Forensic Science International: Digital Investigation 40 (2022) 301314

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

RanSAP: An open dataset of ransomware storage access patterns for training machine learning models

Manabu Hirano [a, *], Ryo Hodota [a], Ryotaro Kobayashi [b]

[a] 2-1 Eisei, Toyota, Aichi, Japan
[b] 1-24-2 Nishi-shinjuku, Shinjuku, Tokyo, Japan

ARTICLE INFO

Article history:
Received 6 April 2021
Received in revised form
25 October 2021
Accepted 13 November 2021
Available online 16 December 2021

Keywords:
Ransomware
Open dataset
Storage access pattern
Machine learning
Hypervisor

ABSTRACT

Ransomware, the malicious software that encrypts user files to demand a ransom payment, is one of the most common and persistent threats. Cyber-criminals create new ransomware variants to evade protections shortly after anti-virus software vendors updated their signature (e.g., static feature obtained from binaries) database. Therefore, many ransomware detection systems today begin to employ behavioral features, or dynamic features, in addition to static features. However, even though ransomware detection using dynamic features can deal with ransomware variants, it has the following limitations: (1) it requires the ransomware to be executed, (2) ransomware may behave differently in a real environment that differs from the controlled environment, and (3) a ransomware sample can become deactivated when command and control (C&C) servers are taken down; hence, they make it impossible to compare multiple detection systems proposed by researchers under identical conditions.

To address the limitations, we present RANSAP, our new open dataset of ransomware storage access patterns. The dataset is currently available in a public repository. To our best knowledge, the dataset is one of the few open datasets consisting of dynamic features of ransomware.

Our new open dataset includes storage access patterns of 7 significant ransomware samples and 5 popular benign software samples on various types and conditions of storage devices. Moreover, the dataset provides access patterns of ransomware variants, those on a different version of an operating system, and those on storage devices with a full drive encryption function enabled. We first present a hypervisor-based monitoring system of storage access patterns followed by a design and an implementation of a feature extractor and machine learning models for ransomware detection. Next, a detailed analysis and evaluation of our dataset are presented. Finally, limitations of our new dataset, comparison with other dynamic analysis methods, state-of-the-art ransomware detection, and future research direction are presented.

1. Introduction

Crypto-ransomware is a type of malware that encrypts victims' files and demands a ransom payment. Over recent years, ransomware has become a powerful weapon of large-scale destruction. Most of the ransomware attacks today include double extortion attacks (Penta Security, 2020); attackers encrypt not only the victims' files but also release a copy of the victims' data to the public. Such large-scale ransomware attacks have become significant threats to the public and private sectors. We, therefore, need effective countermeasures against ransomware attacks.

Typical ransomware attacks are performed as follows. Cyber-criminals first deliver ransomware to a victim's computer using several techniques, including an attachment file and a malicious link in a phishing email, to evade the perimeter protections of the organization. Once a computer in the target organization is infected with the ransomware, it first contacts a command and control (C&C) server to manage encryption keys, encrypts files on the victim's computer, and displays a ransom note on the screen. In addition, many ransomware strains have self-propagation mechanisms; for example, WannaCry, exploited bugs on Windows' Server Message Block (SMB) protocol and quickly spread over victims' local area networks.

Like malware detection, ransomware detection is categorized into two approaches: misuse and anomaly (Al-rimy et al., 2018). Anomaly detection learns a system's normal behavior and alarms any deviations as an anomaly. However, it causes a high false alarm

* Corresponding author.
E-mail address: hirano@toyota-ct.ac.jp (M. Hirano).

https://doi.org/10.1016/j.fsidi.2021.301314

rate; therefore, today's malware detection systems employ misuse detection. The misuse detection is further divided into two categories: static and dynamic. Static features of ransomware are mainly obtained from binary files; thus, we can detect ransomware before it runs. Furthermore, the detection using the static features is fast and accurate in identifying previously known ransomware. It is, however, difficult to detect ransomware binaries that leverage obfuscation techniques.

On the other hand, ransomware's dynamic or behavioral features are obtained during test execution in a controlled environment. Although the detection using dynamic features can deal with variants, it has the following limitations: (1) it requires the ransomware to be executed, (2) ransomware may behave differently in a real environment that differs from the controlled environment (Al-rimy et al., 2018), and, (3) ransomware sample can become deactivated when command and control (C&C) servers are taken down; especially, the reason (2) and (3) make it impossible to compare multiple detection methods proposed by researchers under identical conditions. To address the limitations, we need standard and realistic open datasets that include dynamic features of ransomware.

1.1. Contributions of this paper

In this paper, our new open dataset named RANSAP, a collection of dynamic features of ransomware storage access patterns, is presented. Encryption and file operations of ransomware cannot be hidden because they are the most important means to demand ransom. We have, therefore, decided to collect behavioral features of storage access patterns for our new dataset. The dataset is currently available in a public repository (Hirano, 2021). To our best knowledge, the proposed dataset is one of the few open datasets consisting of dynamic features of ransomware. The contributions of our research are as follows:

- We offer an open dataset consisting of storage access patterns of 7 well-publicized ransomware samples and 5 benign software on various types of and conditions of storage devices. The dataset can be used, for example, by researchers to train a machine-learning-based classifier.
- The dataset also includes access patterns of ransomware variants, those on a different version of an operating system, and those on storage devices with a full drive encryption function enabled. We show detection performance of them and discuss how practical our behavioral based approach is. This dataset can be used to analyze, for example, the differences in access patterns of ransomware families.
- While many modern ransomware detection systems obtain dynamic features from an operating system layer, our new dataset consists of the behavioral features obtained from a thin hypervisor layer. Our collection method aims to minimize unnecessary observer effects and preserve original access patterns as much as possible. The main contribution of this research is to answer the research question: such low-level storage access patterns can discriminate ransomware?

1.2. Relationship with previous studies

Hirano et al. have presented a machine-learning-based ransomware detection system using storage access patterns (Hirano and Kobayashi, 2019) and the storage access patterns were collected using a light-weight hypervisor originally presented in (Hirano et al., 2017). The previous papers (Hirano and Kobayashi, 2019) provided detection results using storage access patterns of only two ransomware samples (i.e., WannaCry and TeslaCrypt) and one benign application (i.e., Zip program) on only a single condition. As a result, the effectiveness and robustness of the detection method were not sufficiently evaluated. Although the total number of CSV files in the previous study was only 60, that of CSV files in our newly created open dataset is 1495.

Taking advantage of the variety of storage access patterns on the new dataset, we present the new research direction on ML-based ransomware detection using low-level behavioral features. We, in particular, offer the novel analysis reports of the following storage access patterns: the 7 significant ransomware samples, including the latest ransomware strains such as Darkside and Sodinokibi (REvil), and the 5 major benign applications, including encryption, secure delete, Zip, Excel, Firefox, under various conditions such as the number of decoy files, the difference in an operating system, storage-level encryption, and types and partition size of storage devices.

This paper newly provides limitations of our dataset, comparison with other datasets and other dynamic analysis methods, use cases of the proposed dataset and detection method, discussions on state-of-the-art ransomware and malware detection, including evasion techniques and adversarial ML attacks, and future research direction on our novel low-level behavioral features.

1.3. Structure of this paper

The rest of this paper is structured as follows. In Section 2, we provide an overview of our new dataset, including how storage access patterns of ransomware were collected, the structure of the dataset, and types of storage devices and conditions of decoy files used in the creation of the dataset. In Section 3, we present the contents of the RANSAP dataset, including the detail of ransomware samples and benign software samples, those of ransomware variants, those on a different version of an operating system, and those on storage devices with a full drive encryption function enabled. In Section 4, we present a design and an implementation of a feature extractor, visualization of the dataset using a dimensionality reduction algorithm, and a prototype of a ransomware detection system using machine learning algorithms. In Section 5, we illustrate the similarity on access patterns of ransomware and those of benign software under various conditions with confusion matrices. Section 6 presents comparisons between the proposed hypervisor-based method and OS-centric methods through experiments and shows how the proposed method complements the weakness of the conventional OS-centric methods. In Section 7, the numbers of I/O requests in more realistic environments are examined; performance degradation of the classifier under complex conditions is discussed. Limitations of the proposed dataset are discussed in Section 8. In Section 9, we discuss the availability of open malware datasets of behavioral futures and comparison with other dynamic analysis methods. Use cases of the proposed dataset and detection method are then presented in Section 10. We compare our dataset and detection method with state-of-the-art papers in Section 11 and show future research direction in Section 12. Finally, concluding remarks are presented in Section 13.

2. Overview of the RANSAP dataset

Our new dataset, a collection of time-series storage access patterns for ransomware researchers, is currently available in a public repository (Hirano, 2021). In this section, we first explain how we have collected storage access patterns of ransomware; the setup and procedure used in creating the dataset are presented, focusing on our hypervisor-based monitoring system. Second, a format of CSV files and directory naming conventions of the dataset

are described. Finally, detailed information of the dataset, such as the number of CSV files, hash values of ransomware samples, types of storage devices, and conditions of decoy files, are provided.

2.1. Hypervisor-based monitoring system

Fig. 1 illustrates a schematic diagram of two machines used in the creation of the dataset. The hypervisor layer (grey box) of the test machine, a customized version of BitVisor (Shinagawa et al., 2009), collects input and output operations issued by a ransomware sample or by a benign software sample. The monitoring system works as follows: BitVisor is first booted from a write-protected USB flash drive protected from being overwritten by ransomware. After BitVisor was booted, an operating system is next booted from a test serial Advanced Technology Attachment (ATA) device (i.e., a hard disk drive or a solid state drive). We then execute a ransomware sample or a benign software sample on the operating system. Next, the developed Advanced Host Controller Interface (AHCI) driver of BitVisor begins to intercept low-level input and output operations (i.e., sector-based read and write accesses) via direct memory access (DMA) protocols. Finally, the access patterns are sent to the monitoring machine via User Datagram Protocol (UDP) packets on the 10 Gigabit Ethernet; the received access patterns are saved in a single Comma Separated Value (CSV) file per execution. We replaced the serial ATA device with the new one at every execution because ransomware overwrites test drives.

