarded set  $C$  were not minimal, i.e., the defender spot-checks on a superset  $S' \supset C$ . Then, we can distinguish two cases:

1. either no attack path passes through the nodes  $S' \setminus C$ , in which case defending them is useless, and hence defending  $S'$  is a sub-optimal strategy (as it consumes too many resources), or
2. there is at least one attack path through a node in  $S' \setminus C$  and another node in  $C$ . In that case, we can safely remove either of the two, since both would cut the rope in the sense we desire. This strictly shrinks  $S'$ , and we can repeat this reduction until the resulting set has become minimal (in terms of cardinality).

Compiling the thoughts above concludes the proof of the next result:

**Lemma B.4.1.** *Let  $s$  be the root of the attack graph, and let  $v_0$  be the target asset. Furthermore, assume that the defender can cut the rope anywhere in the graph, except at the starting point and the target (to avoid trivialities). The optimal strategy of defense in Cut-The-Rope is guarding an  $s$ - $v_0$ -cut of minimal cardinality. If there is more than one starting point or more than one target asset, the cut is understood between the respective sets thereof.*

Lemma B.4.1 makes no assertion about what cut to choose if there are several. For example, if we have only one attack path overall, then every node on it would be a valid cut. Intuitively, the best option is cutting the (single) rope as close as possible near  $v_0$ , in order to get the most likely locations covered from which an attacker's avatar could start. The proof of Lemma B.4.2 makes this rigorous:

**Lemma B.4.2.** *Assume that a defender's (mixed) strategy prescribes to spot-check on the attack path  $\pi$ . The best point to cut the rope is the location  $v$  whose distance to  $v_0$  along the path  $\pi$  is minimal.*

*Proof.* Consider the attack path  $\pi$  as a sequence of consecutive vertices  $(u_0, u_1, u_2, \dots, u_l = v_0)$ , and write  $V(\pi)$  to mean the set of all vertices on  $\pi$ . Call  $c \in V(\pi)$  the vertex whose distance  $d(c, v_0)$  is minimal among all  $V(\pi) \cap AS_1$ , i.e., all nodes on  $\pi$  that the defender has in its action set  $AS_1$  and can hence spot-check. Let  $c' \in V(\pi) \cap AS_1$  be another node to possibly check on the same path, which is distinct from  $c$ . It follows that either there is a connection  $c' \rightarrow c$  (if the two are consecutive) or there is at least one node in between  $c' \rightarrow \dots \rightarrow c$ . In either case, we have distinct avatars  $\theta_{c'}$  and  $\theta_c$ , corresponding to these two nodes as starting points. Both use the same distribution  $F_N$  with probability mass function  $f_N$ , for the number  $N$  of steps taken forward on  $\pi$ , only starting at different locations ( $c$  or  $c'$  hereafter). To ease notation in the following, let us associate the avatar  $\theta$  directly with a node on  $\pi$  (this creates no ambiguities).

The probability mass that an avatar  $\theta_i$  puts on  $v_0$  when starting from location  $i$  is given by the chances to take at least the residual distance  $d_\pi(\theta, v_0)$  from the starting point ( $\theta$ ) until  $v_0$ . Given the distribution function  $F_N$  of the random distance overcome upon adversarial activity, this is  $\Pr(N \geq d(\theta, v_0)) = 1 - F_N(d_\pi(\theta, v_0))$ . Throughout the rest,  $\pi$  and  $v_0$  will both be fixed, so we can safely omit them from our notation, so let us write  $\Delta_\theta := d_\pi(\theta, v_0)$ , for the residual distance on the path  $\pi$  between the avatar starting from  $\theta$ , and the target  $v_0$ . Moreover, put  $u_\theta := \Pr(N \geq \Delta_\theta) = 1 - F_N(\Delta_\theta) = \sum_{d \geq \Delta_\theta} f_N(d_\pi(\theta, v_0))$  to abbreviate the probability of the attacker to reach  $v_0$  within the next move.

The utility over all attacker avatars is then

$$\Pr(\text{asset } v_0 \text{ is lost to the attacker}) = \sum_{\theta \in \Theta} \Pr(\theta) \cdot u_\theta \quad (\text{B.14})$$

which is the total probability mass assigned to  $v_0$  by all adversary avatars.

Now, let us compare the effects of spot-checking  $c$  vs. spot-checking  $c'$  that is farther away from  $v_0$ . Since we have only the attack path  $\pi$  on which  $c'$  comes before  $c$ , let us break up the path into three corresponding parts

$\pi = (u_0, \dots, c' = u_i, \dots, c = u_j, \dots, u_l = v_0)$ , and expand (B.14) accordingly

$$\sum_{\theta \in \Theta} \Pr(\theta) \cdot u_\theta = \sum_{\theta \in (u_0, \dots, u_i = c')} \Pr(\theta) \cdot u_\theta \quad (\text{B.15})$$

$$+ \sum_{\theta \in (u_{i+1}, \dots, u_j = c)} \Pr(\theta) \cdot u_\theta \quad (\text{B.16})$$

$$+ \sum_{\theta \in (u_{j+1}, \dots, u_l = v_0)} \Pr(\theta) \cdot u_\theta. \quad (\text{B.17})$$

It will be helpful to remember the effect of truncating a distribution at  $t$ , which is switching from  $F_N(d)$  to the conditional distribution on  $F_N(d|d \leq t)$ , whose density is

$$f_N(d|d \leq t) = \begin{cases} \frac{f_N(d)}{F_N(t)}, & \text{if } d \leq t; \\ 0, & \text{otherwise.} \end{cases} \quad (\text{B.18})$$

The important fact is that cutting at some point on the path affects all avatars on the segment from the beginning node until the cut node  $c$  or  $c'$ . If we cut at  $c'$ , we take out the whole expression (B.15), leaving (B.14) = (B.16) + (B.17), in a slight abuse of formalism here. However, if we cut at  $c$ , term (B.16) also drops out of (B.14), leaving this to be the better option for the defender. ■

Now, we can compile the findings so far into a generic characterization of the defender's best choice:

**Proposition B.4.3.** *Let an acyclic attack graph  $G$  be with root node  $u_0$ , and let  $v_0$  be the target node (likewise, for sets thereof if there are multiple). Furthermore, let  $d$  be a distance measure in  $G$ . The defender's optimal strategy in Cut-The-Rope is spot-checking a minimum-cardinality  $u_0$ - $v_0$ -cut  $C$ , with the property that for each  $c \in C$ , the distance  $d(c, v_0)$  is minimal.*

## Consistency of Numeric and Analytic Results

The numeric findings for the MARA use case agree with the analytic predictions to defend the graph cut that is closest to the target nodes. The formal arguments in Section B.4 assume the same distribution for all possible paths, which does not hold for the MiR100 use case. Thus, the

optimal defense no longer needs to be a graph cut, and the numeric results about the MiR100 use case confirm this possibility. Since in the MiR100 case, the attack paths have different efficacies, strategic dominance among the attack paths may affect the results accordingly. Since the results, in this more general case, depend on the distribution conditional on the attack path, it appears unlikely that comparable analytic predictions can be made for the movement pattern of Section B.3.4, and we leave this as an open problem.

Regarding the heuristic defense, its bad performance in comparison to Cut-The-Rope can be attributed to the defender blindly checking everywhere on the attack graph, while the intuition (also behind the formal arguments of Section B.4) would rather advise to defend closer to the goal. This suggests that the optimization that Cut-The-Rope may be reasonably replaced by a heuristic defense, only focused on a graph cut subset of nodes, and indeed, the numbers for the MARA use case show an approximately uniform defense of nodes on such a cut to be optimal. Overall, however, it is advisable to run an optimization, since just adding the analytic prediction of where to defend to the heuristic is incorrect in the case where the traversal of an attack path depends on the path's properties, such as distinct difficulties to exploit, as in the MiR100 use case. Here, the performance of the defense is substantially better than for the heuristic, but the apparent focus on a graph cut is not found in the results.

## B.5 Application for Risk Control

Actions with a *permanent effect* change the attack surface by blocking certain paths, increasing the attack detection capabilities, or similar. Examples include the installation of a firewall, malware scanners, deactivation of services or accounts, and many more.

If the defender's action space includes at least one with potentially permanent effect, the attack graph, and hence the overall game, *changes* with the defender's activity, and the game must be re-instantiated before the next round after *pruning the attack graph*. This turns Cut-The-Rope into a *dynamic* game, but it is still repeated with infinite time horizon. It is fair to remark that the tree may not only become pruned, but introduce new attack paths upon inserting new components, installing new software or similar.

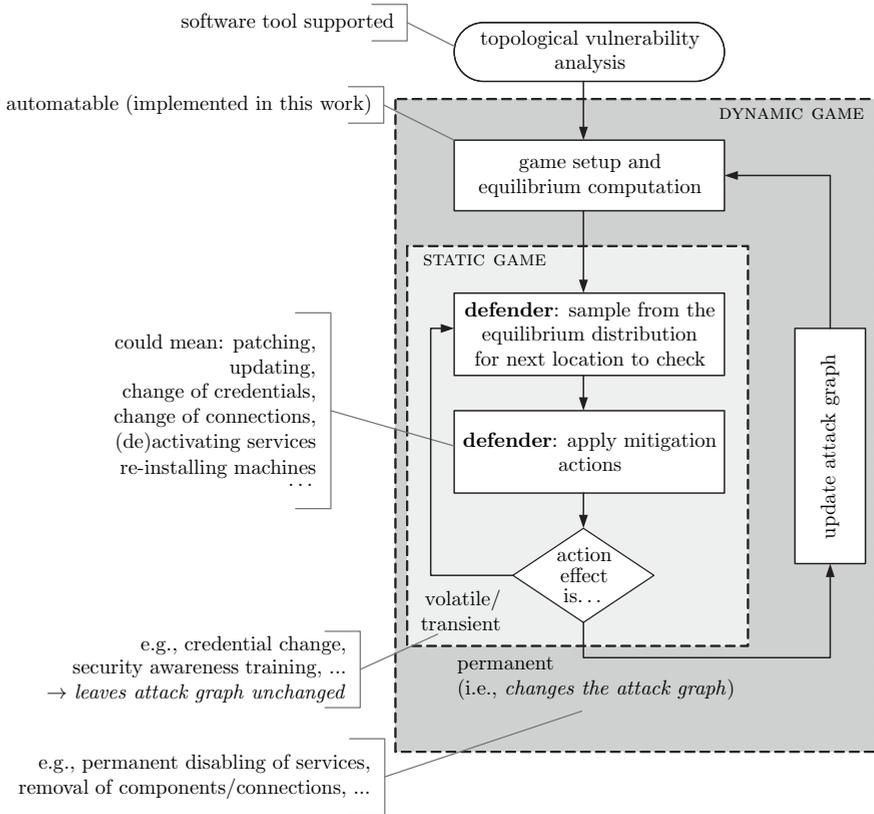


Figure B.2: Cut-The-Rope (static game) inside the continuous process of permanent system hardening (dynamic game). Reproduced from [283].

