Evasive Ransomware Attacks Using Low-level Behavioral Adversarial Examples

Manabu Hirano
Department of Information and Computer Engineering
National Institute of Technology, Toyota College
Toyota, Japan
hirano@toyota-ct.ac.jp

Ryotaro Kobayashi
Faculty of Informatics
Kogakuin University
Tokyo, Japan
ryo.kobayashi@cc.kogakuin.ac.jp

***Abstract*—Protecting state-of-the-art AI-based cybersecurity defense systems from cyber attacks is crucial. Attackers create adversarial examples by adding small changes (i.e., perturbations) to the attack features to evade or fool the deep learning model. This paper introduces the concept of low-level behavioral adversarial examples and its threat model of evasive ransomware. We formulate the method and the threat model to generate the optimal source code of evasive malware. We then examine the method using the leaked source code of Conti ransomware with the micro-behavior control function. The micro-behavior control function is our test component to simulate changing source code in ransomware; ransomware's behavior can be changed by specifying the number of threads, file encryption ratio, and delay after file encryption at the boot time. We evaluated how much an attacker can control the behavioral features of ransomware using the micro-behavior control function to decrease the detection rate of a ransomware detector.**

***Index Terms*—ransomware, evasion attacks, perturbation, deep learning, behavioral features.**

I. Introduction

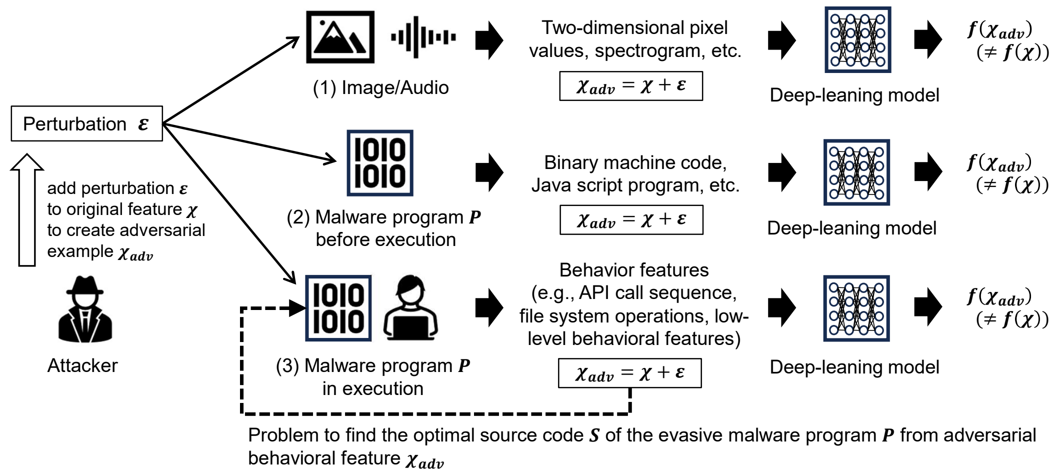
The term *adversarial example* was first introduced by Szegedy et al. [1]; they found that applying an imperceptible non-random perturbation to a test image (i.e., the image is called *adversarial example*) can arbitrarily change the neural network's prediction. As the application of deep learning in cybersecurity increased, the attacks against AI-based defense systems increased. Macas et al. presented a survey of attacks and defenses in deep-learning-based cybersecurity systems from the perspective of adversarial examples [2]; the system includes malware detection, botnet detection, network intrusion detection, fraud detection, and cyber-physical system (CPS) security. In malware detection, they described attacks using adversarial examples of static features obtained from a Portable Executable (PE) file or Android Application Package (APK), an image created from malware binary, and Control Flow Graph (CFG) features. An adversarial example is created using methods such as gradient-based attacks, optimization-based attacks, and generative adversarial network (GAN)-based attacks. This paper focuses on ransomware detection using ransomware's behavioral features.

Ransomware attacks can be detected using indicators obtained from Application Programming Interface (API) and system call monitoring, Input and Output (I/O) monitoring, file system operation monitoring, etc [3]. Machine-learning-based ransomware detection method uses structural features, behavioral features, and both structural and behavioral features [4]; the behavioral features include hardware behavior, file system behavior, network traffic behavior, API call behavior, and a set of all the behavioral features. This paper employs low-level storage and memory access patterns as behavioral features. We first describe the difficulty of creating evasive ransomware against behavioral-based ransomware detectors.

A. Difficulty on generating evasive malware from behavioral adversarial examples

Fig. 1 shows evasion attack scenarios using adversarial examples χ_{adv} on image or audio classifier, static-analysis-based malware detector, and dynamic-analysis-based malware detector. The first example is an image or audio classifier using a deep-learning model f. An attacker aims to cause the deep-learning model to return an incorrect class. The attacker adds subtle perturbation (e.g., slight noise) ϵ that is not recognized by human perception to original image pixel values or audio spectrogram χ to create an *adversarial example* χ_{adv} (= χ + ϵ). A deep-learning model $f(\chi)$ returns a class label for the input χ. Evasion attacks succeed when an attacker inputs *adversarial example* χ_{adv} to the target deep-learning model f and obtains incorrect class label $f(\chi_{adv})$ not equal to the correct class label $f(\chi)$.

The second example is a malware detector using a deep-learning model f. An attacker aims to evade the malware detector by causing the deep-learning model to return an incorrect class (i.e., benign class). The attacker adds a perturbation ϵ to the malware program $\mathcal{P}$ to evade the target malware detector (i.e., deep-learning model) f. Evasion attack succeeds when the attacker inputs *adversarial example* χ_{adv} to the target deep-learning model f and obtained incorrect class $f(\chi_{adv})$ not equals to $f(\chi)$. Hu and Tan presented an evasion attack method named *MalGAN* [5]; they assume the target deep-learning model f uses a 160-dimensional binary vector χ to represent the malware program's static feature. The 160-dimensional binary vector χ consists of a series of binary values $\{0, 1\}$ that represents the existence of 160 representative API calls (i.e., 1 means the API call exists in the malware program, 0 means it does not exist). Since



Perturbation ε
add perturbation ε to original feature χ to create adversarial example χ_{adv}
Attacker
(1) Image/Audio
(2) Malware program P before execution
(3) Malware program P in execution
Two-dimensional pixel values, spectrogram, etc.
Binary machine code, Java script program, etc.
Behavior features (e.g., API call sequence, file system operations, low-level behavioral features)
$\chi_{adv} = \chi + \varepsilon$
Deep-leaning model
$f(\chi_{adv})$ $(\neq f(\chi))$
Problem to find the optimal source code S of the evasive malware program P from adversarial behavioral feature χ_{adv}

Fig. 1: Evasion attack scenarios using adversarial examples χ_{adv} on image/audio classifier, static-analysis-based malware detector, and dynamic-analysis-based malware detector.

deletion of critical API calls affects functions of malware (e.g., deleting *WriteFile* API call affects ransomware's destruction function), *MalGAN* only adds perturbation ϵ to API calls that do not affect the original malware functions. The attacker can create a malware program by adding some binary code that calls the API calls (i.e., the code corresponding to the perturbation ϵ) at the places that do not affect the original function (e.g., slack space) in the original binary program $\mathcal{P}$. Creating an evasive malware program $\mathcal{P}$ against static-analysis-based malware detector using an *adversarial example* $\boldsymbol{\chi_{adv}}$ is relatively easy when attackers can directly insert binary code (i.e., perturbation ϵ) to the original binary program $\mathcal{P}$ without corrupting its functions.

