SaMOSA: Sandbox for Malware Orchestration and Side-Channel Analysis

Meet Udeshi
m.udeshi@nyu.edu
NYU Tandon School of Engineering
Brookyln, NY, USA

Venkata Sai Charan Putrevu
v.putrevu@nyu.edu
NYU Tandon School of Engineering
Brookyln, NY, USA

Prashanth Krishnamurthy
prashanth.krishnamurthy@nyu.edu
NYU Tandon School of Engineering
Brookyln, NY, USA

Ramesh Karri
rkarri@nyu.edu
NYU Tandon School of Engineering
Brookyln, NY, USA

Farshad Khorrami
khorrami@nyu.edu
NYU Tandon School of Engineering
Brookyln, NY, USA

ABSTRACT

Cyber-attacks on operational technology (OT) and cyber-physical systems (CPS) have increased tremendously in recent years with the proliferation of malware targeting Linux-based embedded devices of OT and CPS systems. Comprehensive malware detection requires dynamic analysis of execution behavior in addition to static analysis of binaries. Safe execution of malware in a manner that captures relevant behaviors via side-channels requires a sandbox environment. Existing Linux sandboxes are built for specific tasks, only capture one or two side-channels, and do not offer customization for different analysis tasks. We present the SaMOSA Linux sandbox that allows emulation of Linux malwares while capturing time-synchronized side-channels from four sources. SaMOSA additionally provides emulation of network services via FakeNet, and allows orchestration and customization of the sandbox environment via pipeline hooks. In comparison to existing Linux sandboxes, SaMOSA captures more side-channels namely system calls, network activity, disk activity, and hardware performance counters. It supports three architectures predominantly used in OT and CPS namely x86-64, ARM64, and PowerPC 64. SaMOSA fills a gap in Linux malware analysis by providing a modular and customizable sandbox framework that can be adapted for many malware analysis tasks. We present three case studies of three different malware families to demonstrate the advantages of SaMOSA.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**; **Virtualization and security**; *Distributed systems security*.

KEYWORDS

Linux Sandbox, Malware Analysis, Malware Emulation, Operational Technology, Side Channels

1 INTRODUCTION

The proliferation of embedded computers connected to the internet has grown immensely with the modernization of operational technology (OT) and cyber-physical systems (CPS). Majority of these run a Linux operating system (OS) on ARM, PowerPC or x86 architectures [2, 7]. In effect, malwares targetted towards Linux-based embedded devices have seen a proportional spike. While malware analysis has long focused on Windows due to widespread adoption and higher prevalence of attacks, analysis of Linux binaries is extremely important for OT and CPS security, so that malwares are detected before they cause lasting damage. While static analysis can reveal known malwares based on code signatures, an unseen malware's intentions become more apparent during execution, thus requiring dynamic analysis of execution behavior. Dynamic analysis of malware involves analysis of its execution trace captured via different side-channel sources such as system calls (syscall), hardware performance counters (HPC), network activity, and disk activity [10, 24]. Many approaches use statistical and machine learning based methods to detect anomalous execution behavior [1]. While methods focused on embedded devices of OT and CPS primarily use microarchitectural side-channels such as HPC [14, 26, 27], some incorporate multiple sources simultaneously for more elaborate detection [28]. For dynamic analysis, safe execution of binaries requires a sandbox environment where a malware is sufficiently isolated so that it cannot affect important systems, yet it is provided all capabilities to perform malicious actions. Additionally, the sandbox should capture all relevant information about the binary's execution such that any malicious behavior is meaningfully tracked and can be detected. We present the SaMOSA sandbox that is tailored for Linux malware analysis, supports multiple processor architectures popular in OT and CPS, captures multiple side-channels for in-depth analysis, provides pipeline hooks for custom orchestration of the sandbox environment, and automates the end-to-end execution pipeline. Alrawi et al. [3] studied malware sandboxes of the last 20 years and proposed sandbox design guidelines in which they outlined that the sandbox should rely on emulation or virtualization, be modular and customizable, perform monitoring outside the sandbox environment as much as possible, provide a realistic OS along with an emulated network environment to trigger malware behavior, and collect monitoring data from multiple components for comparative analysis. SaMOSA is designed in line with these guidelines. The contributions of this paper are threefold:

(1) The SaMOSA sandbox for automated malware emulation and analysis on Linux for x86, ARM, and PowerPC
(2) Orchestration framework to run user-defined commands and scripts via pipeline hooks, providing versatility and customization for different malware analysis
(3) Capturing four time-synchronized side-channels (system calls, hardware performance counters, network activity, disk activity) for deep insights into malware behavior

2 RELATED WORK

Table 1: Comparison of existing Linux sandboxes

Sandbox	Multi Arch	Network Mon.	HPC Mon.	Disk Mon.	Syscall Mon.	Network Emu.	Orchestration
Limon [17]	✗	✓	✗	✗	✓	✗	✗
F-Sanbox [18]	✗	✓	✗	✗	✓	✗	✗
Detux [12]	✓	✓	✗	✗	✓	✗	✗
Padawan [10]	✓	✓	✗	✗	✓	✓	✗
ELFEN [13]	✓	✓	✗	✗	✓	✗	✗
LiSa [25]	✓	✓	✗	✗	✓	✗	✗
SaMOSA (ours)	✓	✓	✓	✓	✓	✓	✓