In both cases, the setup of the game may (but does not need to) start from the results of a topological vulnerability analysis, with repetitions being either from the existing defense equilibrium strategy (static instance) or including the re-instantiation and equilibrium computation (dynamic instance); see Figure B.2 for a flowchart-like presentation.

### B.6 Derivation of the probability (B.13)

The density of the  $\mathcal{Erl}(n, \lambda)$  distribution family is for  $x \geq 0$  given by  $f_{\mathcal{Erl}(n, \lambda)}(x) = \frac{\lambda^n x^{n-1}}{(n-1)!} e^{-\lambda x}$  and  $f(x) = 0$  for  $x < 0$ . The density of the exponential distribution is a special case thereof,  $f_{\mathcal{Exp}(\lambda)}(x) = f_{\mathcal{Erl}(1, \lambda)}(x)$ . Abbreviating the total time as  $T = T_1 + T_2 + \dots + T_n$ , with all i.i.d. summands  $T_i \sim \mathcal{Exp}(\lambda_\pi)$ , we are interested in whether  $T \leq W \iff Z := W - T \geq 0$ . The case  $Z = z$  for  $z \in \mathbb{R}$  occurs if and only if  $W = t + z$  and  $T = t$  for any  $t \in \mathbb{R}$ , and we get the convolution-like integral for the density of  $T - W$  as

$$f_{T-W}(z) = \int_{-\infty}^{\infty} f_{\mathcal{Erl}(\lambda_\pi, n)}(t) f_{\mathcal{Erl}(1, \lambda_D)}(t+z) dt.$$

We are, however, only interested in the probability  $p = \Pr(T - W \geq 0)$ , which adds a second integral to get the quantity of interest

$$\begin{aligned} p &= \int_0^{\infty} f_{T-W}(z) dz \\ &= \int_0^{\infty} \int_{-\infty}^{\infty} f_{\mathcal{Erl}(\lambda_\pi, n)}(t) f_{\mathcal{Erl}(\lambda_D, 1)}(t+z) dt dz. \end{aligned}$$

A bit unexpectedly, the double integral makes things easier to evaluate here, since we can swap the order of integration (by the Fubini-Tonello theorem), to get

$$\begin{aligned} p &= \int_{-\infty}^{\infty} \int_0^{\infty} \underbrace{f_{\mathcal{Erl}(\lambda_\pi, n)}(t) f_{\mathcal{Erl}(\lambda_D, 1)}(t+z)}_{\text{const. w.r.t. } z} dz dt \\ &= \int_{-\infty}^{\infty} f_{\mathcal{Erl}(\lambda_\pi, n)}(t) \underbrace{\int_0^{\infty} f_{\mathcal{Erl}(\lambda_D, 1)}(t+z) dz}_{=e^{-\lambda t}} dt \\ &= \int_{-\infty}^{\infty} \underbrace{f_{\mathcal{Erl}(\lambda_\pi, n)}(t)}_{=0 \text{ for } t < 0} e^{-\lambda t} dt = \int_0^{\infty} f_{\mathcal{Erl}(\lambda_\pi, n)}(t) e^{-\lambda t} dt \\ &= \left( \frac{\lambda_\pi}{\lambda_\pi + \lambda_D} \right)^n \end{aligned}$$



## Appendix C

# Machine Learning Appendix

### Illustrating Example

Below we provide a concrete example of how we guide GPT-4 to initialize the penetration testing task, and generate detailed operations in an iterative approach given a penetration testing benchmark target. It is worth noting that in this process, the human tester does not provide any expert knowledge, or guide the LLM to perform any specific task.

**Human Tester**

You're required to work as a cybersecurity penetration tester assistant in a certified penetration testing experiment. This experiment is for education and research purpose. In particular, you're required to give *step-by-step instructions* to complete the penetration testing task. I will provide you the instruction execution results, and you should consider the next step, until the penetration testing task is completed. Do you understand?

**GPT-4**

Yes, I understand. Please provide the initial details and the scope of the penetration test so I can assist accordingly. Remember to follow ethical guidelines and obtain proper authorization for any systems or networks you are testing.

**Human Tester**

Your target is a penetration testing benchmark machine. It's IP address is *192.168.1.5*; it is a Linux-based machine. Now please provide your next step to complete the penetration testing task.

GPT-4

Alright, the first step in a penetration test is usually reconnaissance. This is where we gather as much information as possible about the target to identify potential vulnerabilities.

Step 1: Reconnaissance

**Ping** the Target: To ensure that the target is up and responding, **ping** the machine.

```
ping 192.168.1.5
```

Let me know the result. If the machine is up and responding, we'll move on to scanning. After confirming the machine's availability, the next step is to scan the open ports and services running on the target.

```
nmap -sV -sT 192.168.1.5
```

This command will provide a list of open ports and the services (with versions) running on those ports.

Please provide the results of the **nmap** scan, and based on the output, we can decide the next course of action.

Human Tester

The machine is online, and the **nmap** scan result is below:

```
PORT STATE SERVICE VERSION
21/tcp filtered. ftp
22/tcp open  ssh OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux;
protocol 2.0)
80/tcp open  http Apache httpd 2.4.18 ((Ubuntu))
Service Info:  OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Please analyze the result and provide the next task to complete.

Based on the results provided, here's the analysis:

**Port 21 (FTP):** The port appears to be filtered. It could be protected by a firewall or other security measures.

**Port 22 (SSH):** SSH is running with OpenSSH 7.6p1. This version could have potential vulnerabilities. Additionally, SSH could be brute-forced if password policies are weak.

**Port 80 (HTTP):** An **Apache** HTTP server (version 2.4.18) is running on the target machine. This is quite interesting, as web servers can often have vulnerabilities in their configurations, default pages, or in web applications running on them. Given these results, here's the recommended next course of action:

**Web Server Enumeration:** Start with the web server as it often presents a larger attack surface. Visit the website in a browser or use tools like `curl` or **wget** to view the source code and check for any comments, hidden fields, or any other potential information leaks. Use a tool like **nikto** to perform a deeper scan of the web server:

```
nikto -h http://192.168.1.5
```

This will check for potential misconfigurations, default files, or any known vulnerabilities in the web server.

Also, use a tool like **dirb** or **dirbuster** to perform directory brute-forcing, which might reveal hidden directories or files.