The reason why the authors have employed the lightweight hypervisor to collect storage access patterns instead of other popular hypervisors are as follows. BitVisor (Shinagawa et al., 2009), a lightweight hypervisor specialized in monitoring input and output devices, employs Virtual Machine Extensions (VMX) instructions of Intel CPU (Intel Corporation, 2016) to create a thin virtualization layer on a computer. Although most hypervisors create multiple virtual machines to provide multi-tenant cloud services and improve physical hardware utilization, BitVisor creates only one virtual machine to create a lightweight and stealthy surveillance layer. This single virtual machine model makes it possible to collect and monitor the behavioral features of applications without modifying operating system components. Moreover, many hypervisors virtualize storage devices such as hard disk drives (HDDs) and solid state drives (SSDs) using virtual disk files. In contrast, BitVisor does not create virtual disk files and processes input and output operations directly on a physical storage device. This design can preserve original features of storage access patterns with the lowest overhead (Hirano et al., 2017); we, therefore, employ BitVisor to develop our monitoring system. While the developed monitoring system does not collect OS-level behavioral features such as file names, file extensions, timestamps (e.g., modified, accessed, and created time), it collects only low-level storage access patterns. Thus, one of the challenges of this study is to investigate how effective our method is in detecting ransomware and its variants under various conditions without using the semantic information that depends on the operating system.

Table 1
Specification of the test machine used in collecting storage access patterns.

CPU	Intel Cerelon G3920 (2.9 GHz)
RAM	DDR4-2133 8 GiB
Motherboard	ASRock H110M-HDV
Network	Intel Pro1000 (GbE), Intel X550-T1 (10 GbE)
SSD	Crucial CT250MX500SSD1 250 GB
HDD	Seagate ST250DM000 250 GB

Table 1 shows the specification of the test machine used in collecting storage access patterns. The serial ATA devices (i.e., HDD and SSD) were formatted in New Technology File System (NTFS) with GUID partition tables (GPT). In addition, every operating system was installed in Unified Extensible Firmware Interface (UEFI) mode. Both the HDDs and SSDs of 250 GB were formatted in two different partition sizes of 120 GB and 250 GB.

Fig. 2 illustrates the flow chart in the creation of the dataset. While most popular hypervisors, such as VMware, Hyper-V, Virtual Box, virtualize storage devices as files, BitVisor does not use such virtual disk files. We, therefore, employ a drive duplicator to overwrite the entire contents of a physical drive with the initial content before every execution of a sample to ensure that every trial is done under an identical condition.

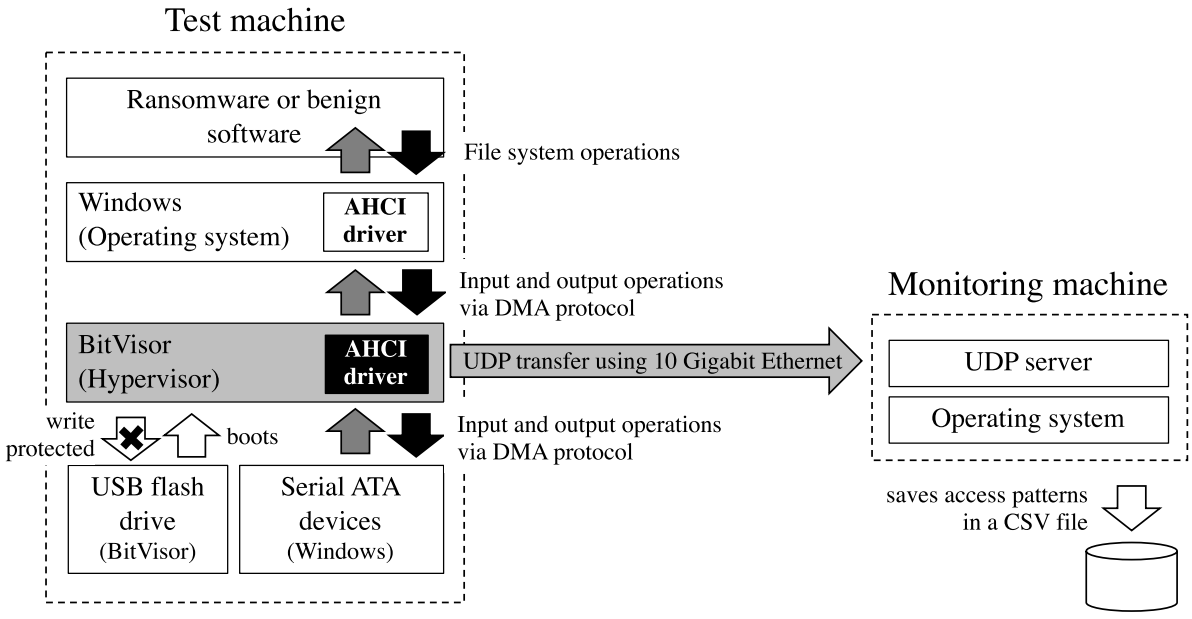


Fig. 1. Schematic diagram of the developed hypervisor-based monitoring system.

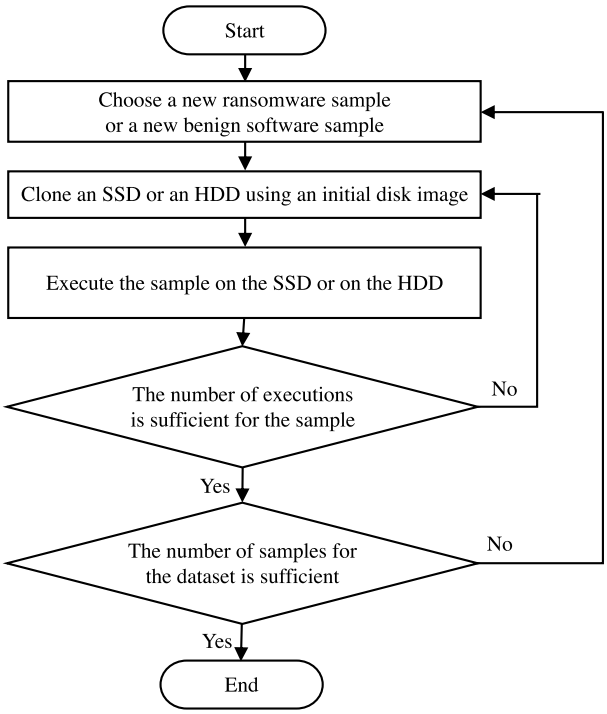


Fig. 2. Flow chart in the creation of the dataset.

2.2. Directory structure and format

The directory structure of our new dataset is presented in Table 2. The directory original includes CSV files of storage access patterns of the 7 ransomware samples and the 5 benign software samples. The directory extra includes CSV files of access patterns of ransomware variants, those on a different version of an operating system, and those on storage devices with a full drive encryption function enabled.

A subdirectory name indicates a sample's name followed by a suffix shown in Table 3. For example, storage access patterns of WannaCry on a solid state drive (SSD) of 120 GB are stored in/original/win7-120gb-ssd/WannaCry-w10dirs. The suffix -w10dirs indicates that the access patterns were obtained under a condition (b). In condition (b), there are 10 directories, and each directory contains about 1000 decoy files (9872 decoy files in total) on the desktop. The detail of the conditions will be explained in Section 2.3.

Two CSV files are created at each execution of a ransomware sample or that of a benign software sample; The format of a CSV file of read accesses (ata_read.csv) is as follows:

- Timestamp [s]
- Timestamp [μs]
- Logical Block Address (LBA) of a read sector
- Size of a block accessed by a sample [byte]

The format of a CSV file of write accesses (ata_write.csv) is as follows:

- Timestamp [s]
- Timestamp [μs]
- Logical Block Address (LBA) of a written sector
- Size of a block accessed by a sample [byte]
- Normalized Shannon entropy of a written sector (0–1)

A single line in a CSV file represents one read access or one write access. For example, a line of "1587971915, 93484727, 16616152, 4096, 0.1676553996446963″ indicates the following access. The first two columns indicate that the date of write access was "Mon Apr 27, 2020, 07:18:35″ and "93,484,727 μs". The third column indicates that the logical block address of the written sector was 16,616,152. The fourth column indicates that the size of a written block was 4096-byte.[1] Finally, the fifth column indicates that the normalized Shannon entropy was approximately 0.1676.

Shannon entropy is a metric to measure uncertainty in a series of bit patterns (Shannon, 1948). Ransomware encrypts a large amount of victims' files in a short period of time and the encryption operations increases entropy of a written sector on a storage device; and, hence, Shannon entropy will be one of the most important features that indicates activities of ransomware. $H(s)$ is a Shannon entropy of a single written sector s, and it is calculated as follows:

$$H(s) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i) \quad (1)$$

where $p(x_i)$ is a probability of a byte x_i, which is an ith byte in a sector s, and n is a size of the sector s in byte. Major sector size n of off-the-shelf storage devices are 512-byte and 4096-byte. We employed storage devices using the 512-byte sector in the creation of the dataset. $H(s)$ produces a value between 0 and 8, where 8 represents a perfectly even distribution of byte values, where 0 represents a sequence of the same byte values. The Shannon entropy in the dataset is finally normalized between 0 and 1.

2.3. Conditions of decoy files

We collected storage access patterns under the following 3 conditions: (a) a clean installation of Windows operating system without any decoy files on the desktop, (b) addition of 10 directories to the desktop of the condition (a); each directory contains about 1000 files in various size (9872 files and 5.2 GiB in total),

Table 2
Structure of the dataset.

Directory name	Category	Section
original/win7-120gb-hdd/ original/win7-250gb-hdd/ original/win7-120gb-ssd/ original/win7-250gb-ssd/	Ransomware and benign software	3.1
extra/win7-250gb-ssd/	Ransomware variants	3.2
extra/win2008r2-250gb-ssd/	Different version of OS	3.3
extra/win7ent_bl-250gb-ssd/	Full drive encryption	3.4

Table 3
Suffixes of sub directories in 3 conditions; the numbers of decoy files, and top 5 file-types.

Condition	Suffix	Numbers of decoy files and top 5 file-types
(a)	(None)	There are no decoy files
(b)	-w10dirs	9872 decoy files in various sizes pdf (2471 files), html (2457 files) txt (1558 files), doc (810 files) ppt (681 files)
(c)	-largefiles	605 large decoy files in large size ppt (201 files), txt (181 files), xls (90 files), doc (74 files), ps (38 files)

[1] The developed system records write and read accesses per 4096-byte, and then divides them into eight 512-byte units.

and (c) addition of a single directory containing relatively large 605 files between 10 MiB and 100 MiB (12.5 GiB in total) to the desktop of the condition (a).