The last example is a behavioral-based malware detector using a deep-learning model f. An attacker's goal is to evade the malware detector by causing the deep-learning model to return an incorrect class (i.e., benign class); this time, the malware detector (i.e., deep-learning model) f uses a behavioral feature χ obtained when the malware is executed. The attackers added perturbation ϵ to the original behavioral feature χ to create an *behavioral adversarial example* $\boldsymbol{\chi_{adv}}$ (= $\chi + \epsilon$). Evasion attack succeeds when the attacker inputs χ_{adv} to the target deep-learning model f and obtained incorrect class $f(\chi_{adv})$ not equals to $f(\chi)$. The attacker's goal is to find the optimal source code $\mathcal{S}_{adv}$ of the evasive malware program $\mathcal{P}_{adv}$ from *behavioral adversarial examples* $\boldsymbol{\chi_{adv}}$.

B. Contribution and organization of this paper

Although creating *behavioral adversarial examples* $\boldsymbol{\chi_{adv}}$ at the feature level is relatively easy, creating an evasive malware program (i.e., source code) that produces the *behavioral adversarial examples* $\boldsymbol{\chi_{adv}}$ is challenging as shown in the example (3) of Fig. 1. This paper's contributions are as follows:

- We formulate a method and threat model in creating evasive ransomware program; an attacker finds the optimal source code $\mathcal{S}$ of evasive ransomware program $\mathcal{P}$ that produces *behavioral adversarial examples* $\boldsymbol{\chi_{adv}}$ to evade the target behavioral-based ransomware detector f.
- We present the *micro-behavior control function* to examine the formulated method in a real-world ransomware sample using the leaked Conti ransomware's source code.
- We evaluated how much an attacker can change behavioral features $\boldsymbol{\chi_{adv}}$ by modifying the ransomware source code. At last, we evaluated the success rate of evasive ransomware attacks.

The rest of this paper is organized as follows. Section II presents the formulation of the evasive ransomware problem using *behavioral adversarial examples*. Section III describes the low-level behavioral features used in this paper. Section IV presents the design and implementation of the *micro-behavior control function* to examine the formulated method. Section V shows how much the *micro-behavior control function* can change the behavioral features; section VI shows how much the *micro-behavior control function* can evade the ransomware detector. Section VII describes the limitations and future work. We conclude the summary in Section VIII.

II. Formulation of Behavioral Adversarial Examples and Threat Model on Evasive Ransomware Problem

We aim to examine how the attackers can change a behavioral feature χ of ransomware to evade the target behavioral-based ransomware detector represented as a deep-learning model f by adding small perturbations ϵ. In this paper, we define *behavioral adversarial examples* $\boldsymbol{\chi_{adv}}$ of ransomware and threat model as follows.

- Attackers can create an approximation of the target ransomware detector (i.e., *substitute* model f) by submitting sufficient queries and receiving responses to learn the input and output mappings. We assume that attackers know what behavioral features the target ransomware detector's deep learning model uses (i.e., *grey box* attack).

When the *substitute* model f became an imitation close enough to the target ransomware detector's deep learning model f_{orig}, attackers can use the *substitute* model f to create adversarial behavioral feature $\boldsymbol{\chi_{adv}}$ by submitting the unlimited number of queries and of responses.

- Attackers' first goal is to find an adversarial behavioral feature $\boldsymbol{\chi_{adv}}$ that is classified as *benign* by the target detector with minimum perturbation $\boldsymbol{\epsilon}$.
- Attackers have access to a source code $\mathcal{S}_{orig}$ of ransomware; Attackers' final goal is to find the optimal source code $\mathcal{S}_{adv}$ that produces a behavioral adversarial feature $\boldsymbol{\chi_{adv}}$. The optimization (i.e., selecting the optimal source code from candidates) is performed by minimizing the ransomware's functional degradation $\mathcal{F}$ and performance degradation $\mathcal{P}$.

In our threat model, attackers can find adversarial behavioral feature $\boldsymbol{\chi_{adv}}$ in (1). We assume the attacker aims to evade a binary classification model (i.e., ransomware or benign class).

$$\boldsymbol{\chi_{adv}} = \boldsymbol{\chi} + \arg\min_{\boldsymbol{\epsilon}} \Big\{ ||\boldsymbol{\epsilon}||_p : f(\boldsymbol{\chi_{adv}}) \neq f(\boldsymbol{\chi}) \Big\} \quad (1)$$

where $\boldsymbol{\epsilon}$ is a tensor representing the minimum perturbation to the original behavioral feature $\boldsymbol{\chi}$ (i.e., $\boldsymbol{\chi_{adv}} = \boldsymbol{\chi} + \boldsymbol{\epsilon}$). $||\boldsymbol{\epsilon}||_p$ is an L^p-norm; it is Euclidean norm when p is 2. A function $f(\boldsymbol{\chi})$ returns a class label (i.e., ransomware or benign); attackers modify original ransomware that produces a behavioral feature $\boldsymbol{\chi}$ to evade the target detector; the modified version of ransomware produces a behavioral adversarial feature $\boldsymbol{\chi_{adv}}$. When $f(\boldsymbol{\chi_{adv}}) \neq f(\boldsymbol{\chi})$ is satisfied, the attacker succeeds in evading the target detector using the modified version of ransomware that is misclassified in benign class; the perturbation $\boldsymbol{\epsilon}$ should be small as possible.