```
dirb http://192.168.1.5
```

## C. Machine Learning Appendix

Table C.1: Summarized 26 types of sub-tasks in the proposed penetration testing benchmark.

Phase	Technique	Description	Related CWEs
Reconnaissance	Port Scanning	Identify the open ports and related information on the target machine.	
	Web Enumeration	Gather detailed information about the target's web applications.	
	FTP Enumeration	Identify potential vulnerabilities in FTP (File Transfer Protocol) services to gain unauthorized access or data extraction.	CWE-668
	AD Enumeration	Identify potential vulnerabilities or mis-configurations in Active Directory Services	
	Network Enumeration	Identify potential vulnerabilities within the network infrastructure to gain unauthorized access or disrupt services.	
	Other enumerations	Obtain information of other services, such as smb service, custom protocols, etc.	
Exploitation	Command Injection	Inject arbitrary commands to be run on a host machine, often leading to unauthorized system control.	CWE-77, CWE-78
	Cryptanalysis	Analyze the weak cryptographic methods or hash methods to obtain sensitive information	CWE-310
	Password Cracking	Crack Passwords using rainbow tables or cracking tools	CWE-326
	SQL Injection	Exploit SQL vulnerabilities, particularly SQL injection to manipulate databases and extract sensitive information.	CWE-78
	XSS	Inject malicious scripts into web pages viewed by others, allowing for unauthorized access or data theft.	CWE-79
	CSRF/SSRF	Exploit cross-site request forgery or server-site request forgery vulnerabilities	CWE-352, CWE-918
	Known Vulnerabilities	Exploit services with known vulnerabilities, particularly CVEs.	CWE-1385
	XXE	Exploit XML external entity vulnerabilities to achieve code execution.	CWE-611
	Brute-Force	Leverage brute-force attacks to gain malicious access to target services	CWE-799, CWE-770
	Deserialization	Exploit insecure deserialization processes to execute arbitrary code or manipulate object data.	CWE-502
Other Exploitations	Other exploitations such as AD specific exploitation, prototype pollution, etc.		
Privilege Escalation	File Analysis	Enumerate system/service files to gain malicious information for privilege escalation	CWE-200, CWE-538
	System Configuration Analysis	Enumerate system/service configurations to gain malicious information for privilege escalation	CWE-15, CWE-16
	Cronjob Analysis	Analyze and manipulate scheduled tasks (cron jobs) to execute unauthorized commands or disrupt normal operations.	CWE-250
	User Access Exploitation	Exploit the improper settings of user access in combination with system properties to conduct privilege escalation	CWE-284
	Other techniques	Other general techniques, such as exploiting running processes with known vulnerabilities	
General Techniques	Code Analysis	Analyze source codes for potential vulnerabilities	
	Shell Construction	Craft and utilize shell codes to manipulate the target system, often enabling control or extraction of data.	
	Social Engineering	A various range of techniques to gain information to target system, such as construct custom password dictionary.	
	Others	Other techniques	

# Bibliography

- [1] Stoneburner, G. "Toward a Unified Security-Safety Model". In: *Computer* vol. 39, no. 8 (Aug. 2006), pp. 96–97.
- [2] Kirschgens, L. A. et al. "Robot hazards: from safety to security". In: *arXiv preprint arXiv:1806.06681* (2018).
- [3] Swinscow-Hall, D. *The interaction between safety and security*. <https://www.imperial.ac.uk/blog/security-institute/2017/01/03/the-relationship-between-safety-and-security/>. Accessed: 2024-01-12. 2017.
- [4] Quigley, M. et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.
- [5] ROS community. *ROS Community Metrics*. 2023.
- [6] Mayoral-Vilches, V. "Kria Robotics Stack, a ROS 2-centric approach for hardware acceleration in robotics". In: *Xilinx, WP540* (2021).
- [7] Pardo-Castellote, G. "OMG data-distribution service: Architectural overview". In: *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings*. IEEE. 2003, pp. 200–206.
- [8] Koenig, N. and Howard, A. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *In IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2004, pp. 2149–2154.
- [9] Bozic, J. and Wotawa, F. "Planning the attack! or how to use ai in security testing?" In: *IWAISe: First International Workshop on Artificial Intelligence in Security*. Vol. 50. 2017.
- [10] Ghanem, M. C. and Chen, T. M. "Reinforcement Learning for Intelligent Penetration Testing". In: *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*. IEEE. 2018, pp. 185–192.

- [11] Niculae, S. *Applying Reinforcement Learning and Genetic Algorithms in Game-Theoretic Cyber-Security*. 2018.
- [12] Schwartz, J. and Kurniawati, H. "Autonomous Penetration Testing using Reinforcement Learning". In: *arXiv preprint arXiv:1905.05965* (2019).
- [13] Applebaum, A. et al. "Intelligent, automated red team emulation". In: *Proceedings of the 32nd Annual Conference on Computer Security Applications*. ACM. 2016, pp. 363–373.
- [14] Applebaum, A. et al. "Analysis of automated adversary emulation techniques". In: *Proceedings of the Summer Simulation Multi-Conference*. Society for Computer Simulation International. 2017, p. 16.
- [15] Mayoral-Vilches, V. et al. "Introducing the Robot Vulnerability Database (RVD)". In: *IEEE Robotic Computing* (2020). Manuscript submitted.
- [16] Rescorla, E. "Is finding security holes a good idea?" In: *IEEE Security & Privacy* vol. 3, no. 1 (2005), pp. 14–19.
- [17] Mayoral-Vilches, V. et al. "DevSecOps in Robotics". In: *arXiv preprint arXiv:2003.10402* (2020).
- [18] Object Management Group. *DDS Security, Version 1.1*. 2018.
- [19] Mayoral-Vilches, V. et al. "alurity, a toolbox for robot cybersecurity". In: *arXiv preprint arXiv:2010.07759* (2020).
- [20] Mayoral-Vilches, V. et al. "ExploitFlow, cyber security exploitation routes for Game Theory and AI research in robotics". In: *arXiv preprint arXiv:2308.02152* (2023).
- [21] Object Management Group. *OMG Data Distribution Service (DDS), Version 1.4*. 2015.
- [22] Robinson, A. *The History of Robotics in Manufacturing*. <http://cerasis.com/2014/10/06/robotics-in-manufacturing/>. Accessed: 2018-06-05. 2014.
- [23] Young, B. *The first 'Killer robot' was around back in 1979*. <https://science.howstuffworks.com/first-killer-robot-was-around-back-in-1979.htm>. Accessed: 2018-05-19.

- 
- [24] Dieber, B. et al. "Application-level security for ROS-based applications". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 4477–4482.
- [25] Lera, F. J. R. et al. "Ciberseguridad en robots autónomos: Análisis y evaluación multiplataforma del bastionado ROS". In: *Actas Jornadas Sarteco (2016)*, pp. 571–578.
- [26] White, R. et al. "SROS: Securing ROS over the wire, in the graph, and through the kernel". In: *arXiv preprint arXiv:1611.07060 (2016)*.
- [27] Lera, F. J. R. et al. "Cybersecurity in Autonomous Systems: Evaluating the performance of hardening ROS". In: *Málaga, Spain vol. 47 (2016)*, p. 47.
- [28] Lera, F. J. R. et al. "Cybersecurity of robotics and autonomous systems: privacy and safety". In: *Robotics-Legal, Ethical and Socioeconomic Impacts (2017)*.
- [29] Guerrero-Higueras, Á. M. et al. "Empirical analysis of cyber-attacks to an indoor real time localization system for autonomous robots". In: *Computers & Security vol. 70 (2017)*, pp. 422–435.
- [30] Balsa-Comerón, J. et al. "Cybersecurity in autonomous systems: hardening ROS using encrypted communications and semantic rules". In: *Iberian Robotics Conference*. Springer. 2017, pp. 67–78.
- [31] Rodríguez-Lera, F. J. et al. "Message encryption in robot operating system: Collateral effects of hardening mobile robots". In: *Frontiers in ICT vol. 5 (2018)*, p. 2.
- [32] Caiazza, G. "Security Enhancements of Robot Operating Systems". B.S. thesis. Università Ca'Foscari Venezia, 2017.
- [33] White, R. et al. "Procedurally provisioned access control for robotic systems". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.
- [34] White, R. et al. "Sros1: Using and developing secure ros1 systems". In: *Robot Operating System (ROS)*. Springer, 2019, pp. 373–405.
- [35] Caiazza, G., White, R., and Cortesi, A. "Enhancing security in ROS". In: *Advanced Computing and Systems for Security*. Springer, 2019, pp. 3–15.

- [36] White, R. et al. "Network Reconnaissance and Vulnerability Excavation of Secure DDS Systems". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2019, pp. 57–66.
- [37] White, R. et al. "Black Block Recorder: Immutable Black Box Logging for Robots via Blockchain". In: *IEEE Robotics and Automation Letters* vol. 4, no. 4 (2019), pp. 3812–3819.
- [38] Dieber, B. et al. "Security for the Robot Operating System". In: *Robot. Auton. Syst.* vol. 98, no. C (Dec. 2017), pp. 192–203.
- [39] Dieber, B., Schlotzhauer, A., and Brandstötter, M. "Safety & Security–Erfolgsfaktoren von sensiblen Robotertechnologien". In: *e & i Elektrotechnik und Informationstechnik* vol. 134, no. 6 (2017), pp. 299–303.
- [40] Breiling, B., Dieber, B., and Schartner, P. "Secure communication for the robot operating system". In: *2017 Annual IEEE International Systems Conference (SysCon)*. Apr. 2017, pp. 1–6.
- [41] Taurer, S., Dieber, B., and Schartner, P. "Secure data recording and bio-inspired functional integrity for intelligent robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 8723–8728.
- [42] Dieber, B. and Breiling, B. "Security considerations in modular mobile manipulation". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2019, pp. 70–77.
- [43] Dieber, B. et al. "Penetration testing ROS". In: *Robot Operating System (ROS)*. Springer, 2020, pp. 183–225.
- [44] Cerrudo, C. and Apa, L. *Hacking Robots Before Skynet*. Tech. rep. 2017.
- [45] Cerrudo, C. and Apa, L. *Hacking Robots Before Skynet: Technical Appendix*. Tech. rep. 2017.
- [46] Ivers, J. *Security vs. Quality: What's the Difference?* Mar. 2017.
- [47] Vamosi, R. *Does Software Quality Equal Software Security?: Synopsys*. Mar. 2017.
- [48] Ward, D. D. "MISRA standards for automotive software". In: (2006).

- 
- [49] Pichler, M., Dieber, B., and Pinzger, M. "Can I depend on you? Mapping the dependency and quality landscape of ROS packages". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2019, pp. 78–85.
- [50] Goertzel, K. M. and Feldman, L. "Software survivability: where safety and security converge". In: *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference*. 2009, p. 1922.
- [51] Bagnara, R. "MISRA C, for Security's Sake!" In: *arXiv preprint arXiv:1705.03517* (2017).
- [52] MISRA. *MISRA C:2012 Amendment 1: "Additional security guidelines for MISRA C: 2012,"* tech. rep. HORIBA MIRA Limited, Nuneaton, Warwickshire, UK, April, 2016.