Several sandboxes have been developed for binary analysis. Cuckoo Sandbox [11] and its open-source successor CAPE Sandbox [6] are widely used for Windows, however they do not support Linux binaries and require complex, time-consuming configuration. To address this gap, several Linux sandboxes were developed. Limon [17] and Detux [12] are Python-based sandboxes for emulating Linux malwares. However, their scripts and VM images are out-dated, posing challenges in updating to latest OS and software needed to analyze recent malware. F-sandbox [18], derived from Firmadyne [8] and Detux frameworks, is specifically designed for ELF malware targeting MIPS architecture. Padawan sandbox [10] performs static analysis via binary disassembly and employs QEMU for sandboxing and dynamic analysis with support for multiple architectures. It captures low-level kernel and userspace events via SystemTap. However, Padawan is closed-source and not publicly available, posing similar challenges in updating to latest software. LiSa Sandbox[25] and ELFEN [13] are recent fully automated Linux malware analysis platforms, supporting static analysis via strings and YARA signatures and dynamic analysis via QEMU emulation. LiSa also captures system calls via SystemTap and additionally captures network activity from inside the sandbox, while ELFEN relies on eBPF filters. Both LiSa and ELFEN use buildroot-based custom-compiled OS images that provide a minimal Linux environment which may not represent a full-featured OS environment like Ubuntu. Also, using SystemTap or eBPF for capturing system calls and events requires implementing custom filters for specific events which is more suitable for debugging specific issues rather than capturing comprehensive system information for malware analysis.

Although these sandboxes target Linux, they are limited in the environments they provide and the execution traces they capture. To address these limitations, we developed a Linux sandbox that fully automates the binary execution pipeline and captures granular time series based execution trace information regarding system calls, hardware performance counters, network access, and disk activity. Table 1 provides a comprehensive feature comparison with related sandboxes.

3 IMPLEMENTATION

SaMOSA incorporates the following components: virtual machine (VM) emulation using QEMU [21]; network emulation using FakeNet [15] and capture using `tcpdump`; system call capture using `sysdig` [23] running inside the sandbox; hardware performance counter (HPC) measurements of the QEMU process using `perf`; and, disk access measurements via QEMU tracing. Figure 1 shows an overview of SaMOSA with three stages: orchestration, execution, and analysis. Orchestration involves setting up and configuring the sandbox environment by selecting specific VM image to boot up, providing configuration options for execution, and adding pipeline hooks with custom scripts and commands. The orchestration stage allows the user to customize the SaMOSA sandbox as desired for particular malware. Next, the execution stage boots up the VM, sets up all monitoring utilities, initiates the desired network configuration, and executes the binary. The execution stage is end-to-end automated and does not require user involvement. Finally, the analysis stage collects the time synchronized side-channel data along with the VM snapshot into a directory for easy post-processing. For this paper, we perform minimal post-processing and analysis, however we ensure to capture rich data so that it can support advanced analysis to provide deep insights.

3.1 Sandbox VM Configuration

The sandbox operates inside a virtual machine (VM) emulated using the QEMU multi-architecture emulator [21]. QEMU supports multiple architectures popular in OT and CPS systems like x86, ARM, and PowerPC. We operate QEMU in full-system emulation mode to run an entire OS instead of single binaries. Unlike existing sandboxes, we install the popular OSes like Ubuntu or Debian on the VM instead of building a custom minimal OS. This provides a realistic environment in the sandbox that reflects real-world setups. SaMOSA's architecture support also depends on Sysdig and filesystem device availability. We support three architectures with SaMOSA: x86-64, ARM64, and PowerPC64 little endian (PPC64LE). We built VM images pre-installed with Ubuntu 20.04 for x86-64 and ARM64. We used a pre-built Debain Trixie QEMU image for PPC64LE[1]. We installed the Sysdig prebuilt packages for x86-64 and ARM64, and compiled it from source code for PPC64LE as no pre-built package was available. We could not use Ubuntu 20.04 for PPC64LE as Sysdig compilation failed due to a kernel version incompatibility, so we chose the Debian image.

For x86-64, we run QEMU with kernel virtualization (KVM) so it runs on the host CPU. We attach a NVMe disk device that traces reads and writes with a logical block address. For ARM64, we run the `virt` machine with a Cortex A72 CPU to mimic the hardware of a Raspberry Pi. As NVMe is not supported, we attach a virtio block device that traces reads and writes with a sector address instead. For PPC64LE, we run the P-series machine with the Power9 CPU, and attach a NVMe disk device. For each architecture, the VMs are booted with 4GB RAM and 4 cores.

3.2 Network Configuration and FakeNet

We create a network bridge interface along with a subordinate tap interface on the host for the sandbox to isolate it's network traffic

[1]PPC64LE image is from https://people.debian.org/~gio/dqib/.