However, unlike conventional adversarial examples problems such as image and audio perturbation, we need to generate the evasive ransomware program $\mathcal{P}_{adv}$ that can be executed on a computer. We need to formulate how to find the optimal source code $\mathcal{S}_{adv}$ of the evasive ransomware program $\mathcal{P}_{adv}$. We first formulate the mapping between a *behavioral adversarial perturbation* $\boldsymbol{\epsilon}$ and source code $\mathcal{S}$ in (2).

$$\boldsymbol{\epsilon}_{i,j} = \boldsymbol{\mathcal{E}}(\mathcal{S}_i, \mathcal{V}_j) \quad (2)$$

where $\boldsymbol{\mathcal{E}}(\mathcal{S}_i, \mathcal{V}_j)$ is a projection function from a source code candidate $\mathcal{S}_i$ and an execution environment $\mathcal{V}_j$ to an adversarial perturbation $\boldsymbol{\epsilon}$. For example, an evasive ransomware program $\mathcal{P}_{i,j}$ created from a source code $\mathcal{S}_i$ produces a behavioral adversarial feature $\boldsymbol{\chi}_{adv}$ (=$\boldsymbol{\chi} + \boldsymbol{\epsilon}_{i,j}$) when it is executed on an environment $\mathcal{V}_j$ (e.g., specifications of computers and network including the number of Central Processing Unit (CPU) cores and network throughput). Although evasive ransomware should have some adaptive mechanism to reproduce behavioral adversarial examples on various computer and network environments, we tested only one computer and network environment (i.e., constant $\mathcal{V}$) in this paper. The advanced adaptive mechanism of evasive ransomware that can be executed on various environments is possible future work.

The optimal source code selection is performed in (3).

$$\mathcal{S}_{adv} = \arg\min_{\mathcal{S}} \Big\{ \alpha\mathcal{F}(\mathcal{S}) + \beta\mathcal{P}(\mathcal{S}, \mathcal{V}) : f(\boldsymbol{\chi} + \boldsymbol{\mathcal{E}}(\mathcal{S}, \mathcal{V})) \neq f(\boldsymbol{\chi}) \Big\} \quad (3)$$

where function $\mathcal{F}(\mathcal{S})$ returns the degradation in the number of functions original ransomware has when attackers introduce perturbation $\boldsymbol{\epsilon}$; the functions include file enumeration and encryption, network drive encryption, data exfiltration, lateral movement, privilege escalation, Command and Control communication. On the other hand, the function $\mathcal{P}(\mathcal{S}, \mathcal{V})$ returns the performance degradation of ransomware (e.g., the number of encrypted files per second, the throughput of data exfiltration) on the execution environment $\mathcal{V}$. Attackers balance functional degradation $\mathcal{F}$ and performance degradation $\mathcal{P}$ by deciding appropriate coefficients α and β in (3) to select the optimal source code from candidates while the minimum condition for a successful evasion attack (1) is satisfied.

The contribution of this paper is to examine a method to find an optimal source code $\mathcal{S}_{adv}$ of ransomware that produces a behavioral adversarial feature $\boldsymbol{\chi_{adv}}$ that meets above requirements. The following section presents the detail of the behavioral feature $\boldsymbol{\chi}$ we used in this paper.

III. Low-level behavioral features used in ransomware detector

We presented a hypervisor-based monitoring system to collect low-level behavioral features for detecting ransomware's destruction phase [6], [7]. In our defense-in-depth approach, we employed the thin hypervisor as an additional protection layer to the conventional OS-level protection layer. Fig. 2 shows the hypervisor-based monitoring system; the low-level behavioral features collected in the hypervisor layer consist of access patterns on storage devices (e.g., Solid State Drive) and memory access patterns on Random Access Memory (RAM). We developed the monitoring functions using a thin hypervisor named BitVisor [8]. The storage access patterns are obtained using an extended Advanced Host Controller Interface (AHCI) para-pass through driver [6], while memory access patterns are obtained using a hardware-assisted memory virtualization technology [7], Intel's Extended Page Table (EPT) [9]; we examined a deep-learning-based ransomware detector trained using the low-level behavioral features. Please refer to our paper [7] for details on the hypervisor-based monitoring system.

Fig. 3 shows graphs of a behavioral feature $\boldsymbol{\chi}$ obtained using the developed hypervisor-based monitoring system; a low-level behavioral feature $\boldsymbol{\chi}$ consists of a storage feature $\boldsymbol{\chi_s}$ and a memory feature $\boldsymbol{\chi_m}$. A storage feature $\boldsymbol{\chi_s}$ consists of the following five-dimensional feature vectors ($\boldsymbol{x_0}$ to $\boldsymbol{x_4}$): entropy of written blocks (top-left graph), write and read throughput in Byte/s (center-top graph), and variance of written and read Logical Block Address (LBA) on storage devices (top-right graph). On the other hand, a memory feature $\boldsymbol{\chi_m}$ consists of the following 18-dimensional feature vectors ($\boldsymbol{x_5}$ to $\boldsymbol{x_{22}}$):

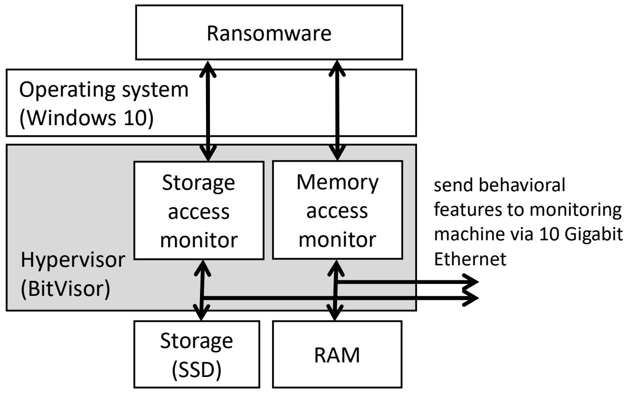

Ransomware
Operating system (Windows 10)
Hypervisor (BitVisor)
Storage access monitor
Memory access monitor
send behavioral features to monitoring machine via 10 Gigabit Ethernet
Storage (SSD)
RAM

Fig. 2: Hypervisor-based monitoring system to collect low-level behavioral features [6], [7].

entropy in write and read-write operations (bottom-leftmost graph), the number of EPT violations (i.e., the number of memory accesses) for 4KiB, 2MiB, and Memory Mapped Input Output accesses of write, read, instruction fetch, and read-write operations (the three bottom-center graphs), and variance of accessed Guest Physical Address (GPA) of write, read, instruction fetch, and read-write operations (bottom-rightmost graph). We counted the number of memory accesses using EPT violation. An EPT violation occurs when a hypervisor does not have the accessed GPA page table entry in the Extended Page Table (EPT). Once the address is translated, the EPT violation never occurs at the same address. Therefore, our system periodically flushes a Translation Lookaside Buffer (TLB) to cause intentional EPT violations. Performance degradation caused by the monitoring system was examined in our previous paper [10], which presented the use of memory access patterns to detect ransomware. We presented the details of the feature engineering in another paper [7], which showed the analysis of our dataset that contains memory and storage access patterns of ransomware.