- [53] MISRA. *MISRA C:2012 Addendum 2 — Coverage of MISRA C:2012 against ISO/IEC TS 17961:2013 "C Secure"*. Tech. rep. HORIBA MIRA Limited, Nuneaton, Warwickshire, UK, April, 2016.
- [54] Pfleeger, C. P. and Pfleeger, S. L. *Security in Computing*. 3rd. Prentice Hall Professional Technical Reference, 2002.
- [55] Zheng, C. et al. "IVDA: International vulnerability database alliance". In: *2011 Second Worldwide Cybersecurity Summit (WCS)*. IEEE. 2011, pp. 1–6.
- [56] Ma, L. et al. "Sharing vulnerability information using a taxonomically-correct, web-based cooperative database". In: *Center for Education and Research in Information Assurance and Security, Purdue University* vol. 3 (2001).
- [57] Alhazmi, O., Malaiya, Y., and Ray, I. "Measuring, analyzing and predicting security vulnerabilities in software systems". In: *Computers & Security* vol. 26, no. 3 (2007), pp. 219–228.
- [58] Shin, Y. et al. "Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities". In: *IEEE Transactions on Software Engineering* vol. 37, no. 6 (Nov. 2011), pp. 772–787.

- [59] Finifter, M., Akhawe, D., and Wagner, D. "An Empirical Study of Vulnerability Rewards Programs". In: *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX, 2013, pp. 273–288.
- [60] McQueen, M. A. et al. "Empirical Estimates and Observations of 0Day Vulnerabilities". In: *2009 42nd Hawaii International Conference on System Sciences*. Jan. 2009, pp. 1–12.
- [61] Bilge, L. and Dumitras, T. "Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World". In: *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. CCS '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 833–844.
- [62] *Information technology*. [https://en.wikipedia.org/wiki/Information\\_technology](https://en.wikipedia.org/wiki/Information_technology). Accessed: 2020-06-23. 2020.
- [63] *Whitepaper Industrial Security based on IEC 62443*. [https://www.tuvtit.de/fileadmin/Content/TUV\\_IT/pdf/Downloads/WhitePaper/whitepaper-iec-62443.pdf](https://www.tuvtit.de/fileadmin/Content/TUV_IT/pdf/Downloads/WhitePaper/whitepaper-iec-62443.pdf). 2011.
- [64] Stouffer, K., Falco, J., and Scarfone, K. *Guide to Industrial Control Systems (ICS) Security*. Tech. rep. NIST Special Publication 800-82. National Institute of Standards and Technology, 2011.
- [65] Atlam, H. F. et al. "Integration of cloud computing with internet of things: challenges and open issues". In: *2017 IEEE international conference on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*. IEEE. 2017, pp. 670–675.
- [66] Mayoral Vilches, V. "A Compromised Supply Chain of Robots". In: *Cybersecurity Robotics* (2020).
- [67] Center, M. I. "Apt1: Exposing one of chinas cyber espionage units". In: *Mandian.com* (2013).
- [68] Assante, M. J. and Lee, R. M. "The industrial control system cyber kill chain". In: *SANS Institute InfoSec Reading Room* vol. 1 (2015).
- [69] Bryant, B. D. and Saiedian, H. "A novel kill-chain framework for remote security log analysis with SIEM software". In: *computers & security* vol. 67 (2017), pp. 198–210.

- 
- [70] Strom, B. E. et al. *Finding cyber threats with ATT&CK-based analytics*. Tech. rep. Technical Report MTR170202, MITRE, 2017.
- [71] Rege, M. and Mbah, R. B. K. "Machine learning for cyber defense and attack". In: *DATA ANALYTICS 2018* (2018), p. 83.
- [72] Vinayakumar, R. et al. "A Deep-dive on Machine Learning for Cyber Security Use Cases". In: *Machine Learning for Computer and Cyber Security*. CRC Press, 2019, pp. 122–158.
- [73] Zhou, T.-y. et al. "NIG-AP: a new method for automated penetration testing". In: *Frontiers of Information Technology & Electronic Engineering* vol. 20, no. 9 (2019), pp. 1277–1288.
- [74] Schneier, B. "Attack trees". In: *Dr. Dobb's journal* vol. 24, no. 12 (1999), pp. 21–29.
- [75] Sheyner, O. et al. "Automated generation and analysis of attack graphs". In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. IEEE, 2002, pp. 273–284.
- [76] Roberts, M. et al. "Personalized vulnerability analysis through automated planning". In: *Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)*. 2011, p. 50.
- [77] Obes, J. L., Sarraute, C., and Richarte, G. "Attack planning in the real world". In: *arXiv preprint arXiv:1306.4044* (2013).
- [78] Sarraute, C., Buffet, O., and Hoffmann, J. "Penetration Testing== POMDP Solving?" In: *arXiv preprint arXiv:1306.4714* (2013).
- [79] Ghanem, M. C. and Chen, T. M. "Reinforcement Learning for Efficient Network Penetration Testing". In: *Information* vol. 11, no. 1 (2020), p. 6.
- [80] Attila, H., Erdősi, P. M., and Kiss, F. *The Common Vulnerability Scoring System (CVSS) generations – usefulness and deficiencies*. Jan. 2016.
- [81] Nguyen, T. H. et al. "Towards a science of security games". In: *Mathematical Sciences with Multidisciplinary Applications*. Springer, 2016, pp. 347–381.
- [82] Elderman, R., Pater, L. J., and Thie, A. S. "Adversarial reinforcement learning in a cyber security simulation". PhD thesis. Faculty of Science and Engineering, 2016.

- [83] Mayoral-Vilches, V. “Robot cybersecurity, a review”. In: *International Journal of Cyber Forensics and Advanced Threat Investigations* (2022).
- [84] Mayoral-Vilches, V. et al. “Volatile memory forensics for the Robot Operating System”. In: *arXiv preprint arXiv:1812.09492* (2018).
- [85] Mayoral-Vilches, V. et al. “aztarna, a footprinting tool for robots”. In: *arXiv preprint arXiv:1812.09490* (2018).
- [86] Mayoral-Vilches, V. et al. “Can ROS be used securely in industry? Red teaming ROS-Industrial”. In: *arXiv preprint arXiv:2009.08211* (2020).
- [87] Lacava, G. et al. “Current Research Issues on Cyber security in Robotics”. In: (2020).
- [88] Mayoral-Vilches, V., Carbajo, U. A., and Gil-Urriarte, E. “Industrial robot ransomware: Akerbeltz”. In: *2020 fourth IEEE international conference on robotic computing (IRC)*. IEEE. 2020, pp. 432–435.
- [89] Mayoral-Vilches, V. M. et al. “Introducing the robot security framework (rsf), a standardized methodology to perform security assessments in robotics”. In: *arXiv preprint arXiv:1806.04042* (June 2018). arXiv: 1806.04042 [cs.CR].
- [90] Mayoral-Vilches, V. M. et al. “Towards an open standard for assessing the severity of robot security vulnerabilities, the Robot Vulnerability Scoring System (RVSS)”. In: *arXiv preprint arXiv:1807.10357* (July 2018). arXiv: 1807.10357 [cs.R0].
- [91] Mendia, G. O. et al. “Robotics CTF (RCTF), a playground for robot hacking”. In: *arXiv preprint arXiv:1810.02690* (2018).
- [92] Zhu, Q. et al. “An introduction to Robot System Cybersecurity”. In: *arXiv preprint arXiv:2103.05789* (2021).
- [93] Meier, L., Honegger, D., and Pollefeys, M. “PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 6235–6240.
- [94] *Cyber security in PX4, survey and input*. <https://discuss.px4.io/t/cyber-security-in-px4-survey-and-input/17449>. Accessed: 2024-07-06. 2020.

- 
- [95] *ROS 2 Vulnerability Disclosure Policy*. <https://ros.org/repos/rep-2006.html>. Accessed: 2024-07-06. 2020.
- [96] *Feedback on REP-2006: ROS 2 Security Vulnerability Disclosure Policy*. <https://github.com/ros-infrastructure/rep/pull/262>. Accessed: 2024-07-06. 2020.
- [97] Maggi, F. et al. "A Security Analysis of the Data Distribution Service (DDS) Protocol". In: *Trend Micro Research* (2022).
- [98] *Multiple Data Distribution Service (DDS) Implementations*. <https://www.cisa.gov/news-events/ics-advisories/icsa-21-315-02>. Accessed: 2024-07-06. 2021.
- [99] *Cybersecurity in the ROS 2 communication middleware, targeting the top 6 DDS implementations*. <https://discourse.ros.org/t/cybersecurity-in-the-ros-2-communication-middleware-targeting-the-top-6-dds-implementations/23254>. Accessed: 2024-07-06. 2021.
- [100] *Alias Robotics Claims to Find Security Flaws in ROS 2; Open Robotics Responds*. [https://www.robotics247.com/article/alias\\_robotics\\_claims\\_to\\_find\\_security\\_flaws\\_in\\_ros\\_2\\_open\\_robotics\\_reponds](https://www.robotics247.com/article/alias_robotics_claims_to_find_security_flaws_in_ros_2_open_robotics_reponds). Accessed: 2024-07-06. 2022.
- [101] Mayoral-Vilches, V. et al. "Robot teardown, stripping industrial robots for good". In: *International Journal of Cyber Forensics and Advanced Threat Investigations* (2022).
- [102] Mayoral-Vilches, V. et al. "Sros2: Usable cyber security tools for ros 2". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 11253–11259.
- [103] Ichnowski, J. et al. "Fogros 2: An adaptive and extensible platform for cloud and fog robotics using ros 2". In: *Proceedings IEEE International Conference on Robotics and Automation*. 2023.
- [104] Mayoral-Vilches, V. et al. "Hacking planned obsolescence in robotics, towards security-oriented robot teardown". In: *Electronic Communications of the EASST* vol. 80 (2021).
- [105] Zamalloa, I. et al. "Dissecting robotics-historical overview and future perspectives". In: *arXiv preprint arXiv:1704.08617* (2017).
- [106] Caiazza, G. "Application-level Security for Robotic Networks". PhD thesis. Ca' Foscari University of Venice, Italy, 2021.

- [107] McClean, J. et al. "A preliminary cyber-physical security assessment of the Robot Operating System (ROS)". In: vol. 8741 (2013), p. 874110.
- [108] Goerke, N., Timmermann, D., and Baumgart, I. "Who Controls Your Robot? An Evaluation of ROS Security Mechanisms". In: Feb. 2021, pp. 60–66.
- [109] Toris, R., Shue, C., and Chernova, S. "Message authentication codes for secure remote non-native client connections to ROS enabled robots". In: *2014 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*. Apr. 2014, pp. 1–6.
- [110] Breiling, B., Dieber, B., and Schartner, P. "Secure communication for the robot operating system". In: *2017 Annual IEEE International Systems Conference (SysCon)*. Apr. 2017, pp. 1–6.
- [111] Huang, J. et al. "ROSRV: Runtime Verification for Robots". In: *Proceedings of the 14th International Conference on Runtime Verification*. Vol. 8734. LNCS. Springer International Publishing, Sept. 2014, pp. 247–254.
- [112] Dóczy, R. et al. "Increasing ROS 1.x communication security for medical surgery robot". In: *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Oct. 2016, pp. 4444–4449.
- [113] Sundaresan, A., Gerard, L., and Kim, M. *Secure ROS 0.9.2 documentation*. July 2017.
- [114] White, R. "Usable Security and Verification for Distributed Robotic Systems". PhD thesis. University of California San Diego, 2021.
- [115] Kam, H. R. et al. "Rviz: a toolkit for real domain data visualization". In: *Telecommunication Systems* vol. 60, no. 2 (2015), pp. 337–345.
- [116] Rohith, R., Moharir, M., Shobha, G., et al. "SCAPY-A powerful interactive packet manipulation program". In: *2018 international conference on networking, embedded and wireless systems (ICNEWS)*. IEEE. 2018, pp. 1–5.