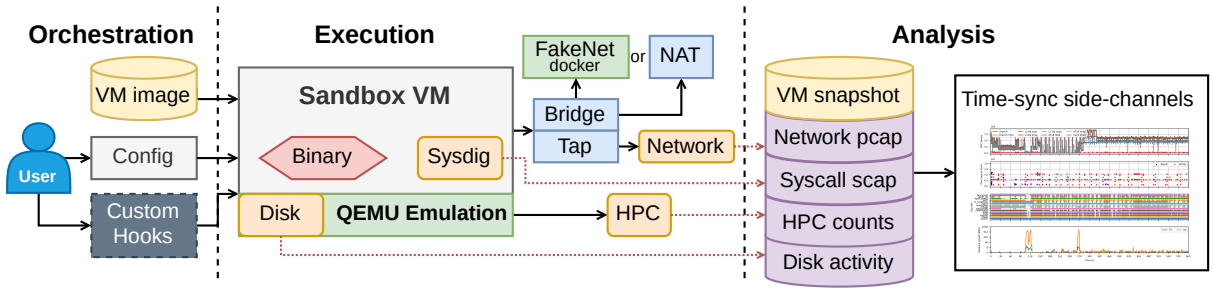

Figure 1: Overview of the SaMOSA sandbox. SaMOSA involves three stages: *orchestration* where the user configures hooks to customize the sandbox environment, *execution* which executes the binary while collecting side-channels, and *analysis* where the time-synchronized data is post-processed and plotted to observe correlations across multiple side-channels.

from the internet. The FakeNet tool intercepts and redirects all network traffic according to configurable rules, emulating internet services such as DNS or HTTP servers. We invoke FakeNet inside a separate Docker container running on the host so that it does not interfere with host network interfaces. The Docker container is connected to the sandbox bridge, and the QEMU process is connected to the sandbox tap. This allows FakeNet to intercept and manage all sandbox traffic. FakeNet responds to intercepted network requests from the sandbox with a generally positive reply (e.g., providing a fake HTML page to an HTTP request of any URL). This emulation enables interesting malware behaviors in case they require network interaction with command-and-control (C2) servers to begin operations. In cases where internet access is required for a particular malware, we do not invoke FakeNet and instead setup Network Address Translation (NAT) for the sandbox bridge via IP tables on the host. This is necessary when malwares download additional payloads from the internet for further execution (e.g., cryptominer bots download mining software from public websites like GitHub).

3.3 Side-channel Monitoring

Syscalls: System calls reveal important information about how the binary interacts with the OS. We use `sysdig` [23] to capture OS-wide system calls during binary execution. `sysdig` is executed inside the VM, just before the binary, and stopped after the elapsed execution time. It stores syscalls in a capture file that is copied out after execution.

HPC measurement: HPC measurements capture low-level execution patterns such as cache usage, branch mispredictions, and CPU activity. We capture HPC measurements using the `perf` tool directly of the host QEMU process that emulates the sandbox. This is advantageous over capturing HPC within the sandbox as host-side HPC provides low-level hardware measurements reflecting the resource usage of the entire sandbox. In contrast, collecting HPC measurements from within the emulated system can be unreliable, particularly when testing malware that actively evades monitoring by disabling or tampering with system measurement tools.

Network activity: Network activity is captured via `tcpdump` on the sandbox tap interface so that it records all sandbox activity in either the FakeNet or the NAT configuration. As the VMs have been assigned static IPs, this allows us to determine the direction of packets into and out of the sandbox during post-processing.

Disk activity: Measuring disk activity can reveal malicious behavior like reading, rewriting, and renaming multiple files typical of ransomware, or writing logs or caches typical of crypto-miners. We leverage QEMU's built-in tracing infrastructure to monitor disk activity. We trace read/write events on the emulated NVMe and virtio block partitions, which captures logical blocks accessed. By using QEMU's tracing at the hypervisor layer, we avoid overloading the sandbox with disk activity monitoring agents, thus reducing the chance of detection or interference. Additionally, QEMU's traces record host-side timestamps, allowing time synchronization with other side-channels.

3.4 Orchestration and Customization

SaMOSA is modular and customizable to allow configuration of user-specified commands triggered via hooks in the sandbox execution pipeline. We define four hook points as follows: "Pre Setup" before the VM is setup and booted, "Pre Run" before measurement starts and the binary is executed, "Post Run" after binary is halted and measurement has stopped, "Post Shutdown" after the VM is shutdown. The "Pre Setup" and "Post Shutdown" hooks can only be run on the host, whereas "Pre Run" and "Post Run" hooks can be run either on the host or inside the sandbox. The hooks can be used to create files, perform setup steps, install additional packages, or customize the environment. In this manner, the execution pipeline can be tailored to certain malware, for example, by setting certain environment variables or modifying the firewall configuration in the sandbox before execution. This makes the sandbox framework versatile and extensible for different kinds of malware analysis.