We have employed the Govdocs1 dataset (Garfinkel et al., 2009), which are available in the open repository (Garfinkel, 2010), to create the decoy files of the condition (b) and those of the condition (c). The Govdocs1 dataset consists of various files, including PDF, JPEG, HTML, Microsoft Office, collected from U.S. government websites. The decoy files of condition (b) are the first 10 directories of the Govdocs1 dataset, and the names of the 10 directories are "000", "001", …, and "009". Table 3 also shows the numbers of the top 5 file-types in the condition (b). The decoy files of condition (c), the large files between 10 MiB and 100 MiB, were selected from all the files in the Govdocs1 dataset. Table 3 shows the numbers of the top 5 file-types in the condition (c).

Fig. 3 shows the file size distribution of the condition (b) and (c). The graph is a logarithmic scale with base 2 in the X-axis (i.e., size of a decoy file). In condition (b), most of the decoy files are distributed around 32 KiB. In contrast, condition (c)'s decoy files are mainly distributed around 8 MiB. Although the number of decoy files of condition (b) is greater than that of condition (c), the total size of decoy files of condition (b) (5.2 GiB) is smaller than that of condition (c) (12.5 GiB). We will examine the effects of decoy files' number and size in detecting ransomware in the later section.

3. The collection

In this section, we explain the categories of storage access patterns included in the RANSAP dataset.

3.1. Ransomware and benign software

We have obtained well-publicized 7 ransomware samples from ANY.RUN (ANY.RUN). The SHA-256 hash values of the samples and the first submission date at VirusTotal (Chronicle) are presented in Table 4. Every sample was confirmed that it works properly (i.e., it encrypts user files) in our controlled environment without an Internet connection, in other words, without active command and control (C&C) servers.

Table 5 shows the numbers of the CSV files of the 7 ransomware samples. Every single CSV file contains access patterns of a single execution of a sample. Each of the CSV files includes access patterns at least in 90 s immediately after a ransomware sample was executed. For example, the CSV files of Cerber, Ryuk, and Sodinokibi include access patterns in 180 s, and those of GandCrab4 and Darkside include access patterns in 300 s because these ransomware samples do not heavily encrypt files immediately after its execution. All the 7 ransomware samples shown in Table 5 were executed on Windows 7 Professional 64 bit. We will analyze how the storage types (i.e., HDD and SSD) and the partition sizes (i.e., 120 GB and 250 GB) of test storage devices affect storage access patterns of ransomware in the later section.

We have collected access patterns of the 5 benign software samples that share some behavioral features with ransomware. Table 6 presents the numbers of the CSV files of the 5 benign software samples. The AESCrypt program is an open-source program to encrypt files using the Advanced Encryption Standard (AES) symmetric encryption algorithm (Packetizer, Inc.). The Zip program is a default data compression program of Windows 7 Professional 64 bit. SDelete, a secure delete program included in the Microsoft Sysinternals suite, securely deletes files, directories, partitions, and disks using the DOD 5220.22-M data sanitizing standard developed by the U.S. Department of Defense to prevent unpredictable data recovery by an attacker (Russinovich and Margosis, 2011). Microsoft Excel is a spreadsheet application, and Firefox is a popular open-source web browser program. Access patterns of each of the 5 benign software were collected in 90 s immediately after the benign software sample was executed. They were saved in a single CSV file per execution. The AESCrypt, Zip, and SDelete programs processed (i.e., encrypts, compresses, and

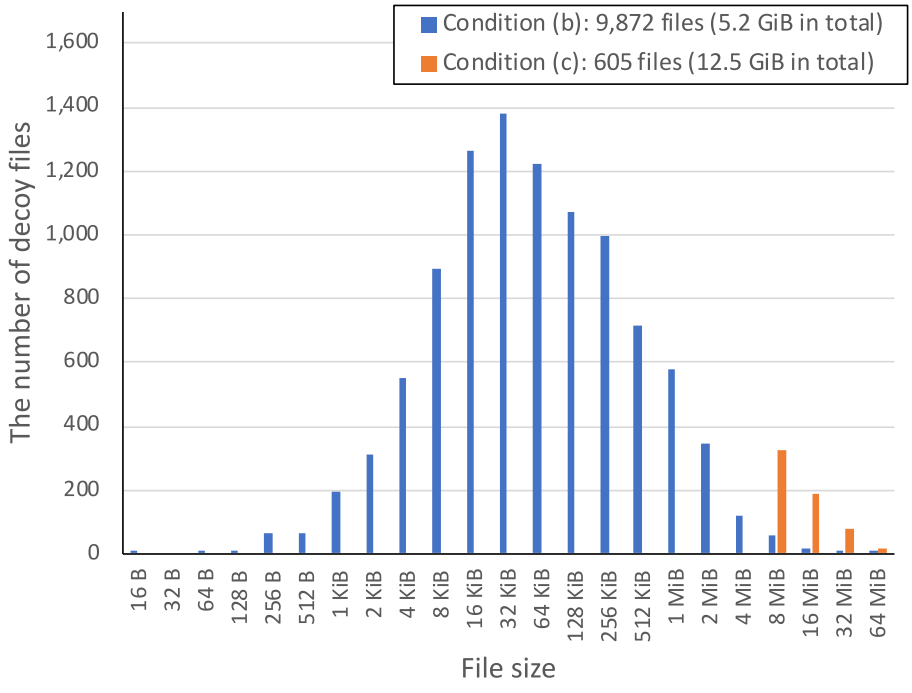


Fig. 3. File size distribution of the condition (b) and (c).

Table 4
Ransomware samples: SHA-256 hash values and the first submission dates at VirusTotal.

Name	SHA-256 hash	Date
TeslaCrypt	9b462800f1bef019d7ec00098682d3ea7fc60e6721555f616399228e4e3ad122	2015–07
Cerber	e67834d1e8b38ec5864cfa101b140aeaba8f1900a6e269e6a94c90fcbfe56678	2017–05
WannaCry	ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa	2017–05
GandCrab v4	c9941b3fd655d04763721f266185454bef94461359642eec724d0cf3f198c988	2018–08
Ryuk	23f8aa94ffb3c08a62735fe7fee5799880a8f322ce1d55ec49a13a3f85312db2	2018–08
Sodinokibi	0fa207940ea53e2b54a2b769d8ab033a6b2c5e08c78bf4d7dade79849960b54d	2019–04
Darkside	b6855793aebdd821a7f368585335cb132a043d30cb1f8dccceb5d2127ed4b9a4	2021–04

Table 5
The numbers of the CSV files in the dataset - ransomware.

OS	Windows 7 Professional SP1 64bit											
Storage	HDD						SSD					
	120 GB			250 GB			120 GB			250 GB		
Condition	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
TeslaCrypt	10	10	10	10	10	10	10	10	10	10	10	10
Cerber	10	10	10	10	10	10	10	10	10	10	10	10
WannaCry	10	10	10	10	10	10	10	10	10	10	10	10
GandCrab v4	10	10	10	10	10	10	10	10	10	10	10	10
Ryuk	10	10	11	10	11	11	10	10	10	10	10	10
Sodinokibi	10	10	10	10	10	10	10	10	10	10	10	10
Darkside	10	10	10	10	10	10	10	10	10	10	10	10

Table 6
The numbers of the CSV files in the dataset - benign software.

OS	Windows 7 Professional SP1 64bit			
Storage	HDD		SSD	
	120 GB	250 GB	120 GB	250 GB
Condition	(b)	(b)	(b)	(b)
AESCrypt v3.10	10	10	10	10
Zip (Windows 7)	10	10	10	10
SDelete v2.02	10	10	10	10
Excel 2016	10	10	10	11
Firefox v90.02	10	10	10	10

deletes) the 9872 files under the condition (b). The Excel program repeatedly processed the CSV files extracted from the Govdocs1 dataset. The Excel program first opened a CSV file, sort the first column of the CSV file in descending order, and save it as a new file. The Firefox program is accessed in our local webserver every 3 s using an auto-pilot plug-in. Our webserver hosted the 9872 files of condition (b).

Modern cryptosystems convert original plaintext into ciphertext that has high entropy. Ransomware family integrated with such cryptographic ability encrypts victims' files; and, therefore, increases the entropy of written sectors on a storage device. On the other hand, compression programs such as Zip also increase the entropy of the original files because it converts redundant bit patterns into less redundant ones. In addition to encryption and compression programs, secure delete programs also increase the entropy of written sectors on a storage device because it overwrites original data with a secure delete pattern that has a certain amount of entropy (Russinovich and Margosis, 2011).

3.2. Ransomware variants

Cyber-criminals evade protections of anti-virus software by modifying or by improving ransomware code so that they continue to gain huge profits from victims around the world. Signature-based ransomware detection systems cannot reach the speed of the creation of ransomware variants. We have collected access patterns of the 21 ransomware variants shown in Table 7. We will examine whether behavioral features on storage access patterns can identify ransomware families or not, especially for unseen variants. We have obtained two variants of WannaCry, those of Ryuk, and 17 variants of Sodinokibi from ANY.RUN (ANY.RUN). The numbers of the CSV files are presented in Table 8. The samples were executed on 250 GB SSD under condition (b). We will analyze the common behavioral features of storage access patterns among the variants in the later section.

3.3. Different version of operating system

We next consider whether a behavioral model created using access patterns on an operating system *A* can classify access patterns on an operating system *B*. Our new dataset includes access patterns collected on Windows Server 2008 R2 operating system in addition to those on Windows 7. The differences between Windows Server 2008 R2 and Windows 7 are as follows: Windows Server 2008 R2 is the same generation of and the server version of Windows 7 operating systems, and they use identical kernel image ntoskrnl.exe; however, it behaves differently (Russinovich et al., 2012). While the behavior on a server version is optimized for high-performance application servers, that on a client version is optimized for interactive desktop applications. The official document published from Microsoft (Russinovich et al., 2012) indicates that there are differences in the number of worker threads, the size of the heap, and the size of the data cache.

The numbers of the CSV files collected using Windows Server 2008 R2 are presented in Table 9. The samples were executed on 250 GB SSD under condition (b). Both Windows Server 2008 R2 and Windows 7 employ the same file system, New Technology File System (NTFS) version 3.1 including Log File Service (LFS) version 1.1, and GUID partition tables. We, therefore, will be able to examine how the difference in a version of an operating system affects the performance of a ransomware detection system under an identical condition of the file system.