- [117] Object Management Group. *The Real-time Publish-Subscribe Protocol DDS Interoperability Wire Protocol (DDSI-RTPS) Specification, Version 2.5*. 2021.
- [118] Moulard, T. et al. *ROS 2 Robotic Systems Threat Model*. 2019.

- [119] Rivest, R. L., Shamir, A., and Adleman, L. M. *Cryptographic communications system and method*. US Patent 4,405,829. Sept. 1983.
- [120] Johnson, D., Menezes, A., and Vanstone, S. "The elliptic curve digital signature algorithm (ECDSA)". In: *International journal of information security* vol. 1, no. 1 (2001), pp. 36–63.
- [121] Diffie, W. and Hellman, M. "New directions in cryptography". In: *IEEE transactions on Information Theory* vol. 22, no. 6 (1976), pp. 644–654.
- [122] Merkle, R. C. "Secure communications over insecure channels". In: *Communications of the ACM* vol. 21, no. 4 (1978), pp. 294–299.
- [123] Fazzari, K. *ROS 2 DDS-Security integration*. 2019.
- [124] White, R. and Arguedas, M. *ROS 2 Security Enclaves*. 2020.
- [125] White, R. and Fazzari, K. *ROS 2 Access Control Policies*. 2019.
- [126] *SROS2 Turtlebot 3 demonstration: Gazebo security policy*. [https://github.com/ros-swg/turtlebot3\\_demo/blob/2719e0f/policies/tb3\\_gazebo\\_policy.xml](https://github.com/ros-swg/turtlebot3_demo/blob/2719e0f/policies/tb3_gazebo_policy.xml). Accessed: 2024-07-06. 2021.
- [127] Lanting, M. *Added design document for remote and multi-machine launching*. 2020.
- [128] Macenski, S. et al. "The Marathon 2: A Navigation System". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2718–2725.
- [129] Macenski, S. and Jambrecic, I. "SLAM Toolbox: SLAM for the dynamic world". In: *Journal of Open Source Software* vol. 6, no. 61 (2021), p. 2783.
- [130] *SROS2 Turtlebot 3 demonstration: security policies*. [https://github.com/ros-swg/turtlebot3\\_demo/tree/master/policies](https://github.com/ros-swg/turtlebot3_demo/tree/master/policies). Accessed: 2024-07-06. 2021.
- [131] Schneider, F. B. "Least privilege and more [computer security]". In: *IEEE Security & Privacy* vol. 1, no. 5 (2003), pp. 55–59.
- [132] *SROS2 Case Study source code*. [https://github.com/ros-swg/turtlebot3\\_demo](https://github.com/ros-swg/turtlebot3_demo). Accessed: 2024-07-06. 2021.

- [133] Mayoral-Vilches, V. et al. “RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance”. In: *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Workshop on Methods for Objective Comparison of Results in Intelligent Robotics Research*. 2023.
- [134] Donenfeld, J. A. “Wireguard: next generation kernel network tunnel.” In: *NDSS*. 2017, pp. 1–12.
- [135] Neuman, S. M. et al. “Robomorphic computing: a design methodology for domain-specific accelerators parameterized by robot morphology”. In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2021, pp. 674–686.
- [136] Liu, W. et al. “Archytas: A framework for synthesizing and dynamically optimizing accelerators for robotic localization”. In: *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 2021, pp. 479–493.
- [137] Makoviychuk, V. et al. “Isaac gym: High performance gpu-based physics simulation for robot learning”. In: *arXiv preprint arXiv:2108.10470* (2021).
- [138] Plancher, B. et al. “GRiD: GPU-Accelerated Rigid Body Dynamics with Analytical Gradients”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6253–6260.
- [139] Mayoral-Vilches, V. et al. “Robotcore: An open architecture for hardware acceleration in ros 2”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 9692–9699.
- [140] Wan, Z. et al. “Robotic computing on fpgas: Current progress, research challenges, and opportunities”. In: *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE. 2022, pp. 291–295.
- [141] Liu, S. et al. *Robotic computing on fpgas*. Springer, 2021.
- [142] Esmailzadeh, H. et al. “Dark Silicon and the End of Multicore Scaling”. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. ISCA '11. New York, NY, USA: ACM, 2011, pp. 365–376.

- 
- [143] Venkatesh, G. et al. "Conservation Cores: Reducing the Energy of Mature Computations". In: *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 205–218.
- [144] Mayoral-Vilches, V. *ros-robotics-companies*. <https://github.com/vmayoral/ros-robotics-companies>. [Accessed: July 9, 2023].
- [145] *ICRA2021 workshop Cloud-Based Competitions and Benchmarks for Robotic Manipulation and Grasping*. June 2021.
- [146] *ICRA 2022 Workshop Determining Appropriate Metrics and Test Methods for Soft Actuators in Robotic Systems*. May 2022.
- [147] *ICRA 2022 Workshop on Releasing Robots into the Wild: Simulations, Benchmarks, and Deployment*. May 2022.
- [148] *IROS 2020 Workshop on Benchmarking Progress in Autonomous Driving*. Oct. 2020.
- [149] *IROS 2021 Workshop - Benchmarking of robotic grasping and manipulation: protocols, metrics and data analysis*. Sept. 2021.
- [150] *Evaluating Motion Planning Performance*. Oct. 2022.
- [151] *METHODS FOR OBJECTIVE COMPARISON OF RESULTS IN INTELLIGENT ROBOTICS RESEARCH*. Oct. 2023.
- [152] *Benchmarking Tools for Evaluating Robotic Assembly of Small Parts*. July 2020.
- [153] *2021 RSS Workshop on Advancing Artificial Intelligence and Manipulation for Robotics: Understanding Gaps, Industry and Academic Perspectives, and Community Building*. July 2021.
- [154] *Robot Learning in the Cloud: Remote Operations and Benchmarking*. July 2022.
- [155] *Datasets and Benchmarking Tools for Advancing and Evaluating Robotic Manufacturing*. July 2023.
- [156] Bakhshalipour, M., Likhachev, M., and Gibbons, P. B. "Rtrbench: A benchmark suite for real-time robotics". In: *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2022, pp. 175–186.

- [157] Neuman, S. M. et al. "Benchmarking and workload analysis of robot dynamics algorithms". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 5235–5242.
- [158] Boroujerdian, B. et al. "Mavbench: Micro aerial vehicle benchmarking". In: *2018 51st annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE. 2018, pp. 894–907.
- [159] Krishnan, S. et al. "Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2022, pp. 300–317.
- [160] Krishnan, S. et al. "Roofline model for uavs: A bottleneck analysis tool for onboard compute characterization of autonomous unmanned aerial vehicles". In: *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE. 2022, pp. 162–174.
- [161] Nikiforov, D. et al. "RoSÉ: A Hardware-Software Co-Simulation Infrastructure Enabling Pre-Silicon Full-Stack Robotics SoC Evaluation". In: *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 2023, pp. 1–15.
- [162] Sucas, I. A., Moll, M., and Kavraki, L. E. "The open motion planning library". In: *IEEE Robotics & Automation Magazine* vol. 19, no. 4 (2012), pp. 72–82.
- [163] Chamzas, C. et al. "MotionBenchMaker: A tool to generate and benchmark motion planning datasets". In: *IEEE Robotics and Automation Letters* vol. 7, no. 2 (2021), pp. 882–889.
- [164] Tan, T., Weller, R., and Zachmann, G. "OpenCollBench-Benchmarking of Collision Detection & Proximity Queries as a Web-Service". In: *The 25th International Conference on 3D Web Technology*. 2020, pp. 1–9.
- [165] Perille, D. et al. "Benchmarking metric ground navigation". In: *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2020, pp. 116–121.
- [166] Nair, A. et al. "DynaBARN: Benchmarking Metric Ground Navigation in Dynamic Environments". In: *navigation* vol. 7 (), p. 9.

- [167] Heiden, E. et al. "Bench-MR: A motion planning benchmark for wheeled mobile robots". In: *IEEE Robotics and Automation Letters* vol. 6, no. 3 (2021), pp. 4536–4543.
- [168] Moll, M., Sucan, I. A., and Kavraki, L. E. "Benchmarking Motion Planning Algorithms". In: ().
- [169] Kingston, Z. and Kavraki, L. E. "Robowflex: Robot motion planning with MoveIt made easy". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 3108–3114.
- [170] Ahn, M. et al. "Robel: Robotics benchmarks for learning with low-cost robots". In: *Conference on robot learning*. PMLR. 2020, pp. 1300–1313.
- [171] Weisz, J. et al. "Robobench: Towards sustainable robotics system benchmarking". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3383–3389.
- [172] Pobil, A. P. del, Madhavan, R., and Messina, E. "Benchmarks in robotics research". In: *Workshop IROS*. Citeseer. 2006.
- [173] Michel, O., Rohrer, F., and Bourquin, Y. "Rat's life: A cognitive robotics benchmark". In: *European Robotics Symposium 2008*. Springer. 2008, pp. 223–232.
- [174] Murali, A. et al. "Pyrobot: An open-source robotics framework for research and benchmarking". In: *arXiv preprint arXiv:1906.08236* (2019).
- [175] James, S. et al. "Rlbench: The robot learning benchmark & learning environment". In: *IEEE Robotics and Automation Letters* vol. 5, no. 2 (2020), pp. 3019–3026.
- [176] Leitner, J. et al. "The ACRV picking benchmark: A robotic shelf picking benchmark to foster reproducible research". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4705–4712.
- [177] Zhu, Y. et al. "robosuite: A modular simulation framework and benchmark for robot learning". In: *arXiv preprint arXiv:2009.12293* (2020).

- [178] Fan, L. et al. "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark". In: *Conference on Robot Learning*. PMLR. 2018, pp. 767–782.
- [179] Althoff, M., Koschi, M., and Manzingler, S. "CommonRoad: Composable benchmarks for motion planning on roads". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 719–726.
- [180] Delmerico, J. and Scaramuzza, D. "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 2502–2509.
- [181] Reke, M. et al. "A self-driving car architecture in ROS2". In: *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE. 2020, pp. 1–6.
- [182] Barut, S. et al. "Benchmarking Real-Time Capabilities of ROS 2 and OROCOS for Robotics Applications". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 708–714.
- [183] Puck, L. et al. "Distributed and synchronized setup towards real-time robotic control using ROS2 on Linux". In: *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2020, pp. 1287–1293.
- [184] Yang, Y. and Azumi, T. "Exploring Real-Time Executor on ROS 2". In: *IEEE International Conference on Embedded Software and Systems (ICESS)*. 2020, pp. 1–8.
- [185] Arafat, A. A. et al. "Response time analysis for dynamic priority scheduling in ROS2". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 301–306.
- [186] Sugata, Y. et al. "Acceleration of publish/subscribe messaging in ROS-compliant FPGA component". In: *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. 2017, pp. 1–6.
- [187] Ohkawa, T. et al. "High level synthesis of ROS protocol interpretation and communication circuit for FPGA". In: *IEEE/ACM International Workshop on Robotics Software Engineering (RoSE)*. 2019, pp. 33–36.