3.5 Execution Pipeline

Figure 2 shows the step-by-step execution pipeline of the sandbox. When the sandbox is triggered, first the "Pre Setup" hooks are triggered. Then, a VM snapshot image is cloned from one of the pre-installed images depending on architecture and OS, and booted with QEMU. Disk tracing is enabled via QEMU tracing options. Once the VM boots up, the binary is copied into the sandbox via a secure shell copy operation (`scp`). Then, the "Pre Run" hooks are triggered. FakeNet is launched inside a Docker container on the host and connected to the sandbox bridge. HPC capture is initialized on the QEMU process using `perf`. Sysdig is triggered inside the sandbox and it writes to a temporary RAM partition so that it does

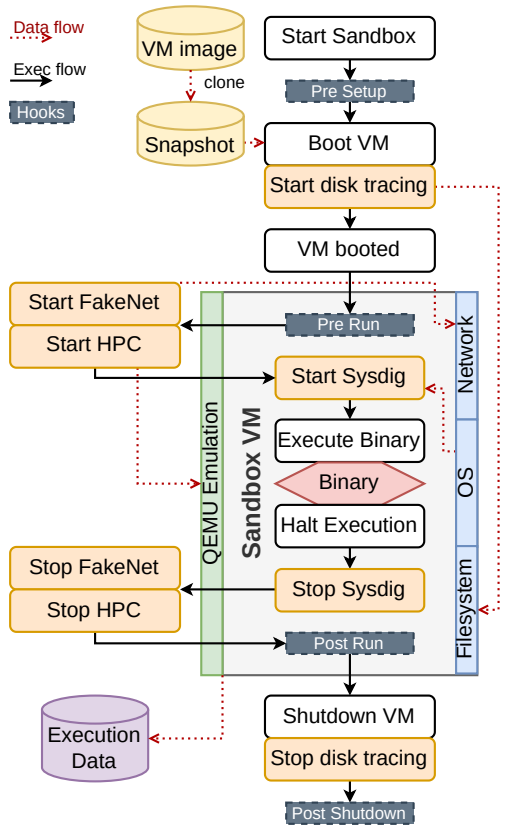

Figure 2: SaMOSA execution pipeline. Each step along with the user-specified custom hooks run automatically to start the sandbox, initiate side-channel monitors, execute the binary, and collect the side-channel data.

not interfere with disk activity. Then, the binary is executed. After the configured execution time elapses, the binary is halted. The "Execute Binary" and "Halt Execution" steps record timestamps, so that measurements outside these timestamps can be discarded during post-processing analysis. Subsequently, Sysdig, HPC, and FakeNet are terminated. The "Post Run" hooks are triggered. Finally, the syscall capture files and program output files of the binary are copied from the VM to the host and the VM is shut down, after which the "Post Shutdown" hooks are triggered. The pipeline is automated end-to-end, so no user interaction is required to execute the binary, start the components, or collect the measurements.

3.6 Analysis

The framework collects execution data and places it in a directory for post-processing. Appendix A.1 lists details of the captured data. The captured data contains a full picture of the entire sandbox execution so that it can support any kind of malware analysis. Automated malware detection algorithms rely on statistical or machine learning methods that require aggregation of the data into a time series. Whereas, deep manual analysis can involve inspecting packets and syscalls for specific information such as IP address, HTTP URLs, or program arguments. For this paper, we aggregate the measurements to display time series plots of activity to demonstrate the advantages of capturing multiple synchronized side-channels.

Section 4 shows plots generated using our analysis. The HPC measurements do not require further aggregation and the counts are plot versus time directly. The disk activity is displayed as a scatter plot where each point is plot as the logical block address versus time, with different colors for reads and writes. Alternately, we can aggregate the activity to plot access speed in bytes per second or similar. Syscalls are also displayed using a scatter plot of syscall type versus time. In this manner, we get a linear scatter plot per syscall type showing when that syscall was triggered. Network activity is aggregated as transmit (TX) and recieve (RX) speed in kilobytes per second by accumulating the packet sizes in a one millisecond window. More advanced network activity analysis is possible by aggregating traffic per IP address and port.

4 CASE STUDIES

We present three case studies of real-world malwares from different families. We demonstrate SaMOSA's advantages with examples of how orchestration helps to customize the sandbox and how the time-synchronized side-channels reveal correlations and provide a deeper understanding of malware behavior.

4.1 GonnaCry Ransomware

GonnaCry[2] is an open-source ransomware variant based on the WannaCry ransomware [5, 9] and targetted for Linux OT systems. It is implemented in Python and compiled as a packed binary containing the Python interpreter library along with python bytecode. GonnaCry encrypts files using 256-bit AES, secures the AES keys using 2048-bit RSA, and "shred"s the original files on disk by overwriting with random data to make recovery difficult. We compiled GonnaCry on a seperate PPC64LE system and executed it with our PPC64LE sandbox. To provide the ransomware with a variety of files for encryption, we orchestrated the sandbox by triggering a custom file generation script at the "Pre Run" stage. The script randomly generated several files with popular document extensions such as "doc", "txt", "pdf", "ppt", etc.

Figure 3 shows the execution activity of GonnaCry running on PPC64LE. We display the HPC values, disk activity, and 15 of the most frequent system calls. We do not display network activity as there is none. The HPC values show high memory load activity in terms of data translation look-aside buffer (dTLB) loads and L1 data cache loads, along with high memory store activity. The HPC activity by itself indicates heavy disk access which is also seen in the disk activity plot. Additionally, the addressing in the disk activity plot reveal that the same logical blocks are being read and written, indicating that many files were overwritten. We also see continuous use of many syscalls for file reading and writing ("read", "write", "llseek", "close", "openat") in addition to "getrandom", hinting at heavy encryption activity. Certain syscalls are only present at the start, such as "statx", "readlinkat", and "getdents64" which are used for listing folder contents. Correlating behaviors across multiple side-channels provides deeper context into GonnaCry's behavior