3.4. Full drive encryption

BitLocker, a full drive encryption function, is integrated with enterprise editions of modern Microsoft Windows operating systems to address threats of data theft or exposure from stolen or inappropriately decommissioned computers (Microsoft, 2018). As a result, access patterns on BitLocker-enabled storage devices may differ from original access patterns on a storage device without BitLocker. However, if they share common storage access patterns, we can use a single behavioral model for both storage devices. Table 9 presents the number of the CSV files of storage access patterns collected on a BitLocker-enabled storage device. The data collection was conducted on Windows 7 Enterprise 64 bit operating system using another write-protected USB flash drive to store a

Table 7
Ransomware samples (variants): SHA-256 hash values and the first submission dates at VirusTotal.

Name	SHA-256 hash	Date
WannaCry variant #1	c365ddaa345cfcaff3d629505572a484cff5221933d68e4a52130b8bb7badaf9	2017−05
WannaCry variant #2	09a46b3e1be080745a6d8d88d6b5bd351b1c7586ae0dc94d0c238ee36421cafa	2017−05
Ryuk variant #1	c7d465c8069662e6206684a7cc80e7f820dff3cfb6feb2f35a7356998694b7c1	2019−11
Ryuk variant #2	74654957ba3c9f1ce8bb513954b9deea68a5a82217806977a1247fb342db109f	2019−10
Sodinokibi variant #1	16bb2fd2a291f8c5a0ca60b834106d037bdbc3c3c47fadfd19e208a4a2a5b4d6	2019−04
Sodinokibi variant #2	139a7d6656feebe539b2cb94b0729602f6218f54fb5b7531b58cfe040f180548	2019−05
Sodinokibi variant #3	d624ffff251fab2558e34bcdb8e490afb9590d26ab4818a7390ecfe3b70087e6	2019−06
Sodinokibi variant #4	2de1c46a6cd770b49b7f73b087cfa459ce79c7eef0fa96065e3855a3520d34a7	2019−08
Sodinokibi variant #5	53178b6cf05de5165b5b15c88426215b502dcc4c681e8c049e37e3bb503cbec9	2019−10
Sodinokibi variant #6	d66f94a9feaa7ab3c06c6afb7e2c00806607b17c77c068539e7c5f11a0447b00	2019−12
Sodinokibi variant #7	9b62f917afa1c1a61e3be0978c8692dac797dd67ce0e5fd2305cc7c6b5fef392	2020−02
Sodinokibi variant #8	3738b0bd30c7468bbceefbc33def28723e4cec8a3b72a1c67b42697513e74f92	2020−04
Sodinokibi variant #9	140f831ddd180861481c9531aa6859c56503e77d29d00439c1e71c5b93e01e1a	2020−06
Sodinokibi variant #10	dd6d818bff148772767d53e19f65bb3c644512ba150df5110d7a549624055dfb	2020−08
Sodinokibi variant #11	154cb7a5938f62a49d0fd65a25846d3372d65a06d6d1e344ee59edca16e58272	2020−10
Sodinokibi variant #12	a060d113134d0e905a7c00d0131d907f042b94323987b1ce2d24fb9e87bda148	2020−12
Sodinokibi variant #13	f4f73a451c1ec493eb3b4395d06de73598fcf5b8f7d13e81418238824d90fda3	2021−02
Sodinokibi variant #14	24b76a1ff50bd83741116def674867aa441dc196db8b140d4a992e1ad736d4d4	2021−04
Sodinokibi variant #15	04419b76566142902680b2c44b216905b44a5743502530066e408bac72d20864	2021−06
Sodinokibi variant #16	0f58625addd69f66282924298d843f12f7c2dc2e4d6571952830b880c08cdfee	2021−07
Sodinokibi variant #17	9f256973ee6ddcd3d781761480c00220a140fad833dc9a6a085f45c419d1714e	2021−09

Table 8
The numbers of the CSV files in the dataset - ransomware variants.

OS	Windows 7 Pro SP1 64bit	
Storage	SSD 250 GB	
Condition	(b)	
WannaCry variant	#1, #2	10 each, 20 in total
Ryuk variant	#1, #2	10 each, 20 in total
Sodinokibi variant	#1 - #17	10 each, 170 in total

Table 9
The numbers of the CSV files in the dataset - a different version of an operating system and full drive encryption.

OS	Windows Server	Windows 7 Enterprise
Storage	2008 R2 Standard	64bit
	SSD 250 GB	BitLocker-enabled SSD 250 GB
Condition	(b)	(b)
TeslaCrypt	11	10
Cerber	10	10
WannaCry	10	10
GandCrab v4	10	10
Ryuk	10	10
Sodinokibi	10	10
Darkside	10	10
AESCrypt	10	10
SDelete	10	10
Zip	10	10
Excel	10	10
Firefox	10	10

storage decryption key to BitLocker and to protect the key from being overwritten by ransomware.

The difference between application-level file encryption and OS-level full drive encryption is described as follows. While most file encryption programs encrypt only the contents of files, they do not encrypt the files' metadata, such as file names and timestamps. On the other hand, an operating system's full drive encryption function encrypts an entire partition on a physical storage device except for boot components. During ransomware attacks, ransomware encrypts files in an application layer. Then, BitLocker encrypts them again in an operating system layer. As described in Section 2.1, access patterns were collected using the developed lightweight hypervisor; therefore, it collects access patterns encrypted by ransomware and BitLocker (i.e., double encryption related to the two layers). We will examine the effects of a full drive encryption function on the performance of a ransomware detection system in the later section.

4. Analysis of the dataset

In this section, we present the usage of our new dataset through some examples of machine learning algorithms. First, a feature extractor's design and implementation are presented, followed by a visualization method using a dimensionality reduction algorithm. Finally, a prototype implementation of a ransomware detection system is presented.

4.1. Feature engineering

Conventional machine learning techniques need careful engineering to design feature extractors. Although users of our new dataset will be able to transform CSV files of the dataset into any form, we present an example of preprocessing method to extract feature vectors. The dataset is composed of many time-series data points, and each of them represents individual access. As described in Section 2.2, a four-dimensional vector represents single read access, and a five-dimensional vector represents single write access. Thus, each vector at a point in time does not represent changes in access patterns over time. If we employ all the individual four and five-dimensional vectors separately, such vector space will make most of the vectors too close; therefore, it would no longer perform good classification.

We have developed a prototype of feature extractor (Hirano and Kobayashi, 2019) that transforms a set of data points within an time window T_{window} into a single five-dimensional vector $x(t) = \{T_{write}(t), T_{read}(t), V_{write}(t), V_{read}(t), H_{write}(t)\}$ where:

- $T_{write}(t)$ is an average wtite throughput [byte/s]
- $T_{read}(t)$ is an average read throughput [byte/s]
- $V_{write}(t)$ is a variance of logical block addresses (LBAs) on which the data were written

- $V_{read}(t)$ is a variance of logical block addresses (LBAs) on which the data were read
- $H_{write}(t)$ is an average normalized Shannon entropy of written sectors

where t is a time elapsed after a sample was executed. A set of the five-dimensional features is $\chi = \{x(0), x(1), x(2), \ldots, x(T_d - T_{window})\}$, where T_d is a time t of the last data point. Our feature extractor shifts the time window of T_{window} in every second until t of $x(t)$ reaches at $(T_d - T_{window})$. For example, when T_{window} is 10 s and T_d is 30 s, a feature vector $x(0)$ is calculated using original data points from 0 s to 10 s, a feature vector $x(1)$ is calculated using those from 1 s to 11 s, …, and a feature vector $x(20)$ is calculated using those from 20 s to 30 s. When a set of features χ is calculated using $T_d = 30$ s, it indicates that we will be able to detect ransomware in 30 s after execution of ransomware; T_d is, therefore, referred to as a detection time of ransomware, or duration of data points fed into a machine-learning model.

The following equation calculates a variance of logical block addresses (LBAs).

$$V(t) = \frac{1}{N-1}\sum_{i=1}^{N}(LBA_i - \overline{LBA})^2 \tag{2}$$

where $\overline{LBA}$ is the mean of LBA_i, LBA_i is a logical block address of an ith read access or that of an ith write access, N is the number of the read accesses or that of the write accesses in T_{window}.

The following equation calculates an average normalized Shannon entropy.

$$H_{write}(t) = \frac{1}{N}\sum_{i=1}^{N} H_i(s) \tag{3}$$

where $H_i(s)$ is a Shannon entropy of an ith written sector calculated using Equation (1) and N is the number of write accesses in T_{window}.

4.2. Visualization using dimensionality reduction algorithm

Dimensionality reduction algorithms are able to convert a set of high-dimensional features $\chi = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$ into a set of two or three-dimensional features $\gamma = \{y_0, y_1, y_2, \ldots, y_{n-1}\}$ that can be displayed in a scatter plot. A technique called t-SNE (van der Maaten and Hinton, 2008) is one of the dimensionality reduction algorithms that is carefully designed to preserve the significant structure of high-dimensional original feature vectors of χ as much as possible in a low-dimensional map of γ. t-SNE is a popular variation of Stochastic Neighbor Embedding (SNE) (Hinton and Roweis, 2002) and such dimensionality reduction algorithm is an essential toolkit for conventional machine learning algorithms, especially to transform high-dimensional raw vectors into low-dimensional vectors. We have employed t-SNE to illustrate the five-dimensional features of our new dataset with scatter plots in two-dimensional features to understand their internal representation.

Fig. 4 shows the results of applying t-SNE to a set of five-dimensional features χ, which were extracted from data points of the CSV files both in Table 5 and in Table 6. The set of feature vectors, χ, was calculated using $T_{window} = 10$ s and $T_d = 30$ s and 60 s. In the scatter plots of the 26 classes, we can distinguish the 3 different conditions of (a), (b), and (c). In the scatter plots of the 12 classes, all the 3 conditions were grouped into a single class. In the scatter plots of the 2 classes, there are only two classes: the ransomware class containing the 7 ransomware samples and the benign class containing the 5 benign software samples. Although we extracted feature vectors from all the types and partition sizes of storage devices, the two-dimensional maps clearly show some of the clusters in the dataset. Therefore, it indicates that we can use the developed five-dimensional feature vectors χ to discriminate ransomware and benign software.