Fig. 4 shows a structure of a behavioral feature $\boldsymbol{\chi}$ that consists of a storage feature $\boldsymbol{\chi_s}$ and a memory feature $\boldsymbol{\chi_m}$; they are collected in 30 s after executing a ransomware sample (i.e., Conti, Darkside, LockBit, REvil, Ryuk, and WannaCry) or a benign application (i.e., AESCrypt, Firefox, Idle, Office, SDelete, and Zip). The graphs is created using $T_{window} = 1$ s and $T_d = 30$; T_{window} is a period to calculate a feature vector $\boldsymbol{x}$. T_d is the duration of access patterns used in detecting ransomware; therefore, T_d is also referred to as the detection time of ransomware. As we can see in Fig. 4, $\boldsymbol{\chi} = \{\boldsymbol{\chi_s}, \boldsymbol{\chi_m}\} = \{\boldsymbol{x}(0), \boldsymbol{x}(1), \boldsymbol{x}(2), ..., \boldsymbol{x}(29)\}$. Each row represents a feature vector at a specific period; for example, $\boldsymbol{x}(0)$ is a 23-dimensional feature vector calculated using storage and memory access patterns between 0 s and 1 s. On the other hand, a behavioral feature $\boldsymbol{\chi}$ can also be expressed in $\boldsymbol{\chi} = \{\boldsymbol{\chi_s}, \boldsymbol{\chi_m}\} = \{\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_{22}}\}$. Each column represents a specific feature vector between 0 s and 29 s. For example, $\boldsymbol{x_0} = \{\boldsymbol{x_0}(0), \boldsymbol{x_0}(1), \boldsymbol{x_0}(2), ..., \boldsymbol{x_0}(29)\}$ is a feature vector of storage entropy between 0 s and 30 s. $\boldsymbol{x_i}(j)$ is an ith feature vector calculated using access patterns between j s and $(j+1)$ s. The following section describes how we can change behavioral feature $\boldsymbol{\chi}$ by changing the micro-behaviors of ransomware.

IV. Micro-behavior control function of evasive ransomware

In Section I, we described difficulty on generating an evasive malware program $\mathcal{P}$ (i.e., source code $\mathcal{S}$ corresponding to $\mathcal{P}$) that produces *adversarial behavioral feature* $\boldsymbol{\chi}_{adv}$. In Section II, we formulated the optimal source code selection problem in (3). The goal of the problem in (3) is to find the best source code $\mathcal{S}_{adv}$ from candidate source codes $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_\infty\}$. $\mathcal{S}_i$ (= $\mathcal{S}_{orig}$ + $\mathcal{S}_{\epsilon_i}$) is a source code to produce a behavioral adversarial feature $\boldsymbol{\chi}_i$; $\mathcal{S}_{orig}$ is an original source code of ransomware and $\mathcal{S}_{\epsilon_i}$ is a patch to create $\mathcal{S}_i$ from $\mathcal{S}_{orig}$. Since the search space is vast and we do not have generative AI that outputs the optimal source code yet, we first test the method in limited source code space; in this paper, we simulated 24 patterns of source code candidates $\boldsymbol{\mathcal{S}} = \{\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_{23}\}$ using boot options of a modified version of Conti ransomware. The boot options consist of the number of threads, encryption ratio, and delay after file encryption; these parameters are used in our *micro-behavior control function* of evasive ransomware. The micro-behavior control function tested in this paper produces 24 patterns of a behavioral feature $\boldsymbol{\chi}_i$. We examined the changes in the behavioral features and tested the success rates of evasion attacks on the target ransomware detector.

Fig. 5 shows a flow chart of the micro-behavior control function of modified Conti ransomware. The extended parts of Conti ransomware are shown in grey. We control the micro-behaviors of Conti ransomware by changing the following three parameters that are specified at the execution time of ransomware: (1) the number of threads, (2) the encryption ratio per file, and (3) the delay after encrypting a file. We added the source code of the extended parts to the leaked source code of Conti ransomware [12]. The development and test were performed using the isolated lab environment.

Table I shows the computer's specification for developing and testing the micro-behavior control function of Conti ransomware. Fig. 6 shows the changes in the number of encrypted files when we executed the Conti ransomware with the micro-behavior control function; we tested the number of threads between 1 and 8. Although we used the computer with eight logical cores (i.e., eight threads), the number of encrypted files decreased when we specified more than three threads. The original Conti ransomware uses twice the number of logical cores as the number of threads; therefore, the number of threads will be 16 when we execute the original Conti ransomware on the computer with a CPU that supports eight logical cores. The performance degradation in more than three threads on the test machine indicates that some ransomware samples are not correctly optimized for computers of various performances. This study aims to change the micro-behavior of ransomware; we use the number of threads between 1

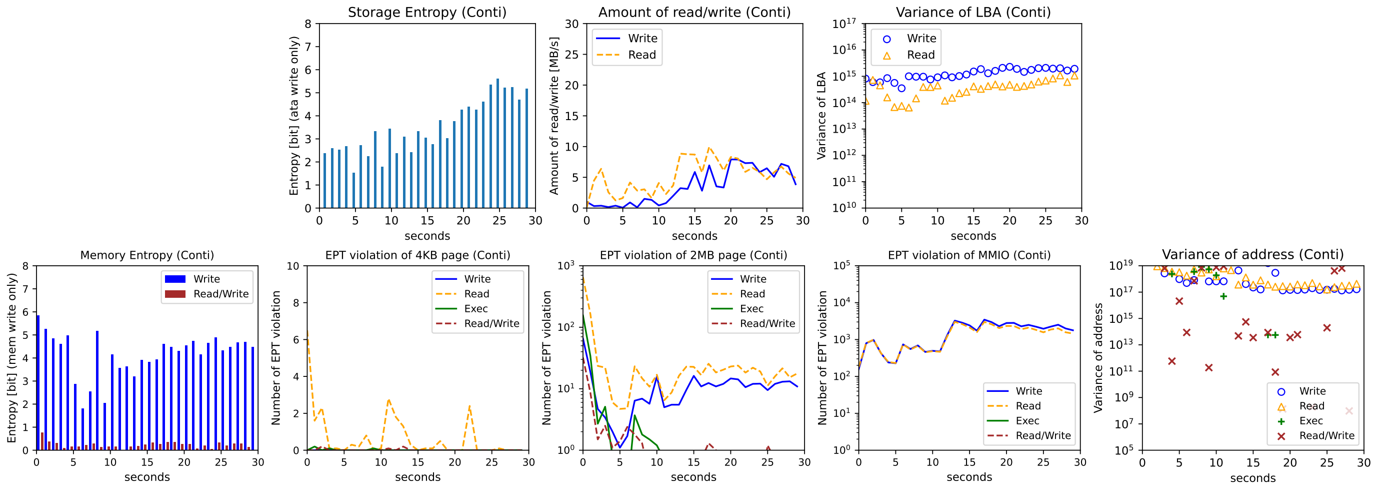


Fig. 3: Example of a storage feature χ_s (top) and a memory feature χ_m (bottom) of Conti ransomware sample in the RanSMAP dataset, open behavioral feature dataset of ransomware storage and memory access patterns [7], [11]. The graphs were created using average behavioral features of 10 trials on the computer with Intel Core i3 and DDR4-2133 16GB RAM.