[2] https://github.com/tarcisio-marinho/GonnaCry

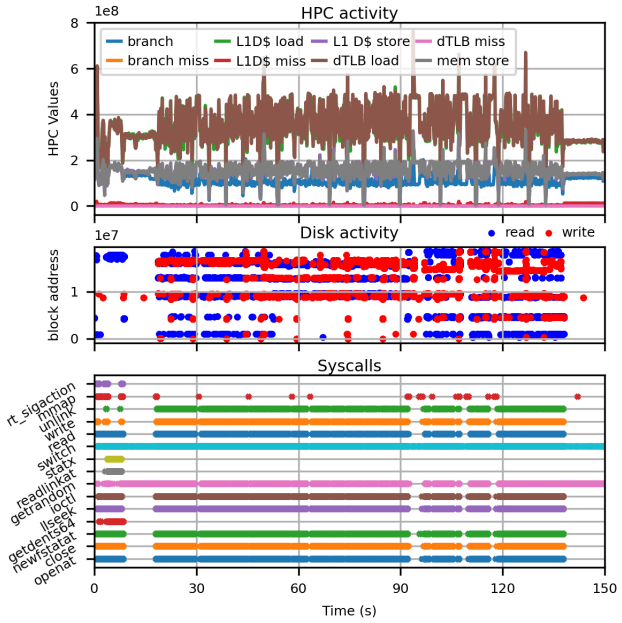

Figure 3: Execution plot of the GonnaCry ransomware on PPC64LE showing HPC, disk activity, and top 15 syscalls.

where it searches and enumerates files at the start, then proceeds with encryption.

4.2 CHAOS Remote Access Trojan

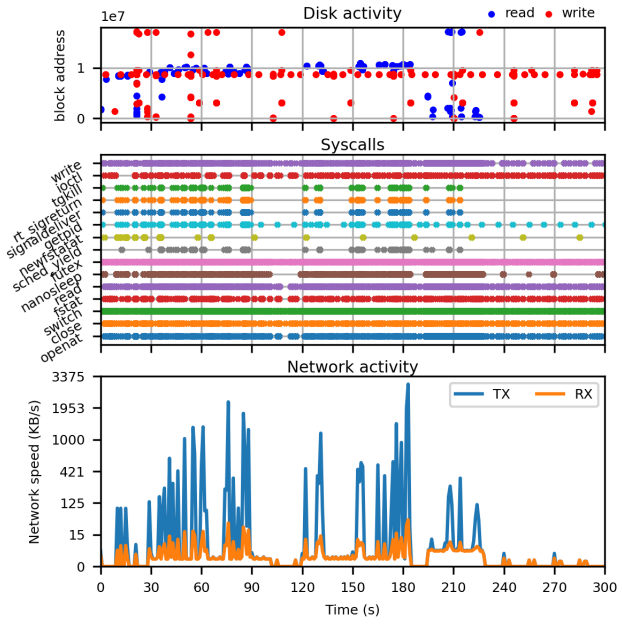

Figure 4: Execution of the CHAOS RAT on x86-64 showing disk activity, top 15 syscalls, and network activity.

CHAOS[3] is a Golang-based open-source remote access tool with comprehensive capabilities for remote administration of Windows and Linux systems. Recently, CHAOS has been used as a remote access trojan (RAT) for data exfiltration and deploying cryptominer bots [16, 19] as it provides a command-and-control (C2) server along with capabilities to execute system commands and exfiltrate files. Appendix A.2 elaborates the functionality of CHAOS.

We executed the CHAOS RAT on our x86-64 sandbox. We utilized SaMOSA's orchestration feature to setup the remote C2 server and trigger an interaction script to communicate with the RAT via the server. The C2 server was started before the VM boots at the "Pre Setup" stage and the interaction script was triggered before execution at the "Pre Run" stage. The interaction script performed various actions such as browsing folders, creating a new user with administrator (sudo) permissions, downloading files and system logs. Figure 4 shows the execution activity. There is not much HPC activity as the RAT is a passive process that waits for commands from the C2 server. Syscalls such as "futex", "nanosleep", "tgkill", and "signaldeliver" indicate that the process uses multithreading and sleeps frequently while waiting for commands. At 20 seconds, we can see interesting correlations such as heavy HPC activity, network activity, and disk activity at lower addresses. We analyzed the syscalls near these timestamps to reveal processes that read password files, created a new user, modified group permissions, and changed passwords. Each of these involve reads and writes to lower block addresses corresponding to system files containing user and group information. In this manner, time-synchronized side-channel activity helped identify activity hotspots for deeper analysis.

The activity from 30 to 180 seconds involved downloading several files from the user directory. We see correlation across the heavy network transfers, multiple reads in the disk activity, and HPC spikes. Similarly, the activity from 195 to 230 seconds shows reads at lower addresses, indicating that the C2 server is downloading system files. After that, we only see regular network activity every 15 seconds, indicating a status or heartbeat signal to keep the RAT connection alive. Identifying these hotspots helps during network analysis to find files that the RAT exfiltrated or injected.