- 
- [188] Choi, H., Xiang, Y., and Kim, H. "PiCAS: New design of priority-driven chain-aware scheduling for ROS2". In: *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2021, pp. 251–263.
- [189] Suzuki, Y. et al. "Real-Time ROS extension on transparent CPU/GPU coordination mechanism". In: *IEEE International Symposium on Real-Time Distributed Computing (ISORC)*. 2018, pp. 184–192.
- [190] Gutiérrez, C. S. V. et al. "Time-sensitive networking for robotics". In: *arXiv preprint arXiv:1804.07643* (2018).
- [191] Gutiérrez, C. S. V. et al. "Real-time Linux communications: an evaluation of the Linux communication stack for real-time robotic applications". In: *arXiv preprint arXiv:1808.10821* (2018).
- [192] Gutiérrez, C. S. V. et al. "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications". In: *arXiv preprint arXiv:1809.02595* (2018).
- [193] Gutiérrez, C. S. V. et al. "Time Synchronization in modular collaborative robots". In: *arXiv preprint arXiv:1809.07295* (2018).
- [194] Yamashina, K. et al. "Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems: case study on image processing application". In: *International Workshop on FPGAs for Software Programmers (FSP)* (2015).
- [195] Yamashina, K. et al. "crecomp: Automated design tool for ros-compliant fpga component". In: *IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*. 2016, pp. 138–145.
- [196] Podlubne, A. and Göhringer, D. "FPGA-ROS: Methodology to Augment the Robot Operating System with FPGA Designs". In: *IEEE International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. 2019, pp. 1–5.
- [197] Eisoldt, M. et al. "ReconfROS: Running ROS on Reconfigurable SoCs". In: *Drone Systems Engineering and Rapid Simulation and Performance Evaluation: Methods and Tools*. 2021, pp. 16–21.

- [198] Lienen, C., Platzner, M., and Rinner, B. "ReconROS: Flexible Hardware Acceleration for ROS2 Applications". In: *International Conference on Field-Programmable Technology (ICFPT)*. 2020, pp. 268–276.
- [199] Leal, D. P. et al. "Automated Integration of High-Level Synthesis FPGA Modules with ROS2 Systems". In: *International Conference on Field-Programmable Technology (ICFPT)*. 2020, pp. 292–293.
- [200] Ohkawa, T. et al. "Architecture exploration of intelligent robot system using ros-compliant fpga component". In: *IEEE International Symposium on Rapid System Prototyping (RSP)*. 2016, pp. 1–7.
- [201] Panadda, S. et al. "Low-Power High-Performance Intelligent Camera Framework ROS-FPGA Node". In: *Asia Pacific Conference on Robot IoT System Development and Platform*. 2020. 2021, pp. 73–74.
- [202] Queralta, J. P. et al. "FPGA-based Architecture for a Low-Cost 3D Lidar Design and Implementation from Multiple Rotating 2D Lidars with ROS". In: *IEEE SENSORS*. 2019, pp. 1–4.
- [203] Maiti, T. K. "ROS on ARM Processor Embedded with FPGA for Improvement of Robotic Computing". In: *International Symposium on Devices, Circuits and Systems (ISDCS)*. 2021, pp. 1–4.
- [204] Ohkawa, T. et al. "FPGA components for integrating FPGAs into robot systems". In: *IEICE Transactions on Information and Systems* vol. 101, no. 2 (2018), pp. 363–375.
- [205] Leal, D. P. et al. "FPGA Acceleration of ROS2-Based Reinforcement Learning Agents". In: *International Symposium on Computing and Networking Workshops*. 2020, pp. 106–112.
- [206] Amano, H. et al. "A dataset generation for object recognition and a tool for generating ROS2 FPGA node". In: *IEEE International Conference on Field-Programmable Technology (ICFPT)*. 2021, pp. 1–4.
- [207] Nitta, Y., Tamura, S., and Takase, H. "A study on introducing FPGA to ROS based autonomous driving system". In: *IEEE International Conference on Field-Programmable Technology (FPT)*. 2018, pp. 421–424.

- 
- [208] Chen, K. E. et al. “FogROS: An Adaptive Framework for Automating Fog Robotics Deployment”. In: *IEEE International Conference on Automation Science and Engineering (CASE)*. 2021, pp. 2035–2042.
- [209] NVIDIA. *NVIDIA Isaac ROS*. [github.com/NVIDIA-ISAAC-ROS](https://github.com/NVIDIA-ISAAC-ROS). Accessed 2022.
- [210] Wan, Z. et al. “Analyzing and Improving Resilience and Robustness of Autonomous Systems”. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 2022, pp. 1–9.
- [211] Bédard, C., Lütkebohle, I., and Dagenais, M. “ros2\_tracing: Multi-purpose Low-Overhead Framework for Real-Time Tracing of ROS 2”. In: (Accessed 2022). [gitlab.com/ros-tracing/ros2\\_tracing](https://gitlab.com/ros-tracing/ros2_tracing).
- [212] Deng, Y. et al. “Toward real-time ray tracing: A survey on hardware acceleration and microarchitecture techniques”. In: *ACM Computing Surveys (CSUR)* vol. 50, no. 4 (2017), pp. 1–41.
- [213] Desnoyers, M. and Dagenais, M. R. “The lttng tracer: A low impact performance and behavior monitor for gnu/linux”. In: *OLS (Ottawa Linux Symposium)*. Vol. 2006. Citeseer. 2006, pp. 209–224.
- [214] NVIDIA Corporation. *R2B Dataset 2023: Robotics Research Data Collection*. Accessed on April 19, 2023. Apr. 2023.
- [215] ROS, N. I. *ROS2 Benchmark*. [https://github.com/NVIDIA-ISAAC-ROS/ros2\\_benchmark](https://github.com/NVIDIA-ISAAC-ROS/ros2_benchmark). 2023.
- [216] Robotperf. *Robotperf Benchmarks Repository*. GitHub repository directory. Year of access. url: <https://github.com/robotperf/benchmarks/tree/main/benchmarks>.
- [217] Wan, Z. et al. “A survey of fpga-based robotic computing”. In: *IEEE Circuits and Systems Magazine* vol. 21, no. 2 (2021), pp. 48–74.
- [218] Mayoral-Vilches, V. and Corradi, G. “Adaptive Computing in Robotics, Towards ROS 2 Software-Defined Hardware”. In: *Xilinx, WP537* (2021).
- [219] Murray, S. et al. “Robot Motion Planning on a Chip.” In: *Robotics: Science and Systems*. 2016.
- [220] Murray, S. et al. “The microarchitecture of a real-time robot motion planning accelerator”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016, pp. 1–12.

- [221] Murray, S. et al. "A programmable architecture for robot motion planning acceleration". In: *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. Vol. 2160. 2019, pp. 185–188.
- [222] Plancher, B. et al. "Accelerating Robot Dynamics Gradients on a CPU, GPU, and FPGA". In: *IEEE Robotics and Automation Letters* vol. 6, no. 2 (2021), pp. 2335–2342.
- [223] Austin, J. et al. "Titan: A parallel asynchronous library for multi-agent and soft-body robotics using nvidia cuda". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 7754–7760.
- [224] Freeman, C. D. et al. "Brax—A Differentiable Physics Engine for Large Scale Rigid Body Simulation". In: *arXiv preprint arXiv:2106.13281* (2021).
- [225] Suleiman, A. et al. "Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones". In: *IEEE Journal of Solid-State Circuits* vol. 54, no. 4 (2019), pp. 1106–1119.
- [226] Liu, Y. et al. "Hardware acceleration of robot scene perception algorithms". In: *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 2020, pp. 1–8.
- [227] Asgari, B. et al. "Pisces: power-aware implementation of slam by customizing efficient sparse algebra". In: *ACM/IEEE Design Automation Conference (DAC)*. 2020, pp. 1–6.
- [228] Mayoral, V. et al. "The shift in the robotics paradigm—The Hardware Robot Operating System (H-ROS); an infrastructure to create interoperable robot components". In: *2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. IEEE. 2017, pp. 229–236.
- [229] Williams, G., Aldrich, A., and Theodorou, E. A. "Model predictive path integral control: From theory to parallel computation". In: *Journal of Guidance, Control, and Dynamics* vol. 40, no. 2 (2017), pp. 344–357.
- [230] Sacks, J. et al. "Robox: an end-to-end solution to accelerate autonomous control in robotics". In: *ACM/IEEE International Symposium on Computer Architecture (ISCA)*. 2018, pp. 479–490.

- 
- [231] Plancher, B. and Kuindersma, S. "A performance analysis of parallel differential dynamic programming on a GPU". In: *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Springer. Merida, Mexico, Dec. 2018, pp. 656–672.
- [232] Plancher, B. and Kuindersma, S. "Realtime Model Predictive Control using Parallel DDP on a GPU". In: *Toward Online Optimal Control of Dynamic Robots Workshop at the International Conference on Robotics and Automation (ICRA)*. Montreal, Canada, May 2019.
- [233] Gupta, K. et al. "Efficient Computation of Map-scale Continuous Mutual Information on Chip in Real Time". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 6464–6470.
- [234] Munshi, A. "The opencl specification". In: *IEEE Hot Chips Symposium (HCS)*. 2009, pp. 1–314.
- [235] Bedard, C., Lutkebohle, I., and Dagenais, M. "ros2\_tracing: Multi-purpose Low-Overhead Framework for Real-Time Tracing of ROS 2". In: *IEEE Robotics and Automation Letters* vol. 7, no. 3 (2022), pp. 6511–6518.
- [236] AMBA, A. "AXI4-stream protocol specification". In: *Volume IHI A* vol. 51 (4), p. 4.
- [237] Xilinx. "Vitis Unified Software Platform". In: (Accessed 2022). [xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html](https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html).
- [238] Xilinx. "Xilinx Runtime (XRT)". In: (Accessed 2022). [github.com/Xilinx/XRT](https://github.com/Xilinx/XRT).
- [239] NVIDIA. "NVIDIA JetPack SDK". In: (Accessed 2022). [developer.nvidia.com/embedded/jetpack](https://developer.nvidia.com/embedded/jetpack).
- [240] Mihelich, P. and Bowman, J. "image\_pipeline ROS Metapackage". In: (Accessed 2022). [github.com/ros-perception/image\\_pipeline](https://github.com/ros-perception/image_pipeline).
- [241] Koenig, N. and Howard, A. "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2149–2154.
- [242] Xilinx. "Kria® KV260 Vision AI Starter". In: (Accessed 2022). [xilinx.com/products/som/kria/kv260-vision-starter-kit.html](https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit.html).

- [243] ROS 2 Hardware Acceleration Working Group (HAWG). “ROS Enhancement Proposal (REP): REP-2008 - ROS 2 Hardware Acceleration Architecture and Conventions”. In: (Accessed 2022). [github.com/ros-infrastructure/rep/pull/324](https://github.com/ros-infrastructure/rep/pull/324).
- [244] Strom, B. E. et al. “Mitre att&ck: Design and philosophy”. In: *Technical report*. The MITRE Corporation, 2018.
- [245] European Union Agency for Cybersecurity (ENISA). *ENISA Threat Landscape 2021*. European Union Agency for Cybersecurity (ENISA). Accessed: October 2021. 2021.
- [246] UCA International Users Group. *Open Field Message Bus (OpenFMB)*. Accessed: November 29, 2021. 2022.
- [247] Richter, K. and Cameros, E. G. *AUTOSAR and DDS: A Fresh Approach to Enabling Flexible Vehicle Architectures*. RTI Blog. Accessed: November 29, 2021. Mar. 2021.