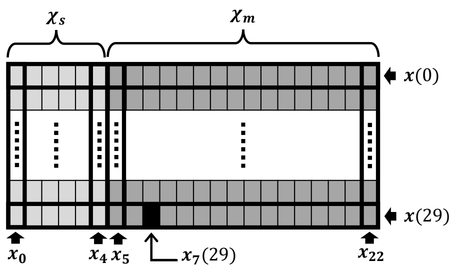


Fig. 4: Structure of a behavioral feature χ consisting of χ_s and χ_m in 30 s after executing a ransomware sample or benign application ($T_{window} = 1$ s and $T_d = 30$ s).

TABLE I: Specification of the computer used in developing the micro-behavior control function of evasive ransomware.

CPU	Intel Core i3 12100
	4 P-cores, 8 threads
RAM	DDR4 2133 8GiB x 2
Motherboard	ASRock B660M-HDV
Solid State Drive	Crucial CT240BX, CT250MX, Samsung 840
Hypervisor	BitVisor downloaded on 25th Jan. 2022 (8c129a1)

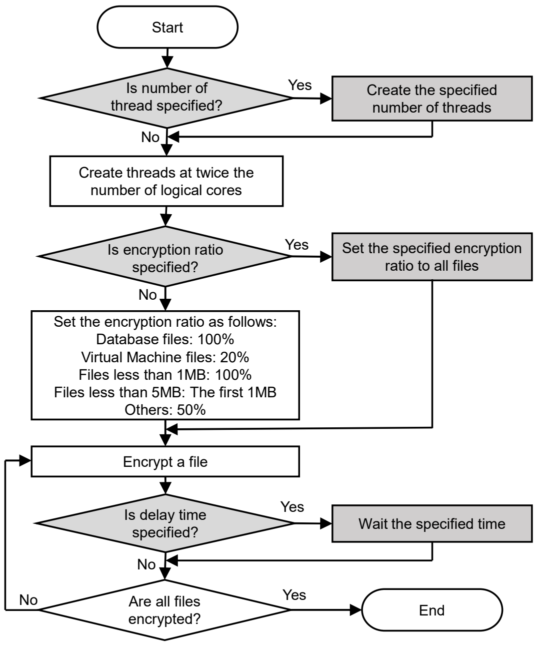

Start
Is number of thread specified?
Yes
Create the specified number of threads
No
Create threads at twice the number of logical cores
Is encryption ratio specified?
Yes
Set the specified encryption ratio to all files
No
Set the encryption ratio as follows:
Database files: 100%
Virtual Machine files: 20%
Files less than 1MB: 100%
Files less than 5MB: The first 1MB
Others: 50%
Encrypt a file
Is delay time specified?
Yes
Wait the specified time
No
No
Are all files encrypted?
Yes
End

Fig. 5: Flow chart of the micro-behavior control function of Conti ransomware. The extended parts are shown in grey.

and 3 in the later section since the number of threads and the number of encrypted files are proportional in the range between 1 and 3 on the test machine. Please note that the behavior depends on the computer on which ransomware is executed (i.e., an environment $\mathcal{V}$); for example, the range will change if we use high-performance file servers with high-grade CPU and storage. In this paper, we tested our micro-behavior control function only on the test machines shown in Table I. Examining the micro-behavior control function of the computers of various performances is possible for future work.

V. Evaluation of Micro-Behavior Contorol Function of Evasive Ransomware

How much can a ransomware author control behavioral feature χ by changing the ransomware's source code? This section presents an analysis of changes in the behavioral feature χ obtained from the modified Conti ransomware with the micro-behavior control function that simulates changes in source code. Fig. 7 shows cosine similarity between adversarial behavioral features χ_{adv} of modified Conti ransomware with the micro-behavior control function. The tested parameters are

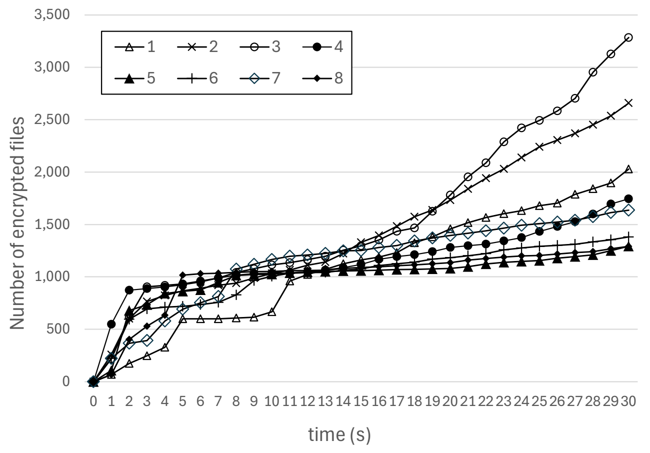


Fig. 6: The number of encrypted files in 30 s when the micro-behavior control function of Conti ransomware was executed using the number of threads between 1 and 8.

$P_{thread} = \{1, 2, 3\}$, $P_{ratio} = \{50\%, 100\%\}$, and $P_{delay} = \{0$ ms, 25 ms, 50 ms, 100 ms$\}$. The number of combinations of parameter P is 24 in total. Please note that we excluded $\boldsymbol{x}_{17}$ and $\boldsymbol{x}_{18}$ since there were no MMIO accesses of instruction fetch and read-write. we calculated cosine similarity between $\boldsymbol{x}_{(i,base)}$ and $\boldsymbol{x}_{(i,k)}$ in (4).

$$Sim(\boldsymbol{x}_{(i,base)}, \boldsymbol{x}_{(i,k)}) = \frac{\boldsymbol{x}_{(i,base)} \cdot \boldsymbol{x}_{(i,k)}}{||\boldsymbol{x}_{(i,base)}|| \, ||\boldsymbol{x}_{(i,k)}||} \tag{4}$$

where $\boldsymbol{x}_{(i,base)}$ is an ith behavioral feature vector $\boldsymbol{x}_i$ of the modified ransomware using the parameter set $base$ that uses $P_{thread} = 3$, $P_{ratio} = 100\%$, and $P_{delay} = 100$ ms; we used the bottom row of Fig. 7 as baseline to calculate all other cosine similarity values. $\boldsymbol{x}_{(i,k)}$ is an ith behavioral feature vector of the modified ransomware using kth parameter set; we use the 24 combinations of parameters (i.e., 24 rows shown in Fig. 7). The numerator is a dot product of two vectors, and the denominator is a product of the L2 norm of each vector.