4.3. Ransomware detection using machine learning algorithms

The advent of deep learning, computational models that are composed of multiple processing layers to learn representations of data (LeCun et al., 2015), have enabled researchers to create pattern-recognition or machine-learning systems without careful engineering and domain expertise to design a feature extractor. Although deep learning solves some specific problems such as image recognition, speech recognition, and language translation, there have been few deep learning algorithms that solve a wide range of security problems; hence, we currently employ the following 3 conventional machine learning algorithms instead of deep learning: random forest (Ho, 1995), support vector machine (SVM) (Boser et al., 1992), and K-nearest neighbor (kNN) (Altman, 1992), to test a set of feature vectors χ described in Section 4.1. We have implemented machine learning models using scikit-learn 0.23.1. We created random forest models using the RandomForestClassifier class with the following parameters: the number of trees is 10, and the maximum depth of the tree is 10. The SVM models were created using the OneVsRestClassifier class with the radial basis function kernel. The kNN models were created using the KNeighborsClassifier class with the number of neighbors $k = 5$.

Both Fig. 5 and Fig. 6 show the unweighted mean of F_1 scores under 5-fold cross validation. F_1 score is a harmonic mean of precision and recall. An unweighted mean of F_1 score is calculated as follows:

$$F_1 = \frac{1}{N}\sum_{i=1}^{N}\left(2\cdot\frac{Precision_i \cdot Recall_i}{Precision_i + Recall_i}\right) \tag{4}$$

where N is the number of classes. Precision and recall were calculated by class as follows:

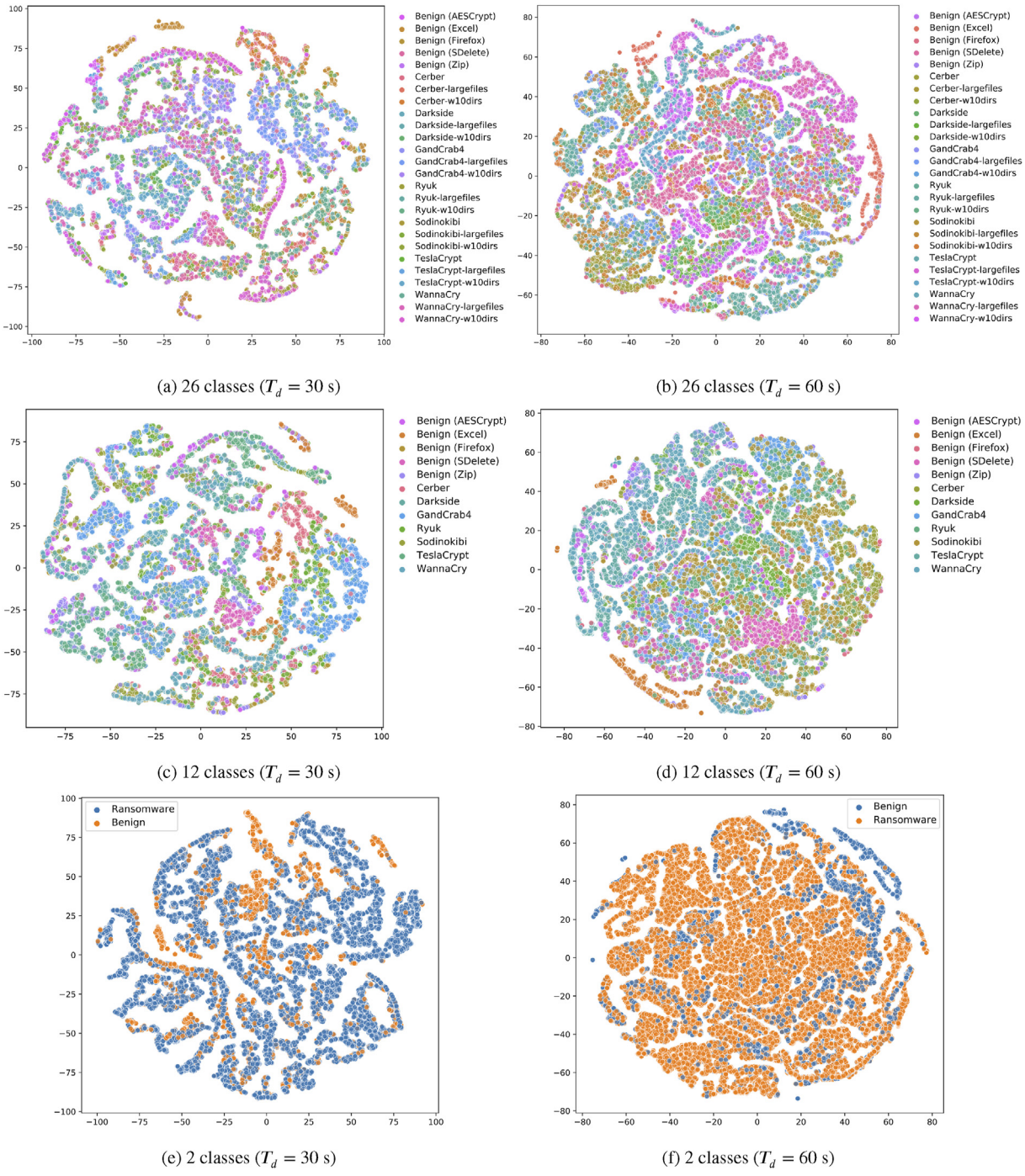
$$Precision_i = \frac{TP_i}{TP_i + FP_i} \tag{5}$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \tag{6}$$

where TP_i (True positive) is the number of correctly classified features belonging to an ith class. FP_i (False positive) is the number of incorrectly classified features belonging to an ith class. FN_i (False negative) is the number of incorrectly classified features as not belonging to an ith class.

Fig. 5 compares the unweighted mean of F_1 scores in $T_d = 30$ s, 60 s, and 90 s. Each set of the feature vectors (χ) was extracted from the CSV files both in Table 5 and in Table 6 using $T_{window} = 10$ s. We used all the types of and all the partition sizes of storage devices (i.e., 120 GB HDD, 250 GB HDD, 120 GB SSD, and 250 GB SSD). In the problem of the 12 classes, the F_1 scores were the highest (0.93) at $T_d = 30$ of random forest and at $T_d = 60$ of K-nearest neighbor. The average F_1 scores of the K-nearest neighbors algorithm was highest. In all the 3 algorithms, the F_1 scores of $T_d = 30$ s were better than or almost equals to those of $T_d = 60$ s and 90 s. The F_1 scores of the 2 classes were better than those of the 12 classes and those of the 26 classes at all T_d.

Fig. 6 compares the unweighted mean of the F_1 scores by the types of and the partition sizes of storage devices. All the machine learning models were trained using $T_d = 30$ s and $T_{window} = 10$ s. The average of the F_1 scores in the 12 classes was the highest when we



(a) 26 classes ($T_d = 30$ s)

(b) 26 classes ($T_d = 60$ s)

(c) 12 classes ($T_d = 30$ s)

(d) 12 classes ($T_d = 60$ s)

(e) 2 classes ($T_d = 30$ s)

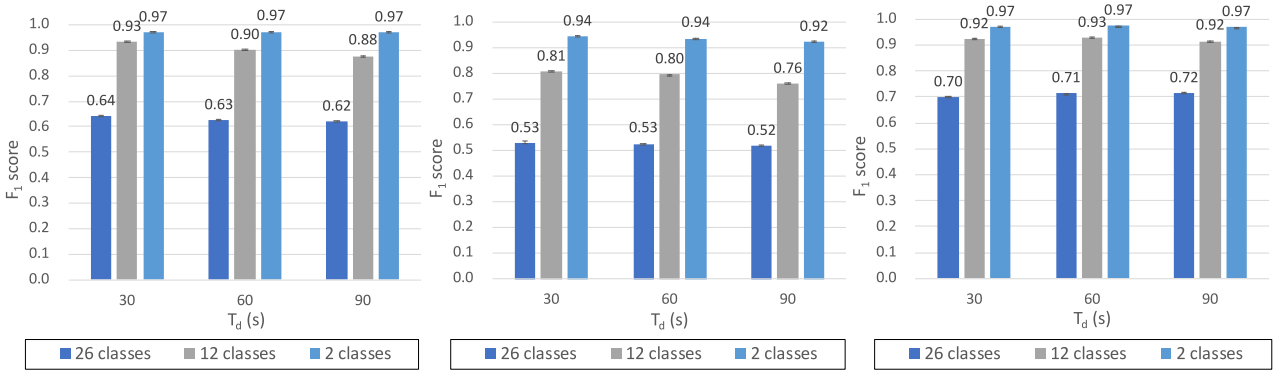
(f) 2 classes ($T_d = 60$ s)

Fig. 4. Visualization by t-SNE.

used random forest (0.959); however, that of the 26 classes was the highest when we used K-nearest neighbor (0.729). In the random forest models, the highest F_1 score of the 26 classes was 0.743 when we used only 250 GB HDD. In contrast, the lowest F_1 score of the 26 classes was 0.642 when we used all the types and partition sizes of storage devices together. We, however, confirmed that most F_1 scores of the 2 classes are within an acceptable range to detect ransomware.

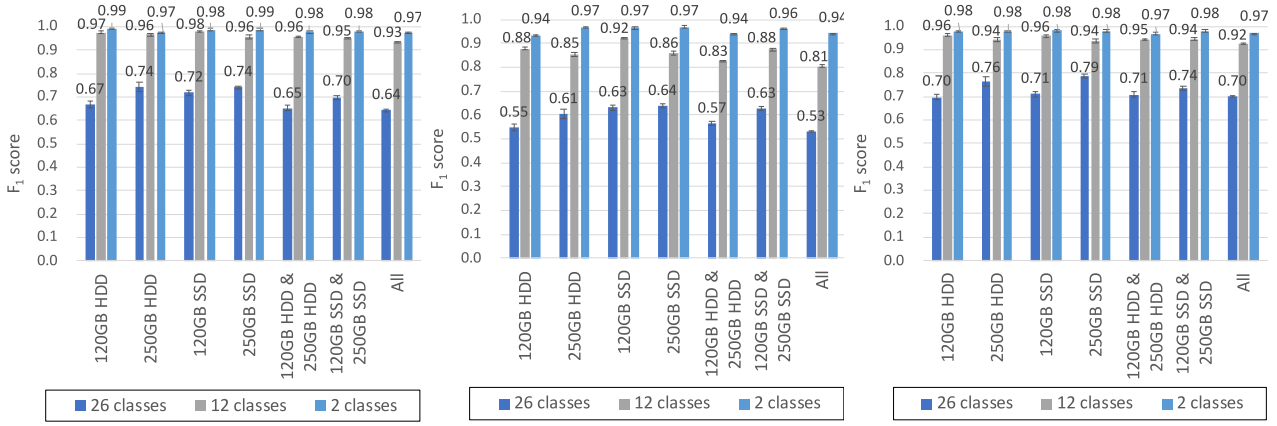
Fig. 7 shows how F_1 scores change with T_{window}. Each of the random forest models was trained using access patterns of all the types and partition sizes of storage device shown both in Table 5 and in Table 6, and $T_d = 90$ s. When we trained the model using $T_{window} = 30$, the F_1 score of the 26 classes was 0.713, that of the 12 classes was 0.953, and that of the 2 classes was 0.996. The F_1 score of the 2 classes and that of the 12 classes is sufficient for a ransomware detection system prototype.