4.3 Kinsing Cryptominer

Kinsing is a cryptomining malware family that has actively targeted Linux OT systems [22] by forming a botnet, spreading laterally across the network, installing rootkits [20, 24], and deploying cryptocurrency miners. We executed the Kinsing payload in our ARM64 sandbox. We utilized SaMOSA's orchestration to re-enable the SSH service that Kinsing disables so that we could extract information from the sandbox, triggered at the "POST RUN" stage. Initially, we ran the payload using FakeNet to emulate the internet services. FakeNet captured the list of C2 IP addresses and URLs accessed by Kinsing to reveal the additional payloads that were downloaded. Figure 6 shows the access logs that list the C2 server IP address (78.153.XX.XX, 80.64.XX.XX) and the HTTP requests for `kinsing_aarch64`, `libsystem.so`, and `ce.sh`.

Even though FakeNet captured the HTTP requests, it only provided fake payloads so we did not observe any further activity. Based on these insights, we triggered Kinsing again using the NAT

[3] https://github.com/tiagorlampert/CHAOS

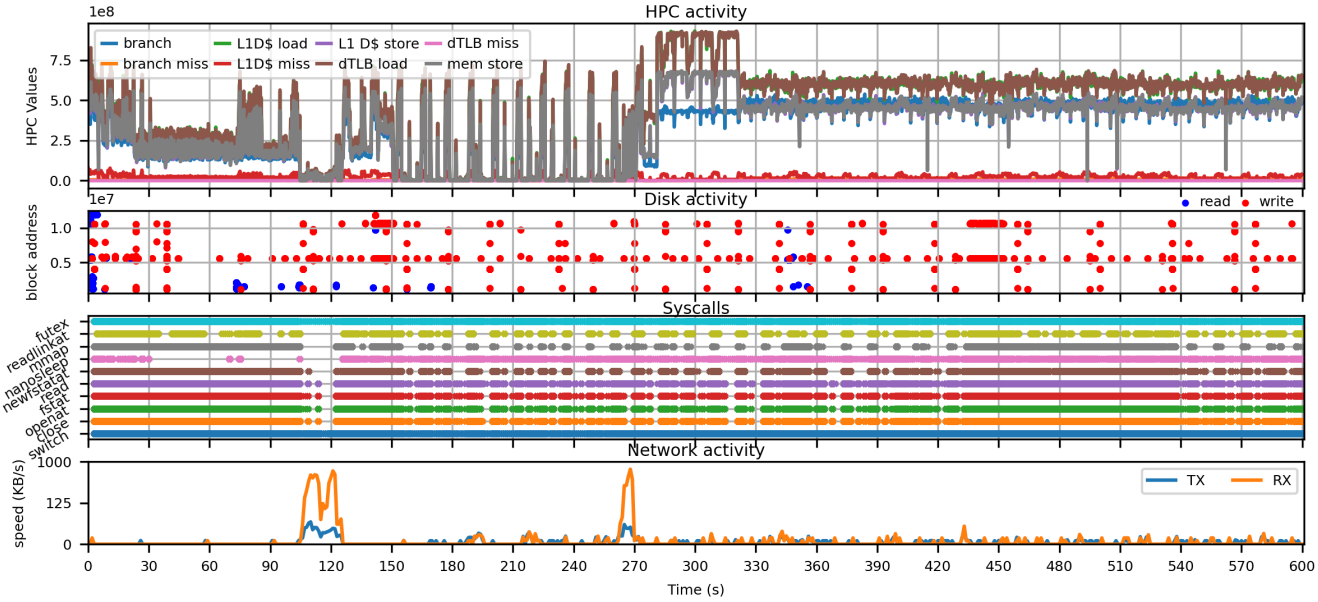

Figure 5: Execution plot of the Kinsing cryptominer on ARM64 showing HPC, disk activity, top 10 syscalls, and network activity.

```
[HTTPListener80] GET /kinsing_aarch64 HTTP/1.1
[HTTPListener80] Host: 78.153.XX.XX
[HTTPListener80] User-Agent: curl/7.68.0
[HTTPListener80] Accept: */*
...
[HTTPListener80] GET /libsystem.so HTTP/1.1
[HTTPListener80] Host: 78.153.XX.XX
[HTTPListener80] User-Agent: curl/7.68.0
[HTTPListener80] Accept: */*
...
[HTTPListener80] GET /ce.sh HTTP/1.1
[HTTPListener80] User-Agent: Wget/1.20.3 (linux-gnu)
[HTTPListener80] Accept: */*
[HTTPListener80] Host: 80.64.XX.XX
[HTTPListener80] Connection: Keep-Alive
```

Figure 6: Kinsing C2 server accesses captured by FakeNet.

so that it could download the correct payloads. Figure 5 shows the execution activity. From 100 to 120 seconds, it downloaded the ARM64-specific Kinsing payload and `libsystem.so` as previously seen in the FakeNet logs. `libsystem.so` is a stealthy rootkit allowing Kinsing to modify system behavior and hide its presence. Between 260 to 270 seconds, it contacted retrieved secondary payloads from additional IPs (185.154.XX.XX, 31.184.XX.XX) as seen in network activity. The malware behavior changed with full internet access as it successfully downloaded the initial payloads. At 270 seconds, it initiated cryptocurrency mining evident via the heavy HPC activity, notably spikes in memory stores, branch operations, and data translation look-aside buffer (dTLB) loads. We observed a bump in network traffic at 430 seconds along with subsequent disk write activity. Deeper packet analysis revealed that Kinsing downloaded a script `cron.sh` from 78.153.XX.XX which eliminates competing malware and establishes persistence. This behavior differs from the one seen with FakeNet where it accessed another script `ce.sh`. After mining started, we also observed periodic network activity related to seed hash and blob number exchanges along with heartbeat signals reflecting ongoing botnet communication and health monitoring. In this manner, SaMOSA helped analyze the Kinsing cryptominer in-depth, first via FakeNet and then with internet access via NAT to reveal interesting malicious behavior correlated across multiple side-channels.