Fig. 8 shows the average cosine similarity of all the 23-dimensional behavioral features, $\overline{Sim}_{(k,sample)}$, between modified Conti ransomware with the micro-behavior control function and 12 samples of the RanSMAP dataset [7], [11]. We calculated the cosine similarity in (5).

$$Sim(\boldsymbol{x}_{(i,k)}, \boldsymbol{x}_{(i,sample)}) = \frac{\boldsymbol{x}_{(i,k)} \cdot \boldsymbol{x}_{(i,sample)}}{||\boldsymbol{x}_{(i,k)}|| \, ||\boldsymbol{x}_{(i,sample)}||} \tag{5}$$

where $\boldsymbol{x}_{(i,k)}$ is an ith behavioral feature of the modified ransomware using kth parameter set; we use the 24 combinations of parameters (i.e., 24 rows in Fig. 8). $\boldsymbol{x}_{(i,sample)}$ is an ith behaviraol feature of a sample. The samples comprise six ransomware and six benign applications of the RanSMAP dataset. For example, $\boldsymbol{x}_{(0,Conti)}$ is storage entropy feature $\boldsymbol{x}_0$ of Conti ransomware. Please note that the Conti ransomware sample of the RanSMAP dataset (shown in the leftmost column) is not the same sample we used in changing ransomware behavior; instead, we used the leaked Conti source code [12]. An average cosine simiairy $\overline{Sim}_{(k,sample)}$ was calculated in (6).

$$\overline{Sim}_{(k,sample)} = \frac{1}{23} \sum_{i=0}^{22} Sim(\boldsymbol{x}_{(i,k)}, \boldsymbol{x}_{(i,sample)}) \tag{6}$$

VI. Success rate of evasive ransomware attacks

Next, we examined the success rate of evasive ransomware attacks using the micro-behavior control function of modified Conti ransomware. We trained the deep-learning model using behavioral feature $\boldsymbol{\chi}$ from the RanSMAP dataset [7], [11]; we used behavioral features $\boldsymbol{\chi}$ of six ransomware (i.e., Conti, Darkside, LockBit, REvil, Ryuk, and WannaCry) and six benign applications (i.e., AESCrypt, Firefox, Idle, Office, SDelete, and Zip) executed on the computer with Intel Core i3 CPU and DDR4 2133Mhz 16GB RAM since we collected adversarial behavioral features $\boldsymbol{\chi_{adv}}$ using the micro-control function on the computer with the same specification. We used $T_{window} = 0.1$ s and $T_d = 30$ s to create behavioral feature χ. We created five deep-learning models and tested them for each parameter set of P_{thread}, P_{ratio}, and P_{delay} and calculated the sum of each confusion matrix. Recall was used to measure the performance of evasive ransomware attacks since we predicted only positive class (i.e., we input only behavioral features of ransomware with the micro-behavior control function). The recall was calculated in (7).

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

where TP (True Positive) is the number of features in the correct class that are correctly classified. FN (False Negative) is the number of incorrectly classified features that are not in the correct class. Fig. 9 shows a recall of modified Conti ransomware with the micro-behavior control function. In Fig. 9(a) (i.e., encryption ratio of 50%), the highest recall of 0.98 was reduced to 0.72 at a minimum; thus, the micro-behavior control function could reduce recall of 0.26. In Fig. 9(b) (i.e., encryption ratio of 100%), the highest recall of 0.98 was reduced to 0.64 at the minimum; thus, the micro-behavior control function could reduce recall of 0.34.

VII. Discussion

We presented the micro-behavior control function to change the behavior of ransomware; this test component simulates source code changes in (3) to find the optimal source code $\mathcal{S}_{adv}$. The experimental results in Fig. 7 show that some features are easy to change and others are difficult. For example, the behavioral features that were easy to change include storage throughput (read) $\boldsymbol{x}_2$, MMIO accesses (write) on RAM $\boldsymbol{x}_{15}$, MMIO accesses (read) on RAM $\boldsymbol{x}_{16}$, variance of GPA on RAM (write) $\boldsymbol{x}_{19}$. Please note that we excluded 4KiB page accesses since the number of 4KiB accesses was very small. On the other hand, the behavioral features that were difficult to change include storage entropy $\boldsymbol{x}_0$, 2MB page accesses on RAM $\boldsymbol{x}_{11}$, $\boldsymbol{x}_{12}$, $\boldsymbol{x}_{13}$, $\boldsymbol{x}_{14}$, and variance of GPA on RAM (instruction fetch) $\boldsymbol{x}_{21}$.