As shown in Fig. 7, the longer the time of T_{window} was, the higher the F_1 scores were. We, however, have to consider a detection time T_d as well as a time window T_{window}. If our detection system needed to detect ransomware in 30 s immediately after execution (i.e., $T_d = 30$ s), we were not able to employ T_{window} greater than 30. When we use $T_{window} = 30$ s, our feature extractor needs at least



(a) Random forest (b) Support vector machine (c) K-nearest neighbor

Fig. 5. Comparison of unweighted mean of F_1 scores by T_d = 30 s, 60 s, and 90 s.



(a) Random forest (b) Support vector machine (c) K-nearest neighbor

Fig. 6. Comparison of unweighted mean of F_1 scores by 7 different groups of storage devices.

access patterns of 30 s to create a single five-dimensional feature. If we employed T_{window} of 10 s and T_d of 30 s, our detection system would be able to begin to predict at least 10 s after the execution of ransomware. The feature extractor creates 20 five-dimensional features from 10 s to 30 s; the model repeatedly predicts each feature's class. When the number of ransomware predictions reaches a threshold, a system would display a warning message, stop the subsequent storage accesses, and restore the original data from a backup. A designer of such a detection system needs to decide a value of the detection time T_d and of the time window T_{window}, and threshold values, according to requirements of the system (e.g., acceptable false alarm rate).

Fig. 8 shows receiver operating characteristic (ROC) curves of T_{window} = 5 s, 30 s, and 60 s. The random forest models of the 12 classes and those of the 26 classes were trained using T_d = 90 s. The feature vectors were extracted from access patterns on all the types and partition sizes of storage devices. ROC curves show how true positive rate varies with false positive rate. We can use a single scalar value called area under the curve, AUC, to compare multiple classifiers. The greater the value of AUC, the better the performance of the classifier is. Fig. 8 shows that the higher the value of T_{window} was, the higher the values of AUC were. However, as described before, we cannot use a too large value of T_{window} because the detection time (T_d) becomes longer. After all, a designer of a detection system needs to decide appropriate values of T_{window} and T_d.

5. Confusion matrices

In the previous section, we have described how F_1 scores change with T_d, T_{window}, machine learning algorithms, the types of and the partition sizes of storage devices, and the number of classes we predict. Each of the F_1 scores shown in the previous section is a single scalar value; that is, we are not able to analyze the detail of the errors being made by our classifier. Confusion matrices present the correct and incorrect classification numbers in a table; therefore, we can analyze each classifier's performance in detail. In this section, we present the confusion matrices of the machine learning models trained using our RANSAP dataset.

5.1. Ransomware and benign software

Table 10 presents the confusion matrix of the 7 ransomware samples under the 3 conditions and the 5 benign software samples on Windows 7. We trained the model using the half of the CSV files in both Tables 5 and 6. We then tested the model using the rest half of the CSV files to create the confusion matrix. Access patterns on all the types and partition sizes of storage devices were used to

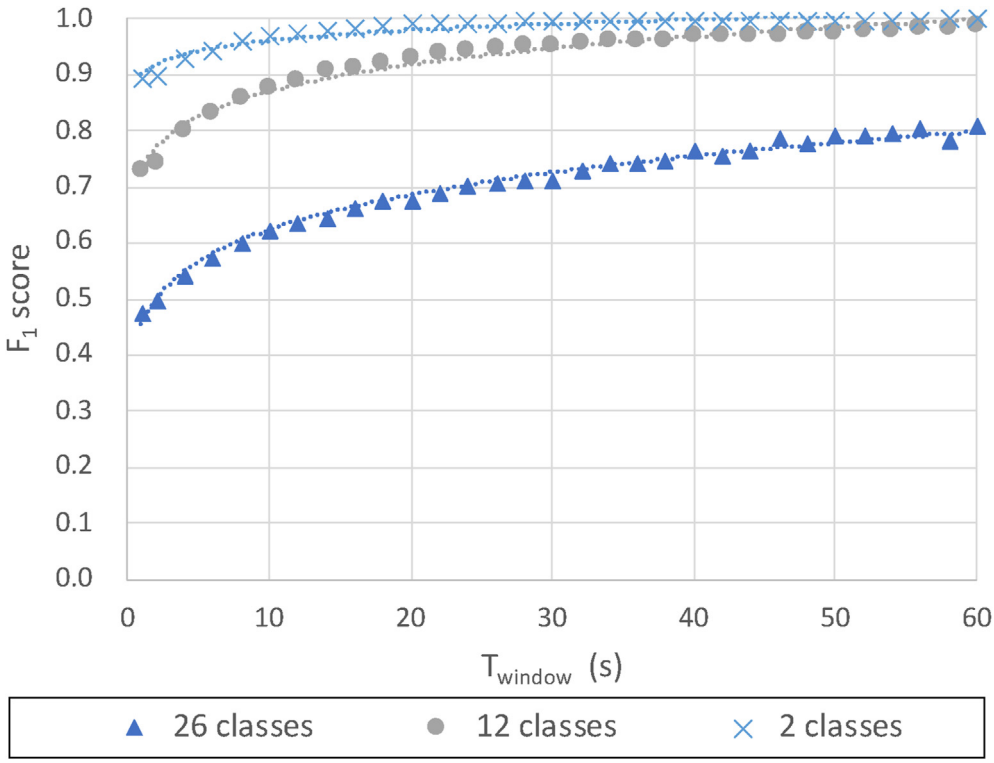


Fig. 7. Changes of unweighted mean of F_1 scores with T_{window}.

train the model. We extracted feature vectors using $T_d = 90$ s and $T_{window} = 10$ s.

Table 10 includes the 12 square outlines in bold black lines at the diagonal. The total number within them indicates that prediction is correct when we do not consider differences in the condition of (a), (b), and (c); and, most of the predictions of the ransomware were within it. TeslaCrypt and WannaCry were precisely classified even when we consider the differences in the condition of (a), (b), and (c). The results indicate that the difference in the condition, the number of and the sizes of files in a desktop did not affect the performance of the developed classifier as we expected.

Next, we analyze the similarity among ransomware families. The orange graphs on the right side show F_1 scores in the 26, 12, and 2 classes problems. In the 26 classes problem, the F_1 scores of TeslaCrypt and those of WannaCry were higher than other ransomware samples; and, conversely, the F_1 scores of Cerber were the lowest. In the 12 classes problem, the F_1 scores of Ryuk were the lowest. Many access patterns of Ryuk were incorrectly classified in GandCrab4, Cerber, and Sodinokibi. We have confirmed the similarity of storage access patterns among these ransomware families.

The purpose of our new dataset is to detect ransomware. False alarm rate or false positive rate, also called type I error, is one of the most critical metrics to deploy a developed classifier in a production environment. The two square outlines in bold red lines in Table 10 represent the number of correct predictions in the 2 classes problem. While most feature vectors of the 7 ransomware samples were correctly classified in the top-left square outline in red, those of the 5 benign software samples were correctly classified in the bottom-right square outline in red. In both classifications, we observed that the number of false positives was low. The average of the F_1 scores in the 2 classes problem was 0.962.

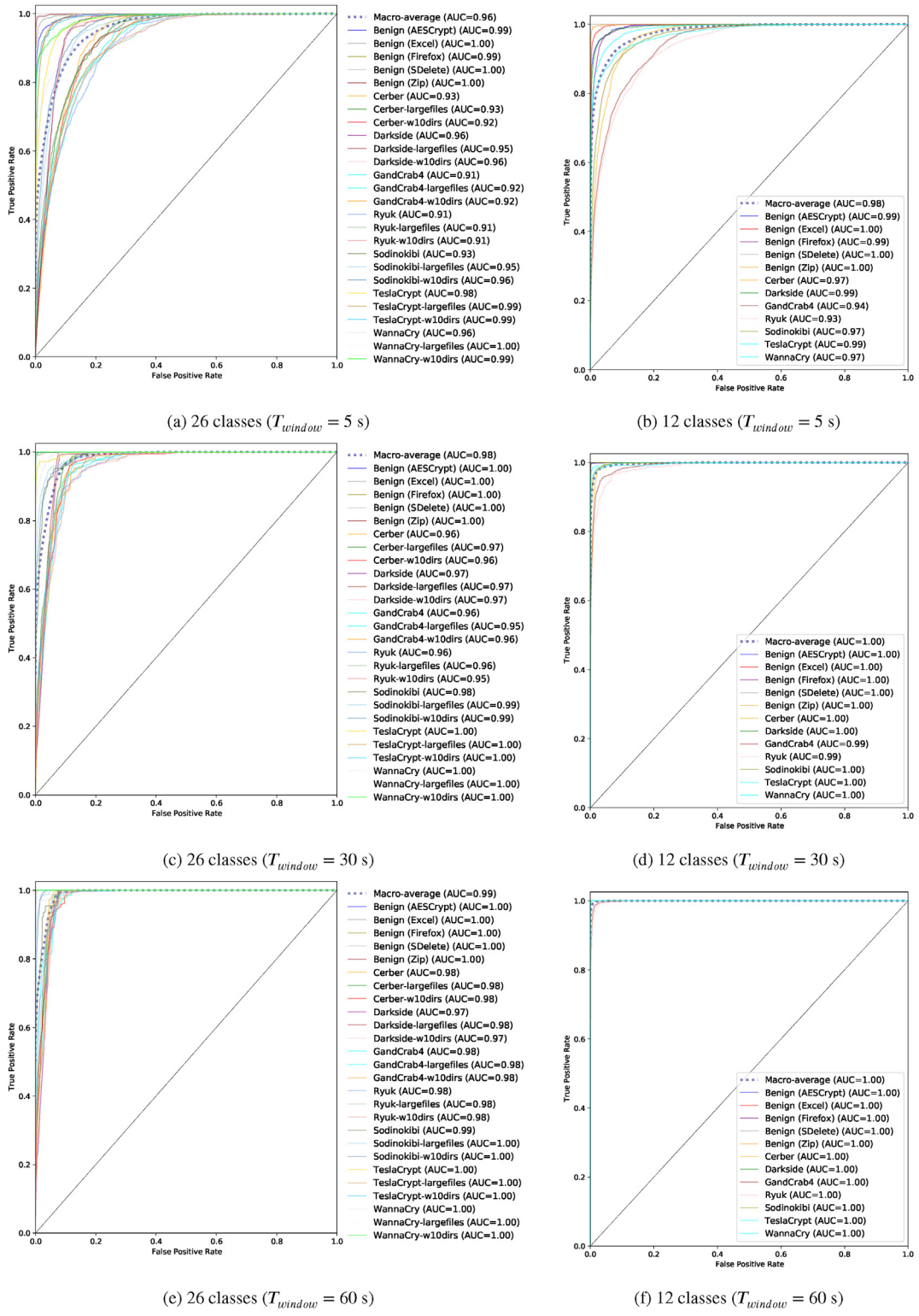
5.2. Ransomware variants

Table 11 presents the confusion matrix of the 21 ransomware variants on Windows 7 operating system. The detail of the ransomware variants was described in Section 3.2. We trained a random forest model using all the CSV files of 250 GB SSD both in Table 5 and in Table 6, and $T_d = 90$ s and $T_{window} = 10$ s. We then classified the 21 ransomware variants in Table 8 using the model.