5 CONCLUSION

In this work, we presented SaMOSA, a Linux sandbox framework designed to facilitate comprehensive malware analysis on Linux systems. SaMOSA supports full-system emulation across multiple architectures (x86-64, ARM64, PPC64LE) and captures time-synchronized execution data across four key side-channels (system calls, hardware performance counters, disk activity, and network traffic). SaMOSA offers extensive insights into malware behavior via the time-synchronized side-channels to aid in dynamic malware analysis. Its modular design, support for orchestration via custom pipeline hooks, and provision of real-world operating systems like Ubuntu and Debian make it adaptable for a wide range of malware analysis tasks. We present case studies of ransomware, remote access trojans, and cryptomining bots to demonstrated how analysis of time-synchronized side-channels offers deep insight into malware behavior. SaMOSA bridges a critical gap in Linux malware analysis by offering a customizable, end-to-end automated, and multi-architecture platform, laying the groundwork for malware detection and threat intelligence for Linux OT and CPS systems.

ACKNOWLEDGMENTS

This work was supported in part by the DOE NETL grants DE-CR0000051 and DE-CR0000017, and the NSF SaTC grant 2039615.

REFERENCES

[1] Amr S Abed, T Charles Clancy, and David S Levy. 2015. Applying bag of system calls for anomalous behavior detection of applications in linux containers. In *2015 IEEE globecom workshops (GC Wkshps)*. IEEE, 1–5.

[2] Mohannad Alhanahnah, Qicheng Lin, Qiben Yan, Ning Zhang, and Zhenxiang Chen. 2018. Efficient signature generation for classifying cross-architecture IoT malware. In *IEEE Conference on Communications and Network Security (CNS)*.

[3] Omar Alrawi, Miuyin Yong Wong, Athanasios Avgetidis, Kevin Valakuzhy, Boladji Vinny Adjibi, Konstantinos Karakatsanis, Mustaque Ahamad, Doug Blough, Fabian Monrose, and Manos Antonakakis. 2024. SoK: An Essential Guide For Using Malware Sandboxes In Security Applications: Challenges, Pitfalls, and Lessons Learned. arXiv:2403.16304 [cs.CR] https://arxiv.org/abs/2403.16304

[4] AQUASEC. [n. d.]. Kinsing V2. https://www.aquasec.com/blog/threat-alert-kinsing-malware-container-vulnerability/. Accessed: 2025-06-07.

[5] Jano Bermudes. 2023. Mitigating cyber risks in industrial control systems. www.marsh.com/en/industries/manufacturing/insights/mitigating-cyber-risks-in-industrial-control-systems.html Accessed: 2025-06-07.

[6] CAPE. [n. d.]. CAPE Sanbox. https://github.com/kevoreilly/CAPEv2.

[7] J. Carrillo-Mondéjar, J.L. Martínez, and G. Suarez-Tangil. 2020. Characterizing Linux-based malware: Findings and recent trends. *Future Generation Computer Systems* 110 (2020), 267–281. https://doi.org/10.1016/j.future.2020.04.031

[8] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. 2016. Towards automated dynamic analysis for linux-based embedded firmware.. In *Networked and Distributed Systems Security (NDSS)*.

[9] Qian Chen and Robert A. Bridges. 2017. Automated Behavioral Analysis of Malware: A Case Study of WannaCry Ransomware. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 454–460. https://doi.org/10.1109/ICMLA.2017.0-119

[10] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. 2018. Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 161–175.

[11] Cuckoo. [n. d.]. Cuckoo Sandbox. https://github.com/cuckoosandbox/cuckoo.

[12] Detux. [n. d.]. Detux Sandbox. https://github.com/detuxsandbox/detux.

[13] ELFEN. [n. d.]. ELFEN Sandbox. https://github.com/nikhilh-20/ELFEN.

[14] Prashanth Krishnamurthy, Ali Rasteh, Ramesh Karri, and Farshad Khorrami. 2024. Tracking Real-time Anomalies in Cyber-Physical Systems Through Dynamic Behavioral Analysis. arXiv:2406.12438 [eess.SY] https://arxiv.org/abs/2406.12438

[15] Mandiant. [n. d.]. FakeNet-NG. https://github.com/mandiant/flare-fakenet-ng.

[16] Alessandro Mascellino. 2022. Chaos RAT Used to Enhance Linux Cryptomining Attacks. https://www.infosecurity-magazine.com/news/chaos-rat-used-linux-cryptominingva/ Accessed: 2025-06-07.