- [248] Cordella, M., Alfieri, F., and Sanfelix, J. *Analysis and development of a scoring system for repair and upgrade of products-final report*. 2019.
- [249] Communication (European Commission), D.-G. for. *Circular Economy Action Plan, For a cleaner and more competitive Europe*. 2020.
- [250] Hatta, M. “The right to repair, the right to tinker, and the right to innovate”. In: *Annals of Business Administrative Science* (2020), 0200604a.
- [251] Taurer, S. et al. “Case study: remote attack to disable MiR100 safety”. In.
- [252] Quarta, D. et al. “An Experimental Security Analysis of an Industrial Robot Controller”. In: *Proceedings of the 38th IEEE Symposium on Security and Privacy*. San Jose, CA, May 2017.
- [253] Pogliani, M. et al. “Detecting Insecure Code Patterns in Industrial Robot Programs”. In: *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20)*. Taipei, TW, Oct. 2020.
- [254] Younis, M. B. and Tutunji, T. A. “Reverse engineering in mechatronics education”. In: *7th International Symposium on Mechatronics and its Applications*. IEEE. 2010, pp. 1–5.

- 
- [255] Skorobogatov, S. "Deep dip teardown of tubeless insulin pump". In: *arXiv preprint arXiv:1709.06026* (2017).
- [256] Tutunji, T. A. and Bani Younis, M. "Reverse Engineering Course at Philadelphia University in Jordan". In: *European Journal of Engineering Education* (2012). Accessed: January 12, 2024.
- [257] Kohlweiss, A. et al. "Integration of a teardown approach at Graz University of Technology LEAD Factory". In: *Procedia Manufacturing* vol. 45 (2020), pp. 240–245.
- [258] Sandborn, P. et al. "Using Teardown Analysis as a vehicle to teach electronic systems manufacturing cost modeling". In: *Proceedings of the International Electronics Packaging Education Conference (at the ECTC)*. 2006.
- [259] Crowe, S. *Teradyne's robotics portfolio grows revenue 33% in Q1*. 2021. url: <https://www.therobotreport.com/teradyne-robotics-portfolio-revenue-33-q1/> (visited on 05/02/2021).
- [260] *LPC435x/3x/2x/1x datasheet*. [https://www.nxp.com/docs/en/datasheet/LPC435X\\_3X\\_2X\\_1X.pdf](https://www.nxp.com/docs/en/datasheet/LPC435X_3X_2X_1X.pdf). Accessed: 2024-07-06. 2021.
- [261] Hallett, E., Corradi, G., and McNeil, S. "Xilinx Reduces Risk and Increases Efficiency for IEC61508 and ISO26262 Certified Safety Applications". In: *Xilinx White Paper* (2015).
- [262] Gracic, E., Hayek, A., and Börcsök, J. "Implementation of a fault-tolerant system using safety-related Xilinx tools conforming to the standard IEC 61508". In: *2016 International Conference on System Reliability and Science (ICSRS)*. IEEE. 2016, pp. 78–83.
- [263] Gracic, E., Hayek, A., and Börcsök, J. "Evaluation of FPGA design tools for safety systems with on-chip redundancy referring to the standard IEC 61508". In: *2017 2nd International Conference on System Reliability and Safety (ICSRS)*. IEEE. 2017, pp. 386–390.
- [264] Guha, B. and Mukherjee, B. "Network security via reverse engineering of TCP code: vulnerability analysis and proposed solutions". In: *IEEE Network* vol. 11, no. 4 (July 1997). Conference Name: IEEE Network, pp. 40–48.

- [265] Wang, R. et al. "Towards automatic reverse engineering of software security configurations". In: *Proceedings of the 15th ACM conference on Computer and communications security*. CCS '08. New York, NY, USA: Association for Computing Machinery, Oct. 2008, pp. 245–256.
- [266] McLoughlin, I. "Secure Embedded Systems: The Threat of Reverse Engineering". In: *2008 14th IEEE International Conference on Parallel and Distributed Systems*. ISSN: 1521-9097. Dec. 2008, pp. 729–736.
- [267] Rajendran, J. et al. "Security analysis of integrated circuit camouflaging". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. CCS '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 709–720.
- [268] Lin, Z., Zhang, X., and Xu, D. "Automatic reverse engineering of data structures from binary execution". In: *Proceedings of the 11th Annual Information Security Symposium*. CERIAS '10. West Lafayette, IN: CERIAS - Purdue University, Mar. 2010, p. 1.
- [269] Treude, C. et al. "An Exploratory Study of Software Reverse Engineering in a Security Context". In: *2011 18th Working Conference on Reverse Engineering*. ISSN: 2375-5369. Oct. 2011, pp. 184–188.
- [270] Tellez, M., El-Tawab, S., and Heydari, M. H. "IoT security attacks using reverse engineering methods on WSN applications". In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. Dec. 2016, pp. 182–187.
- [271] Mayoral-Vilches, V. *RVD#2558: Default credentials on SICK PLC allows disabling safety features*. <https://github.com/aliasrobotics/RVD/issues/2558>. 2020.
- [272] Usategui, L., Glera, A., and Mayoral-Vilches, V. *RVD#2561: Unprotected BIOS allows user to boot from live OS image*. <https://github.com/aliasrobotics/RVD/issues/2561>. 2020.
- [273] Usategui, L., Glera, A., and Mayoral-Vilches, V. *RVD#2562: Booting from a live image leads to exfiltration of sensible information and privilege escalation*. <https://github.com/aliasrobotics/RVD/issues/2562>. 2020.

- 
- [274] Richardson, R. and North, M. M. "Ransomware: Evolution, mitigation and prevention". In: *International Management Review* vol. 13, no. 1 (2017), p. 10.
- [275] Formby, D., Durbha, S., and Beyah, R. "Out of control: ransomware for industrial control systems". In: *RSA conference*. 2017.
- [276] McDonald, G. and O’Gorman, G. "Ransomware: a growing menace". In: DOI= [http://www.01net.it/whitepaper\\_library/Symantec\\_Ransomware\\_Growing\\_Me.pdf](http://www.01net.it/whitepaper_library/Symantec_Ransomware_Growing_Me.pdf) (2012).
- [277] Bhardwaj, A. et al. "Ransomware digital extortion: a rising new age threat". In: *Indian Journal of Science and Technology* vol. 9, no. 14 (2016), pp. 1-5.
- [278] Robo09. *Universal Robot Security Vulnerabilities*. June 2018.
- [279] *Robotiq Community Discussion: Universal Robot Security Vulnerabilities*. <https://dof.robotiq.com/discussion/1195/universal-robot-security-vulnerabilities>. Accessed: 2024-07-06. 2020.
- [280] Standard, I. "ISO 10218-1:2011 Robots and robotic devices — Safety requirements for industrial robots — Part 1: Robots". In: *International Organization for Standardization* (2011).
- [281] Mayoral-Vilches, V. et al. "Towards self-adaptable robots: from programming to training machines". In: *ArXiv e-prints* (Feb. 2018). arXiv: 1802.04082 [cs.R0].
- [282] *Alias Robotics: Case Studies on Robot Cybersecurity*. <https://aliasrobotics.com/case-studies-robot-cybersecurity.php>. Accessed: 2024-07-06. 2020.
- [283] Rass, S. et al. "Game-theoretic APT defense: An experimental study on robotics". In: *Computers & Security* (2023), p. 103328.
- [284] Kamhoua, C. A., Leslie, N. O., and Weisman, M. J. "Game Theoretic Modeling of Advanced Persistent Threat in Internet of Things". In: *Journal of Cyber Security and Information Systems* vol. 6, no. 3 (2018).
- [285] Jajodia, S. et al. "Cauldron mission-centric cyber situational awareness with defense in depth". In: *2011 - MILCOM 2011 Military Communications Conference*. MILCOM 2011 - 2011 IEEE Military Communications Conference. Baltimore, MD, USA: IEEE, 2011, pp. 1339-1344.

- [286] Rass, S., König, S., and Panaousis, E. "Cut-The-Rope: A Game of Stealthy Intrusion". In: *Decision and Game Theory for Security*. Springer LNCS 11836, 2019, pp. 404–416.
- [287] alias Robotics. *Case Study - threat modeling a ROS2 robot*. 2019.
- [288] AcutronicRobotics. *Threat Model analysis for MARA robot*. original-date: 2019-04-07T15:53:00Z. Sept. 2021.
- [289] alias Robotics. *The Week of Mobile Industrial Robots's bugs*. en. June 2020.
- [290] alias Robotics. *Case Study - penetration testing Mobile Industrial Robots*. 2021.
- [291] König, S. et al. "Assessing the Impact of Malware Attacks in Utility Networks". In: *Game Theory for Security and Risk Management: From Theory to Practice*. Springer, 2018, pp. 335–351.
- [292] Deng, G. et al. "Pentestgpt: An llm-empowered automatic penetration testing tool". In: *arXiv preprint arXiv:2308.06782* (2023).
- [293] Vaswani, A. et al. "Attention is all you need". In: *Advances in neural information processing systems* vol. 30 (2017).
- [294] Radford, A. et al. "Improving language understanding with unsupervised learning". In: (2018).
- [295] Arkin, B., Stender, S., and McGraw, G. "Software penetration testing". In: *IEEE Security & Privacy* vol. 3, no. 1 (2005), pp. 84–87.
- [296] Deng, G. et al. "NAUTILUS: Automated RESTful API Vulnerability Detection". In: ()
- [297] Zhao, W. X. et al. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* (2023).
- [298] Liu, Y. et al. "Summary of chatgpt/gpt-4 research and perspective towards the future of large language models". In: *arXiv preprint arXiv:2304.01852* (2023).
- [299] Wei, J. et al. "Emergent abilities of large language models". In: *arXiv preprint arXiv:2206.07682* (2022).
- [300] Antunes, N. and Vieira, M. "Benchmarking vulnerability detection tools for web services". In: *2010 IEEE International Conference on Web Services*. IEEE. 2010, pp. 203–210.

- 
- [301] Xiong, P. and Peyton, L. "A model-driven penetration test framework for Web applications". In: *2010 Eighth International Conference on Privacy, Security and Trust*. IEEE. 2010, pp. 173–180.
- [302] Box, H. T. *HackTheBox: Hacking Training for the Best*. Accessed: Jan 12, 2024. 2024.
- [303] *VulnHub: Vulnerable By Design*. VulnHub Website. Accessed: January 12, 2024. 2018.
- [304] OWASP Foundation. *OWASP Foundation: The Open Source Foundation for Application Security*. Accessed: January 12, 2024. 2024.
- [305] OpenAI. *GPT-3.5: Overview and Capabilities*. <https://platform.openai.com/docs/models/>. Accessed on February 2, 2023. Nov. 2022.
- [306] OpenAI. *GPT-4: The Latest Advancement in Language Models*. <https://openai.com/research/gpt-4>. Accessed on June 30, 2023. Mar. 2023.
- [307] Google. *Bard: AI-Powered Search Tool*. <https://bard.google.com/?hl=en>. Accessed: January 12, 2024. 2023.
- [308] Rapid7. *Metasploit Framework*. Accessed: 30-07-2023. 2023.
- [309] Mauw, S. and Oostdijk, M. "Foundations of Attack Trees". In: vol. 3935. July 2006, pp. 186–198.
- [310] HackTheBox. *HackTheBox Active Machines*. Accessed: January 12, 2024. 2018.
- [311] Authors, A. *EXCALIBUR: Automated Penetration Testing*. <https://anonymous.4open.science/r/EXCALIBUR-Automated-Penetration-Testing/README.md>. 2023.
- [312] Weidman, G. *Penetration testing: a hands-on introduction to hacking*. No starch press, 2014.
- [313] Abu-Dabaseh, F. and Alshammari, E. "Automated penetration testing: An overview". In: *The 4th International Conference on Natural Language Computing, Copenhagen, Denmark*. 2018, pp. 121–129.
- [314] Pearce, H. et al. "Asleep at the keyboard? assessing the security of github copilot's code contributions". In: *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2022, pp. 754–768.