Number of threads	Encryption ratio per file	Delay after a file encryption in ms	Storage					Memory																	Average
										4KiB Page Accesses				2MiB Page Accesses				MMIO Accesses							
			x_0: Entropy (Write)	x_1: Throughput (Write)	x_2: Throughput (Read)	x_3: Variance of LBA (Write)	x_4: Variance of LBA (Read)	x_5: Entropy (Write)	x_6: Entropy (Read-Write)	x_7: Write	x_8: Read	x_9: Instruction fetch	x_{10}: Read-Write	x_{11}: Write	x_{12}: Read	x_{13}: Instruction fetch	x_{14}: Read-Write	x_{15}: Write	x_{16}: Read	x_{19}: Variance of GPA (Write)	x_{20}: Variance of GPA (Read)	x_{21}: Variance of GPA (Inst. fetch)	x_{22}: Variance of GPA (Read-Write)	
1	50%	0	0.96	0.74	0.59	0.77	0.89	0.80	0.82	0.51	0.95	0.43	0.00	0.94	0.92	1.00	0.98	0.67	0.66	0.85	0.97	0.94	0.83	0.77
		25	0.97	0.84	0.77	0.92	0.87	0.90	0.95	0.13	0.94	0.73	0.00	0.98	0.97	0.95	0.97	0.95	0.96	0.87	0.98	0.96	0.88	0.83
		50	0.98	0.74	0.71	0.87	0.91	0.89	0.95	0.26	0.95	0.50	0.00	0.95	0.98	0.96	0.98	0.93	0.94	0.75	0.97	0.96	0.90	0.81
		100	0.93	0.69	0.68	0.83	0.93	0.90	0.95	0.27	0.94	0.40	0.58	0.99	0.94	0.94	0.96	0.91	0.91	0.88	0.97	0.94	0.91	0.83
	100%	0	0.96	0.82	0.70	0.70	0.83	0.80	0.81	0.66	0.95	0.36	0.00	0.94	0.84	0.96	0.96	0.71	0.70	0.66	0.99	0.91	0.93	0.77
		25	0.96	0.92	0.88	0.91	0.96	0.93	0.94	0.46	0.98	0.71	0.00	0.99	1.00	1.00	1.00	0.97	0.97	0.91	0.99	0.97	0.97	0.88
		50	0.96	0.82	0.76	0.87	0.90	0.91	0.95	0.77	0.97	0.74	0.00	0.99	0.98	0.99	0.98	0.95	0.95	0.83	0.98	0.97	0.93	0.87
		100	0.94	0.78	0.78	0.86	0.87	0.94	0.95	0.77	0.98	0.61	0.33	0.99	0.97	0.98	0.97	0.95	0.95	0.70	0.99	0.98	0.90	0.87
2	50%	0	0.96	0.78	0.72	0.80	0.85	0.78	0.85	0.09	0.95	0.47	0.41	0.88	0.87	0.94	0.93	0.80	0.79	0.84	0.97	0.90	0.85	0.78
		25	0.96	0.76	0.55	0.81	0.75	0.74	0.72	0.06	0.80	0.36	0.15	0.88	0.88	0.93	0.94	0.50	0.49	0.83	0.85	0.91	0.91	0.7
		50	0.94	0.80	0.77	0.84	0.90	0.93	0.94	0.24	0.82	0.40	0.13	0.97	0.97	0.97	0.97	0.93	0.94	0.64	0.98	0.96	0.95	0.81
		100	0.97	0.76	0.63	0.81	0.86	0.92	0.92	0.00	0.94	0.61		0.97	0.96	0.97	0.97	0.87	0.88	0.85	0.91	0.96	0.93	0.83
	100%	0	0.96	0.85	0.79	0.82	0.86	0.79	0.83	0.32	0.93	0.41	0.82	0.93	0.90	0.98	0.97	0.79	0.78	0.84	0.96	0.96	0.81	0.82
		25	0.95	0.79	0.60	0.82	0.86	0.79	0.80	0.02	0.96	0.38	0.52	0.96	0.87	0.94	0.94	0.53	0.52	0.84	0.96	0.89	0.92	0.75
		50	0.96	0.88	0.71	0.87	0.88	0.92	0.88	0.03	0.95	0.29	0.32	0.97	0.95	0.95	0.94	0.87	0.88	0.87	0.86	0.98	0.95	0.80
		100	0.95	0.82	0.71	0.88	0.91	0.89	0.91	0.32	0.74	0.70	0.00	0.97	0.96	0.95	0.94	0.93	0.93	0.82	0.97	0.91	0.93	0.82
3	50%	0	0.96	0.85	0.66	0.84	0.89	0.76	0.89	0.40	0.96	0.76	0.00	0.86	0.90	0.95	0.95	0.75	0.75	0.80	0.60	0.92	0.86	0.78
		25	0.97	0.76	0.49	0.77	0.88	0.74	0.73	0.25	0.96	0.47	0.00	0.91	0.86	0.97	0.95	0.54	0.54	0.83	0.97	0.93	0.78	0.73
		50	0.97	0.80	0.64	0.88	0.86	0.85	0.88	0.37	0.96	0.69	0.00	0.95	0.93	0.95	0.94	0.64	0.63	0.88	0.82	0.94	0.96	0.79
		100	0.95	0.91	0.82	0.89	0.94	0.97	0.95	0.50	0.96	0.36	0.00	0.99	0.96	0.96	0.96	0.95	0.95	0.50	0.97	0.95	0.93	0.83
	100%	0	0.96	0.84	0.70	0.83	0.92	0.78	0.80	0.39	0.97	0.58		0.86	0.91	0.98	0.98	0.72	0.71	0.91	1.00	0.98	0.92	0.84
		25	0.96	0.79	0.62	0.84	0.90	0.82	0.80	0.29	0.93	0.39	0.00	0.96	0.85	0.96	0.95	0.55	0.54	0.92	0.83	0.92	0.80	0.74
		50	0.96	0.83	0.65	0.89	0.91	0.91	0.91	0.54	0.90	0.49	0.33	0.95	0.92	0.95	0.96	0.71	0.70	0.92	0.86	0.82	0.86	0.81
		100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Fig. 7: Cosine similarity between adversarial behavioral features χ_{adv} of modified Conti ransomware with the micro-behavior control function. Cosine similarity is calculated using the parameters of $P_{thread} = 3$, $P_{ratio} = 100\%$, and $P_{delay} = 100$ ms.

Fig. 8 shows how an attacker can use behavioral adversarial example $\boldsymbol{\chi}_{adv}$ to mimic benign applications. For example, behavioral adversarial example $\boldsymbol{\chi}_{adv}$ created using the micro-behavior control function with $P_{thread} = 3$, $P_{ratio} = 100\%$, and $P_{delay} = 25$ ms is similar to the behavioral features of SDelete (i.e., benign secure delete program). Please note that each of the 23-dimensional features is not treated equally in deep learning models; therefore, the average cosine similarity is not directly reflected in the result in Fig. 9. The success rate of evasive ransomware attacks can be confirmed in Fig. 9. The recall was reduced to 0.64 at a minimum; however, the success rate is insufficient to complete attacks; further tests using more parameters (e.g., delay of more than 100 ms) are needed.

We discuss the limitations of the presented micro-behavior control function, the test component to simulate changing the source code of evasive ransomware. We formulated how to find the optimal source code $\mathcal{S}_{adv}$ in (3); since the search space of possible source code for all possibilities is vast, we developed the micro-behavior control function to add perturbations to behavioral features by changing the number of threads that affect file encryption speed, the encryption ratio that affects entropy values of written data, and the delay after file encryption that affects encryption speed. The three control functions reduced recall but were insufficient for reliable evasive attacks. More flexible and automated behavioral manipulation techniques are needed to cover the ample input space of source code. The fundamental problem to be solved to generate an evasive ransomware's source code is to find reverse mapping from a behavioral adversarial perturbation $\boldsymbol{\epsilon}$ to a source code $\mathcal{S}$ formulated in (2); the adversarial perturbation $\boldsymbol{\epsilon}$ corresponds to a patch (i.e., diff) to create the target source code $\mathcal{S}_{adv}$ from the original source code $\mathcal{S}_{orig}$.

VIII. Conclusion

Recently, AI-based cybersecurity defense systems have been attacked using adversarial examples; from the defender's perspective, developing a defense system resistant to adversarial examples is crucial. In particular, this paper focused on "behavioral" adversarial examples to evade ransomware detectors. We formulated an evasive ransomware attack and examined it using the test component named the micro-behavior control function in Conti ransomware. We tested the evasive attack on the deep-learning-based ransomware detector. The presented attack reduced recall from 0.98 to 0.64; however, the current prototype cannot cover the vast source code space for reliable evasive attacks. Future work includes a more flexible and automated source code generation method that meets the requirements we formulated in this paper; a reliable defense mechanism against the presented evasive attacks will be needed.