[17] K. A. Monnappa. 2015. Automating Linux Malware Analysis Using Limon Sandbox. BlackHat Europe. https://www.blackhat.com/docs/eu-15/materials/eu-15-KA-Automating-Linux-Malware-Analysis-Using-Limon-Sandbox.pdf

[18] Tran Nghi Phu, Kien Hoang Dang, Dung Ngo Quoc, Nguyen Tho Dai, and Nguyen Ngoc Binh. 2019. A novel framework to classify malware in MIPS architecture-based IoT devices. *Security and Communication Networks* 2019, 1 (2019), 4073940.

[19] Santiago Pontiroli, Gabor Molnar, and Kirill Antonenko. 2025. From open-source to open threat: Tracking Chaos RAT's evolution. https://www.acronis.com/en-us/cyber-protection-center/posts/from-open-source-to-open-threat-tracking-chaos-rats-evolution/ Accessed: 2025-06-07.

[20] Venkata Sai Charan Putrevu, Subhasis Mukhopadhyay, Subhajit Manna, Nanda Rani, Ansh Vaid, Hrushikesh Chunduri, Mohan Anand Putrevu, and Sandeep Shukla. 2024. Adapt: Adaptive camouflage based deception orchestration for trapping advanced persistent threats. *Digital Threats: Research and Practice* 5, 3 (2024), 1–35.

[21] QEMU. [n. d.]. QEMU Emulator. https://www.qemu.org/docs/master/about/index.html.

[22] Redcanary. 2025. Kinsing Saltstack. https://redcanary.com/blog/threat-intelligence/kinsing-malware-citrix-saltstack/. Accessed: 2025-06-07.

[23] Sysdig. [n. d.]. Sysdig. https://github.com/draios/sysdig.

[24] Meet Udeshi, Prashanth Krishnamurthy, Ramesh Karri, and Farshad Khorrami. 2025. Tamper-Proof Network Traffic Measurements on a NIC for Intrusion Detection. *IEEE Transactions on Network and Service Management* 22, 2 (2025), 2214–2224. https://doi.org/10.1109/TNSM.2024.3512180

[25] Daniel Uhrıcek. 2020. Lisa – multiplatform linux sandbox for analyzing IoT malware.

[26] Xueyang Wang, Sek Chai, Michael Isnardi, Sehoon Lim, and Ramesh Karri. 2016. Hardware Performance Counter-Based Malware Identification and Detection with Adaptive Compressive Sensing. *ACM Trans. Archit. Code Optim.* 13, 1, Article 3 (March 2016), 23 pages. https://doi.org/10.1145/2857055

[27] Xueyang Wang, Charalambos Konstantinou, Michail Maniatakos, Ramesh Karri, Serena Lee, Patricia Robison, Paul Stergiou, and Steve Kim. 2016. Malicious firmware detection with hardware performance counters. *IEEE Transactions on Multi-Scale Computing Systems* 2, 3 (2016), 160–173.

[28] Luxin Zheng, Jian Zhang, Faxin Lin, and Xiangyi Wang. 2023. Feature-Fusion-Based Abnormal-Behavior-Detection Method in Virtualization Environment. *Electronics* 12, 16 (2023). https://doi.org/10.3390/electronics12163386

A APPENDIX

A.1 Analysis Data

SaMOSA captures time-synchronized side-channel data for analysis. The following data are captured and placed in the run directory for post-processing and analysis:

(1) Network packet capture (`pcap`) file that can be parsed with `tcpdump` or Wireshark
(2) Syscalls as a system capture (`scap`) file that can be parsed by `sysdig`
(3) HPC measurements as comma seperated values (CSV) where each row contains a timestamp and readings of each HPC counter that was recorded
(4) Disk activity log of timestamped events indicating filesystem read or write along with logical block number
(5) FakeNet generated HTML report and log file containing all intercepted requests and responses
(6) Terminal output of the binary (`stdout` and `stderr`)
(7) VM snapshot image that was booted using QEMU which can be used for forensic analysis
(8) VM output log containing boot up and shutdown messages, and commands executed
(9) JSON file containing run details such as timestamps, binary name, command line arguments, QEMU boot command, and pipeline hooks

A.2 CHAOS RAT Functionality

The CHAOS RAT payload is injected typically via phishing or malicious diagnostic utilities. Upon execution, CHAOS RAT establishes a connection to a remote command-and-control (C2) server using hardcoded JSON Web Tokens and custom port configurations. It fingerprints the host by collecting system metadata such as hostname, MAC address, IP address, and OS version, and supports interactive command execution via reverse shell. The RAT's core functionality includes uploading, downloading, and deleting files, enumerating directories, capturing screenshots, and issuing system-level commands such as shutdown and reboot. These features enable an attacker to maintain long-term access and exfiltrate sensitive data from compromised Linux machines. Additionally, the tool's open-source nature allows threat actors to obfuscate configurations, encode communication channels, and evade detection through custom payloads. Golang's versatile compilation allows CHAOS to be built for various architectures and operating systems.

A.3 Kinsing Botnet Functionality

It typically exploits SSH and FTP vulnerabilities to gain unauthorized access and forms a botnet to spread laterally by scanning for additional vulnerable systems. It deploys Monero cryptocurrency miners, establishes persistence by creating scheduled tasks (cron

jobs), and may even install rootkits [4, 20]. It also demonstrates evasive behavior by removing competing malware, cleaning traces of its own presence, and disabling system monitoring services such as SSH to make recovery difficult.