The yellow graphs in Table 11 show recalls in 12 classes and those in 2 classes. In the problem of the 12 classes, recalls of the two variants of WananCry were the highest, and the mean recall was 0.830. In contrast, those of Ryuk were the lowest, and the mean recall was 0.067. The two variants of Ryuk were mainly classified in GandCrab4, Darkside, and Sodinokibi rather than Ryuk. On the other hand, the mean recall of Sodinokibi in 12 classes was 0.343. Some variants of Sodinokibi were incorrectly classified in WannaCry, TeslaCrypt, Ryuk, and GandCrab4. Most of the predictions, however, fall into the red square outline in Table 11. Thus, the developed classifier correctly identified most of the variants as the ransomware class. The average of the recall scores in the problem of the 2 classes was 0.941. The result indicates that we will be able to predict some unseen ransomware variants using the proposed dataset.

Fig. 9 shows changes in the detection rate (recall) of 17 Sodinokibi variants in 2 years and 5 months. Although the recall of the 12 classes showed some changes in 2 years and 5 months, the recall of the 2 classes did not show significant changes except for them on 2019−04, 2020−04, and 2020−06.

Fig. 10 shows the changes of the 5 feature vectors to the original sample of Sodinokibi. The feature vectors were calculated using $T_{window} = 90$ s and $T_d = 150$ s. While Shannon entropy of written



(a) 26 classes ($T_{window} = 5$ s)

(b) 12 classes ($T_{window} = 5$ s)

(c) 26 classes ($T_{window} = 30$ s)

(d) 12 classes ($T_{window} = 30$ s)

(e) 26 classes ($T_{window} = 60$ s)

(f) 12 classes ($T_{window} = 60$ s)

Fig. 8. Receiver operating characteristic (ROC) curves of random forest models in 26 classes and 12 classes problems.

sectors H_{write} were close to the original feature (i.e., 1.0), values of variance of LBA both on read and write operations, V_{read} and V_{write}, were relatively lower than the original feature and have a few fluctuations. On the other hand, write throughput T_{write} and read throughput T_{read} have some significant fluctuations (e.g., 2019−10, 2020−04, 2020−06, 2021−04, and 2021−06) that caused decrease in recall of 12 classes. We also confirmed that variant #2 has unique features compared to other variants.

Although we did not observe significant concept drift (Widmer and Kubat, 1996) in the two classes problem, the changes of features that decrease the detection rate should be carefully monitored when we deploy the detector in long term. In the case of

Table 10
Confusion matrix of 7 ransomware and 5 benign software on Windows 7.

Actual class	Cond.	TeslaCrypt (a)	TeslaCrypt (b)	TeslaCrypt (c)	Cerber (a)	Cerber (b)	Cerber (c)	WannaCry (a)	WannaCry (b)	WannaCry (c)	GandCrab4 (a)	GandCrab4 (b)	GandCrab4 (c)	Ryuk (a)	Ryuk (b)	Ryuk (c)	Sodinokibi (a)	Sodinokibi (b)	Sodinokibi (c)	Darkside (a)	Darkside (b)	Darkside (c)	AES Crypt	SDelete	Zip	Excel	Firefox	F1-score in 26 classes	F1-score in 12 classes	F1-score in 2 classes
TeslaCrypt	(a)	1216	15	5	27	4	14	1	2	0	2	2	3	33	1	9	6	15	2	3	1	3	0	0	1	31	125	0.760		
	(b)	48	1392	40	24	0	5	0	10	0	0	0	2	10	1	9	0	3	0	0	0	7	43	0	0	39	0	0.890	0.887	
	(c)	38	26	1398	13	2	3	0	0	18	12	12	1	8	0	5	0	3	0	1	0	2	21	0	0	38	0	0.900		
Cerber	(a)	29	0	0	675	209	408	5	0	0	65	7	4	18	22	19	11	6	4	37	1	5	0	0	0	14	10	0.320		
	(b)	29	0	0	440	456	374	3	2	2	44	8	8	37	27	13	12	20	10	33	1	8	2	0	2	30	8	0.330	0.747	
	(c)	19	0	2	531	203	633	2	0	1	64	7	8	11	3	13	6	10	10	10	1	5	0	0	0	14	4	0.360		
WannaCry	(a)	4	0	0	139	7	59	1132	1	0	108	11	6	6	1	17	36	0	3	24	2	16	0	2	0	9	0	0.800		
	(b)	2	15	0	0	0	1	16	1387	15	40	0	5	1	0	19	1	7	4	0	1	0	26	0	0	0	0	0.930	0.911	
	(c)	1	0	19	0	3	1	0	5	1509	0	7	1	0	2	2	0	8	7	0	0	0	9	0	0	0	0	0.960		
GandCrab4	(a)	12	1	0	108	40	63	5	2	0	863	127	71	91	16	66	43	6	4	26	8	4	1	0	0	0	0	0.370		
	(b)	14	2	0	122	40	70	7	2	0	464	533	105	29	39	89	21	6	3	21	5	25	0	0	0	0	0	0.400	0.683	0.985
	(c)	5	0	21	120	34	73	2	0	5	606	129	388	16	24	93	22	25	6	16	1	4	0	0	0	1	0	0.340		
Ryuk	(a)	34	2	0	135	49	41	13	0	0	246	35	19	410	91	358	62	30	17	11	7	3	0	0	0	40	9	0.330		
	(b)	5	4	1	138	36	55	5	1	1	125	78	8	73	575	374	36	34	15	14	12	13	0	0	0	47	5	0.440	0.655	
	(c)	10	1	1	158	34	63	11	0	3	104	51	31	68	129	772	39	54	34	9	11	10	1	0	4	44	6	0.430		
Sodinokibi	(a)	26	0	0	41	28	38	16	0	0	260	36	9	44	1	26	635	149	252	0	0	0	0	0	0	13	30	0.440		
	(b)	42	1	1	5	16	6	1	2	0	29	5	5	4	3	44	140	1026	246	4	0	1	0	0	1	15	22	0.630	0.812	
	(c)	23	1	0	13	20	9	9	1	0	87	7	9	10	3	30	224	200	953	0	0	2	2	0	0	13	18	0.590		
Darkside	(a)	10	6	2	4	0	0	0	0	0	0	5	1	1	2	7	0	17	5	944	141	412	0	0	0	19	0	0.430		
	(b)	7	11	0	2	2	1	3	0	0	2	4	1	3	0	2	1	10	5	822	350	372	0	0	0	27	0	0.300	0.940	
	(c)	10	1	1	0	2	0	0	0	0	0	0	0	0	0	1	5	16	5	875	179	536	0	0	0	14	8	0.350		
AESCrypt		2	5	2	0	1	0	0	21	6	2	0	0	1	0	0	1	0	1	0	0	0	1508	0	41	0	0	0.940	0.936	
SDelete		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1613	0	0	0	1.000	0.999	
Zip		0	0	0	0	0	0	0	0	0	2	0	0	3	0	0	0	0	1	0	0	0	19	0	1584	0	0	0.980	0.977	0.939
Excel		9	1	1	1	0	2	0	0	0	0	0	0	2	3	2	1	5	7	0	0	0	0	0	0	1567	2	0.880	0.875	
Firefox		94	0	0	4	10	23	0	0	0	0	0	0	1	0	0	4	8	6	0	0	0	0	0	0	3	1336	0.870	0.870	

Predicted class: Ransomware | Benign software

Table 11
Confusion matrix of 21 ransomware variants.

Actual class	TeslaCrypt	Cerber	WannaCry	GandCrab4	Ryuk	Sodinokibi	Darkside	AESCrypt	SDelete	Zip	Excel	Firefox	Recall in 12 classes	Recall in 2 classes
WannaCry #1	101	0	669	1	1	21	0	7	0	0	0	0	0.836	0.991
WannaCry #2	112	0	659	0	4	5	0	20	0	0	0	0	0.824	0.975
Ryuk #1	0	4	7	554	77	111	47	0	0	0	0	0	0.096	1.000
Ryuk #2	0	0	0	341	30	0	429	0	0	0	0	0	0.038	1.000
Sodinokibi #1	105	79	0	0	59	75	0	0	0	0	0	474	0.095	0.402
Sodinokibi #2	260	31	83	23	19	362	0	3	0	0	17	2	0.453	0.973
Sodinokibi #3	153	6	61	82	7	472	0	0	0	0	7	12	0.590	0.976
Sodinokibi #4	56	11	305	57	35	299	0	28	0	0	0	9	0.374	0.954
Sodinokibi #5	131	0	386	71	35	161	0	12	4	0	0	0	0.201	0.980
Sodinokibi #6	246	0	59	25	17	436	0	0	0	0	0	0	0.557	1.000
Sodinokibi #7	187	5	77	45	8	456	0	0	0	0	14	8	0.570	0.973
Sodinokibi #8	99	31	338	20	84	114	3	38	0	0	4	69	0.143	0.861
Sodinokibi #9	184	3	312	52	30	77	0	66	0	76	0	0	0.096	0.823
Sodinokibi #10	243	59	59	34	21	380	0	1	0	0	0	3	0.475	0.995
Sodinokibi #11	247	3	66	24	14	430	0	0	0	0	0	0	0.548	1.000
Sodinokibi #12	238	89	68	19	18	364	0	0	0	0	0	4	0.455	0.995
Sodinokibi #13	194	1	290	38	103	151	19	0	4	0	0	0	0.189	0.995
Sodinokibi #14	100	1	443	48	125	66	10	0	6	0	0	1	0.083	0.991
Sodinokibi #15	138	0	393	48	86	63	5	47	0	20	0	0	0.079	0.916
Sodinokibi #16	139	12	110	44	7	465	0	0	7	0	7	9	0.581	0.971
Sodinokibi #17	165	26	227	17	77	272	13	0	3	0	0	0	0.340	0.996

Predicted class: Ransomware | Benign software

Sodinokibi, a custom packer is used to evade signature-based detection (McAfee); the experimental results in Fig. 10, however, indicate that many variants have no major changes in the true malware part. When we observe performance degradation of the classifier, it must be updated to adapt to the latest data distribution. Adaptive learning, a method updating predictive models online during their operation to react concept drifts (Gama et al., 2014), will be needed when the data distribution is frequently changed.