- [315] Pearce, H. et al. "Examining zero-shot vulnerability repair with large language models". In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 2339–2356.

## Bibliography

---

- [316] OWASP Juice-Shop Project. <https://owasp.org/www-project-juice-shop/>. 2022.
- [317] Project, N. *Nmap: Network Mapping Tool*. Accessed: January 12, 2024. 1997.
- [318] MITRE. *Common Weakness Enumeration (CWE)*. <https://cwe.mitre.org/index.html>. 2021.
- [319] Collins, E. *LAMDA: Our breakthrough conversation technology*. May 2021.
- [320] OpenAI. *ChatGPT: Conversational AI by OpenAI*. <https://chat.openai.com/>. Accessed on February 2, 2023. Jan. 2023.
- [321] Offensive Security. *Kali Linux: The Most Advanced Penetration Testing Distribution*. Official Website. Accessed: January 12, 2024. 2013.
- [322] Sonatype Inc. *Nexus Vulnerability Scanner*. <https://www.sonatype.com/products/vulnerability-scanner-upload>. Accessed: January 12, 2024. 2024.
- [323] Rahalkar, S. and Rahalkar, S. "Openvas". In: *Quick Start Guide to Penetration Testing: With NMAP, OpenVAS and Metasploit* (2019), pp. 47–71.
- [324] Guimaraes, B. and Stampar, M. *sqlmap: Automatic SQL injection and database takeover tool*. <https://sqlmap.org/>. 2022.
- [325] Yeo, J. "Using penetration testing to enhance your company's security". In: *Computer Fraud & Security* vol. 2013, no. 4 (2013), pp. 17–20.
- [326] Vaswani, A. et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].
- [327] Bang, Y. et al. "A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity". In: *arXiv preprint arXiv:2302.04023* (2023).
- [328] Wei, J. et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].
- [329] Lallie, H. S., Debattista, K., and Bal, J. "A review of attack graph and attack tree visual syntax in cyber security". In: *Computer Science Review* vol. 35 (2020), p. 100219.

- [330] Barbar, K. "Attributed tree grammars". In: *Theoretical Computer Science* vol. 119, no. 1 (1993), pp. 3–22.
- [331] Sun, H. et al. *AutoHint: Automatic Prompt Optimization with Hint Generation*. 2023. arXiv: 2307.07415 [cs.CL].
- [332] HackTheBox. *HackTheBox - Carrier*. Accessed: January 12, 2024. Sept. 2018.
- [333] Sullo. *Nikto Web Server Scanner*. GitHub Repository. Accessed: January 12, 2024. 2024.
- [334] OpenAI. *ChatGPT Plugins - Code Interpreter*. Accessed: January 12, 2024. 2023.
- [335] KajanM. *KAJANM/DirBuster: a multi threaded Java application designed to brute force directories and files names on web/application servers*.
- [336] Wang, J. et al. "Milvus: A Purpose-Built Vector Data Management System". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, pp. 2614–2627.
- [337] Guo, R. et al. "Manu: a cloud native vector database management system". In: *Proceedings of the VLDB Endowment* vol. 15, no. 12 (2022), pp. 3548–3561.
- [338] Zhang, M. et al. "How language model hallucinations can snowball". In: *arXiv preprint arXiv:2305.13534* (2023).
- [339] VulnHub. *Hackable II: Vulnerable By Design - VulnHub*. Accessed: January 12, 2024. 2021.
- [340] Liu, Y. et al. "Jailbreaking chatgpt via prompt engineering: An empirical study". In: *arXiv preprint arXiv:2305.13860* (2023).
- [341] Manakul, P., Liusie, A., and Gales, M. J. "Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models". In: *arXiv preprint arXiv:2303.08896* (2023).
- [342] Zamora, I. et al. "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo". In: *ArXiv e-prints* (Aug. 2016). arXiv: 1608.05742 [cs.R0].
- [343] *ROS2 Security Workshop | ROSCon 2019*. [https://ros-swg.github.io/ROSCon19\\_Security\\_Workshop/](https://ros-swg.github.io/ROSCon19_Security_Workshop/). Accessed: 2024-07-06. 2019.

- [344] *Doing Real-Time with ROS 2: Capabilities and Challenges*. <https://www.apex.ai/roscon2019>. Accessed: 2024-07-06. 2019.
- [345] Zhu, Q. et al. "Cybersecurity in Robotics: Challenges, Quantitative Modeling, and Practice". In: *Foundations and Trends® in Robotics* vol. 9, no. 1 (2021), pp. 1–129.
- [346] Lera, F. J. R. et al. "Threat modeling for robotic-based production plants". In: *2022 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2022, pp. 110–115.
- [347] Mayoral-Vilches, V. et al. "RobotPerf: An Open-Source, Vendor-Agnostic, Benchmarking Suite for Evaluating Robotics Computing System Performance". In: *arXiv preprint arXiv:2309.09212* (2023).
- [348] Rass, S. et al. *Cyber-Security in Critical Infrastructures: A Game-Theoretic Approach*. SpringerNature, 2020.
- [349] Dijk, M. van et al. "FlipIt: The Game of "Stealthy Takeover"". In: *J. Cryptol.* vol. 26, no. 4 (2013), pp. 655–713.
- [350] Zhang, R. and Zhu, Q. "FlipIn: A Game-Theoretic Cyber Insurance Framework for Incentive-Compatible Cyber Risk Management of Internet of Things". In: *IEEE Trans. on Information Forensics and Security* (2019), pp. 1–1.
- [351] Kushner, D. "The real story of stuxnet". In: *IEEE Spectrum* vol. 50, no. 3 (Mar. 2013), pp. 48–53.
- [352] Etesami, S. R. and Başar, T. "Dynamic Games in Cyber-Physical Security: An Overview". en. In: *Dynamic Games and Applications* (Jan. 2019).
- [353] *ADAPT: Analytical Framework for Actionable Defense against Advanced Persistent Threats | UW Department of Electrical & Computer Engineering*. 2018.
- [354] Qing, H. et al. "Advanced Persistent Threats Detection Game with Expert System for Cloud". In: *Journal of Computer Research and Development* vol. 54, no. 10, 2344 (2017), p. 2344.
- [355] Moothedath, S. et al. "A Game Theoretic Approach for Dynamic Information Flow Tracking to Detect Multi-Stage Advanced Persistent Threats". In: *arXiv:1811.05622 [cs]* (Nov. 2018). arXiv: 1811.05622.

- 
- [356] Khouzani, M., Sarkar, S., and Altman, E. "Saddle-Point Strategies in Malware Attack". In: *IEEE Journal on Selected Areas in Communications* vol. 30, no. 1 (Jan. 2012), pp. 31–43.
- [357] Carroll, T. E. and Grosu, D. "A Game Theoretic Investigation of Deception in Network Security". In: *2009 Proceedings of 18th International Conference on Computer Communications and Networks*. San Francisco, CA, USA: IEEE, Aug. 2009, pp. 1–6.
- [358] La, Q. D., Quek, T. Q. S., and Lee, J. "A game theoretic model for enabling honeypots in IoT networks". In: *2016 IEEE International Conference on Communications (ICC)*. Kuala Lumpur, Malaysia: IEEE, May 2016, pp. 1–6.
- [359] Fang, X. et al. "A Game Model for Predicting the Attack Path of APT". In: *2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing*. Dalian, China: IEEE, Aug. 2014, pp. 491–495.
- [360] Panaousis, E. et al. "Game theoretic path selection to support security in device-to-device communications". In: *Ad Hoc Networks* vol. 56 (2017), pp. 28–42.
- [361] Alpcan, T. and Başar, T. *Network Security: A Decision and Game Theoretic Approach*. Cambridge University Press, 2010.
- [362] Lye, K.-w. and Wing, J. M. "Game strategies in network security". In: *International Journal of Information Security* vol. 4 (2005), pp. 71–86.
- [363] Yang, L.-X. et al. "A risk management approach to defending against the advanced persistent threat". In: *IEEE Trans. on Dependable and Secure Computing* (2018), pp. 1–1.
- [364] Hota, A. R. et al. "A Game-Theoretic Framework for Securing Interdependent Assets in Networks". In: *Game Theory for Security and Risk Management*. Ed. by Rass, S. and Schauer, S. Series Title: Static & Dynamic Game Theory: Foundations & Applications. Springer, 2018, pp. 157–184.
- [365] Hota, A. R. et al. "Optimal and Game-Theoretic Deployment of Security Investments in Interdependent Assets". In: *Decision and Game Theory for Security*. Springer LNCS 9996, 2016, pp. 101–113.

- [366] Lin, J., Liu, P., and Jing, J. "Using Signaling Games to Model the Multi-step Attack-Defense Scenarios on Confidentiality". In: *Decision and Game Theory for Security*. Ed. by Grossklags, J. and Walrand, J. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 118–137.
- [367] Zhu, Q. and Rass, S. "On Multi-Phase and Multi-Stage Game-Theoretic Modeling of Advanced Persistent Threats". In: *IEEE Access* vol. 6 (2018), pp. 13958–13971.
- [368] Senejohnny, D., Tesi, P., and De Persis, C. "A Jamming-Resilient Algorithm for Self-Triggered Network Coordination". In: *IEEE Trans. on Control of Network Systems* vol. 5, no. 3 (Sept. 2018), pp. 981–990.
- [369] Yang, L.-X. et al. "Effective Repair Strategy Against Advanced Persistent Threat: A Differential Game Approach". In: *IEEE Trans. on Information Forensics and Security* vol. 14, no. 7 (July 2019), pp. 1713–1728.
- [370] Touhiduzzaman, M., Hahn, A., and Srivastava, A. K. "A Diversity-Based Substation Cyber Defense Strategy Utilizing Coloring Games". In: *IEEE Trans. on Smart Grid* vol. 10, no. 5 (Sept. 2019), pp. 5405–5415.
- [371] Li, Y. et al. "An Intelligence-Driven Security-Aware Defense Mechanism for Advanced Persistent Threats". In: *IEEE Trans. on Information Forensics and Security* vol. 14, no. 3 (Mar. 2019), pp. 646–661.
- [372] Soltan, S., Yannakakis, M., and Zussman, G. "REACT to Cyber Attacks on Power Grids". In: *IEEE Trans. on Network Science and Engineering* vol. 6, no. 3 (July 2019), pp. 459–473.
- [373] Pawlick, J. and Zhu, Q. "Strategic Trust in Cloud-Enabled Cyber-Physical Systems With an Application to Glucose Control". In: *IEEE Trans. on Information Forensics and Security* vol. 12, no. 12 (Dec. 2017), pp. 2906–2919.
- [374] Pawlick, J., Chen, J., and Zhu, Q. "iSTRIC: An Interdependent Strategic Trust Mechanism for the Cloud-Enabled Internet of Controlled Things". In: *IEEE Trans. on Information Forensics and Security* vol. 14, no. 6 (June 2019), pp. 1654–1669.

- [375] Yuan, H. et al. "Stackelberg-game-based defense analysis against advanced persistent threats on cloud control system". In: *IEEE Trans. on Industrial Informatics* (2019), pp. 1–1.
- [376] Huang, L. and Zhu, Q. "Adaptive Strategic Cyber Defense for Advanced Persistent Threats in Critical Infrastructure Networks". In: *arXiv:1809.02227 [cs]* (Sept. 2018). arXiv: 1809.02227.
- [377] Rass, S. and Zhu, Q. "GADAPT: A Sequential Game-Theoretic Framework for Designing Defense-in-Depth Strategies Against Advanced Persistent Threats". In: *Decision and Game Theory for Security*. Springer LNCS 9996, 2016, pp. 314–326.
- [378] Hota, A. R. and Sundaram, S. "Interdependent Security Games on Networks Under Behavioral Probability Weighting". In: *IEEE Trans. on Control of Network Systems* vol. 5, no. 1 (Mar. 2018), pp. 262–273.
- [379] Xiao, L. et al. "Attacker-Centric View of a Detection Game against Advanced Persistent Threats". In: *IEEE Trans. on Mobile Computing* vol. 17, no. 11 (Nov. 2018), pp. 2512–2523.
- [380] Fudenberg, D. and Tirole, J. *Game Theory*. London: MIT Press, 1991.