Number of threads	Encryption ratio per file	Delay after a file encryption in ms	Ransomware: Conti	Darkside	LockBit	REvil	Ryuk	WannaCry	Benign: AESCrypt	Firefox	Idle	Office	SDelete	Zip
1	50%	0	0.78	0.68	0.76	0.76	0.66	0.82	0.73	0.74	0.70	0.60	0.78	0.75
		25	0.72	0.60	0.72	0.75	0.73	0.78	0.70	0.68	0.62	0.61	0.75	0.70
		50	0.68	0.57	0.66	0.72	0.73	0.73	0.63	0.66	0.61	0.60	0.72	0.67
		100	0.71	0.58	0.72	0.74	0.72	0.78	0.66	0.67	0.61	0.61	0.74	0.70
	100%	0	0.78	0.67	0.76	0.75	0.67	0.82	0.73	0.74	0.69	0.62	0.78	0.80
		25	0.75	0.61	0.70	0.73	0.71	0.78	0.69	0.71	0.62	0.60	0.73	0.74
		50	0.72	0.60	0.67	0.73	0.69	0.76	0.68	0.69	0.62	0.61	0.71	0.75
		100	0.71	0.57	0.67	0.72	0.71	0.77	0.65	0.66	0.59	0.59	0.70	0.70
2	50%	0	0.80	0.69	0.79	0.76	0.72	0.82	0.77	0.73	0.69	0.62	0.80	0.79
		25	0.79	0.67	0.79	0.73	0.71	0.78	0.74	0.72	0.68	0.62	0.77	0.74
		50	0.73	0.59	0.72	0.72	0.72	0.76	0.70	0.68	0.63	0.60	0.72	0.69
		100	0.73	0.61	0.72	0.75	0.77	0.77	0.70	0.68	0.62	0.65	0.79	0.70
	100%	0	0.81	0.68	0.78	0.78	0.70	0.84	0.75	0.74	0.69	0.61	0.77	0.76
		25	0.82	0.68	0.85	0.75	0.69	0.80	0.74	0.76	0.68	0.63	0.82	0.72
		50	0.77	0.63	0.78	0.78	0.72	0.84	0.72	0.71	0.69	0.62	0.76	0.75
		100	0.73	0.61	0.71	0.74	0.72	0.78	0.69	0.66	0.62	0.61	0.76	0.75
3	50%	0	0.82	0.69	0.75	0.79	0.71	0.83	0.75	0.72	0.69	0.63	0.79	0.76
		25	0.78	0.68	0.78	0.73	0.65	0.80	0.72	0.72	0.70	0.59	0.77	0.76
		50	0.76	0.64	0.78	0.72	0.68	0.75	0.72	0.73	0.64	0.65	0.79	0.76
		100	0.69	0.58	0.70	0.72	0.74	0.75	0.65	0.67	0.61	0.60	0.73	0.67
	100%	0	0.87	0.72	0.83	0.82	0.72	0.89	0.80	0.79	0.74	0.64	0.81	0.83
		25	0.79	0.69	0.77	0.71	0.65	0.79	0.74	0.76	0.70	0.61	0.82	0.72
		50	0.74	0.64	0.74	0.73	0.68	0.77	0.68	0.73	0.66	0.63	0.76	0.70
		100	0.77	0.62	0.69	0.73	0.69	0.79	0.69	0.71	0.60	0.60	0.73	0.76

Fig. 8: Average cosine similarity of 23-dimensional behavioral features between adversarial behavioral features χ_{adv} of modified Conti ransomware with the micro-behavior control function and 12 samples of the RanSMAP dataset [7], [11].

		Delay (ms)			
		0	25	50	100
Thread	1	0.72	0.90	0.98	0.98
	2	0.84	0.90	0.98	0.94
	3	0.84	0.74	0.84	0.98

(a) Encryption ratio of 50%

		Delay (ms)			
		0	25	50	100
Thread	1	0.64	0.96	0.98	0.98
	2	0.76	0.82	0.94	0.98
	3	0.90	0.84	0.82	0.84

(b) Encryption ratio of 100%

Fig. 9: Recall of ransomware detector in binary classification (i.e., ransomware or benign) using modified Conti ransomware with the micro-behavior control function. The deep-learning model was trained using the RanSMAP dataset [7], [11].

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP23H03396 and JSPS KAKENHI Grant Number JP23K11114. The authors gratefully acknowledge constructive comments by the anonymous reviewers. The authors thank Syuya Kawai, Kaito Motokawa, and Reishi Kondo for their support in constructing the dataset. The authors gratefully thank the developers of BitVisor [8].

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[2] M. Macas, C. Wu, and W. Fuertes, "Adversarial examples: A survey of attacks and defenses in deep learning-enabled cybersecurity systems," *Expert Systems with Applications*, vol. 238, p. 122223, 2024.

[3] K. Begovic, A. Al-Ali, and Q. Malluhi, "Cryptographic ransomware encryption detection: Survey," *Computers & Security*, vol. 132, p. 103349, 2023.

[4] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022.

[5] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," in *International Conference on Data Mining and Big Data*. Springer, 2022, pp. 409–423.

[6] M. Hirano, R. Hodota, and R. Kobayashi, "RanSAP: an open dataset of ransomware storage access patterns for training machine learning models," *Forensic Science International: Digital Investigation*, vol. 40, p. 301314, 2022.

[7] M. Hirano and R. Kobayashi, "RanSMAP: Open dataset of Ransomware Storage and Memory Access Patterns for creating deep learning based ransomware detectors," *Computers & Security*, vol. 150, 2025, 104202.

[8] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai *et al.*, "BitVisor: A Thin Hypervisor for Enforcing I/O Device Security," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE 2009)*, 2009, pp. 121–130.

[9] Intel Corporation, "Intel® 64 and IA-32 architectures software developer's manual," *Volume 3C: system programming guide, part 3*, 2021.

[10] M. Hirano and R. Kobayashi, "Machine learning-based ransomware detection using low-level memory access patterns obtained from live-forensic hypervisor," in *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2022, pp. 323–330.

[11] M. Hirano, "Open repository of the RanSMAP dataset," https://github.com/manabu-hirano/RanSMAP/, 2024, accessed 5 February 2025.

[12] S. Alzahrani, Y. Xiao, and W. Sun, "An analysis of conti ransomware leaked source codes," *IEEE Access*, vol. 10, pp. 100 178–100 193, 2022.