5.3. Different version of operating system

Table 12 presents the confusion matrix of the 7 ransomware samples and the 5 benign software samples on the Windows Server 2008 R2 operating system. The experiment was conducted as follows: First, we trained a random forest model using access patterns of Windows 7, all the CSV files of 250 GB SSD in Tables 5 and 6. The feature vectors were extracted using $T_d = 90$ s and $T_{window} = 10$ s. Finally, we predict access patterns of ransomware and benign samples on Windows Server 2008 R2 in Table 9 using the model.

We examine whether or not the classifier trained using access patterns on a version of an operating system can predict a correct class using those on a different version of the operating system. Table 12 includes the 12 square outlines at the diagonal, and the numbers in them indicate that the number of the correct predictions. The orange graphs on the right side show the F_1 scores in the 12 and 2 classes. In the problem of the 12 classes, the feature vectors of Ryuk were mainly classified in WannaCry and TeslaCrypt. The feature vectors of GandCrab4 were mainly classified in Darkside, Firefox, and Cerber. On the other hand, some of the feature vectors of the 5 benign software samples were incorrectly classified. For example, the 331 feature vectors of AESCrypt were incorrectly classified in Zip, and, conversely, the 114 feature vectors of

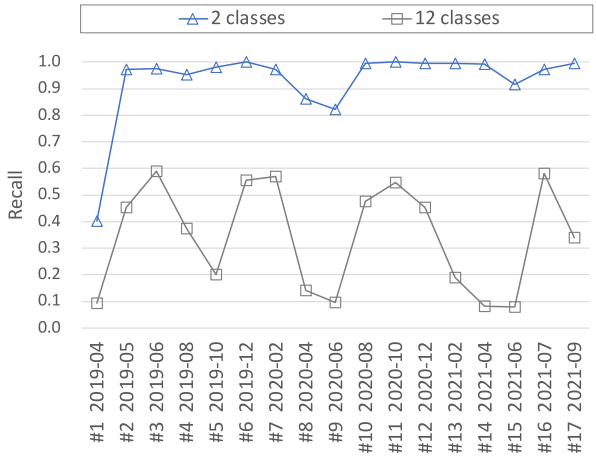


Fig. 9. Changes of detection rates (recall) of Sodinokibi variants in 2 years and 5 months.

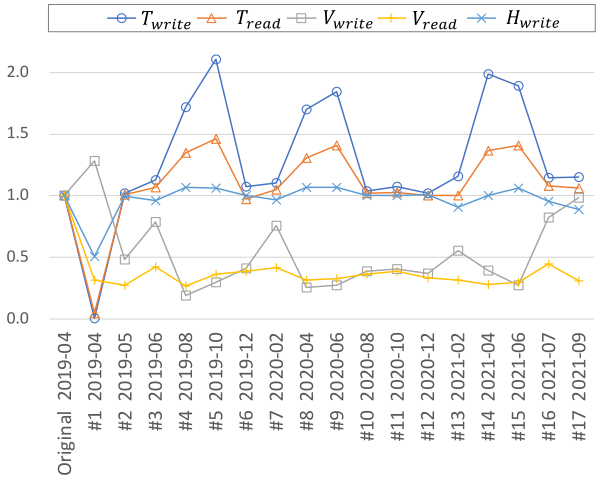


Fig. 10. Changes of the mean of 5 feature vectors of Sodinokibi variants in 2 years and 5 months.

Zip were incorrectly classified in AESCrypt. The result indicates that AESCrypt and Zip share behavioral features when executed on Windows Server 2008 R2.

Incorrect classifications of ransomware raise concerns about false negative rates. For example, some feature vectors of ransomware, except for Cerber and GandCrab4, were incorrectly classified in AESCrypt. These wrong predictions increase our detection system's false negative rates (i.e., type II error). However, the orange graphs in the 2 classes reveal that the F_1 scores are better than we expected. The average of the F_1 scores in the 2 classes problem was 0.818. Hence, it will be possible to deploy the developed classifier on a different version of an operating system after, for example, some improvements of the developed feature extractor.

5.4. Full drive encryption

Table 13 presents the confusion matrix of the 7 ransomware samples and the 5 benign software samples on BitLocker-enabled storage devices. The experiment was conducted as follows: First, we trained a random forest model using access patterns on storage devices without BitLocker, all the CSV files of 250 GB SSD in Table 5 and in Table 6. The feature vectors were extracted using $T_d = 90$ s and $T_{window} = 10$ s. Finally, we predict access patterns of ransomware and benign samples on BitLocker-enabled storage devices in Table 9 using the model.

We examine whether or not the classifier trained using access patterns on storage devices without BitLocker can predict a correct class using those on BitLocker-enabled storage devices. Table 13 includes the 12 square outlines at the diagonal, and the numbers in them indicate that the number of the correct predictions. The orange graphs on the right side show the F_1 score in the 12 and 2 classes. When a classifier randomly chooses a class, every F_1 score of the 12 classes becomes approximately 0.08. Those of the 2 classes become approximately 0.58 and 0.42 in a ransomware class and a benign software class. In our experiments, the average F_1 score in the 12 classes was 0.055, and those in 2 classes were 0.310. Thus, considering these criteria, the classifier failed to predict most of them.

In particular, a significant number of incorrect classifications in Zip were observed. For example, the total number of predictions in all ransomware samples was 5,599, and the number of classifications in Zip was 2941. Thus, 52.5% in all ransomware samples were incorrectly identified as Zip. On the other hand, the total number of predictions in all benign samples was 3981. The number of classifications in Ryuk was 1785. Thus, 44.8% in all benign samples were incorrectly identified as Ryuk. Furthermore, the classifications of the 5 samples (i.e., WannaCry, GandCrab4, Darkside, AESCrypt, and SDelete) entirely failed.

The result we presented here indicates that the Shannon entropy of our feature vectors has a vital role in detecting ransomware. To verify the hypothesis, we calculated the feature

Table 12
Confusion matrix of 7 ransomware and 5 benign software on Windows Server 2008 R2.

Actual class \ Predicted class	TeslaCrypt	Cerber	WannaCry	GandCrab4	Ryuk	Sodinokibi	Darkside	AESCrypt	SDelete	Zip	Excel	Firefox	F1-score in 12 classes	F1-score in 2 classes
TeslaCrypt	350	0	16	0	0	1	0	513	0	0	0	0	0.293	
Cerber	201	541	0	0	0	26	31	0	0	0	1	0	0.614	
WannaCry	71	1	416	2	13	6	14	245	0	3	0	29	0.418	
GandCrab4	61	124	0	33	0	36	396	0	0	0	5	145	0.075	0.839
Ryuk	193	9	408	28	3	17	46	44	0	3	0	12	0.008	
Sodinokibi	409	27	125	9	0	63	3	70	0	0	0	94	0.113	
Darkside	36	181	4	10	0	114	452	2	0	0	1	0	0.519	
AESCrypt	4	5	6	0	0	3	0	450	0	331	0	0	0.402	
SDelete	0	0	217	0	0	0	1	0	582	0	0	0	0.842	
Zip	4	0	0	0	0	3	0	114	0	677	0	0	0.747	0.798
Excel	97	68	0	0	12	42	0	0	0	0	531	30	0.806	
Firefox	80	6	0	0	0	0	0	0	0	0	0	665	0.771	

Ransomware | Benign software

Table 13
Confusion matrix of the 7 ransomware and the 5 benign software on BitLocker-enabled storage devices.

Actual class \ Predicted class	TeslaCrypt	Cerber	WannaCry	GandCrab4	Ryuk	Sodinokibi	Darkside	AESCrypt	SDelete	Zip	Excel	Firefox	F1-score in 12 classes	F1-score in 2 classes
TeslaCrypt	41	37	0	0	6	2	0	40	0	674	0	0	0.058	
Cerber	2	136	0	0	373	26	0	0	0	262	0	0	0.173	
WannaCry	4	2	0	0	0	0	0	510	0	284	0	0	0.000	
GandCrab4	13	394	0	0	82	19	0	48	0	222	0	22	0.000	0.366
Ryuk	99	19	55	0	7	6	0	54	0	558	0	2	0.005	
Sodinokibi	160	86	57	1	17	79	0	40	0	360	0	0	0.103	
Darkside	0	31	0	12	17	111	0	48	0	581	0	0	0.000	
AESCrypt	66	36	0	0	560	110	0	0	0	24	0	0	0.000	
SDelete	7	0	223	0	0	1	0	428	0	141	0	0	0.000	
Zip	211	27	0	2	205	255	0	6	0	92	0	0	0.043	0.254
Excel	0	0	0	0	697	3	0	0	0	3	94	0	0.211	
Firefox	2	6	0	0	323	128	0	33	0	272	0	26	0.062	
									Ransomware		Benign software			

importance in the random forest model as shown in Fig. 11. The higher the value of feature importance is, the more influential the feature is. As we can see in the graph, Shannon entropy $H_{write}(t)$ is the highest in them; however, other vectors such as throughput of write accesses and that of read accesses are also vital. When we employ BitLocker-enabled storage devices, all of the vectors of Shannon entropy become almost 1; therefore, we cannot use the most important feature on them. Nevertheless, why were the F_1 scores low even though the model uses all of the other features? We speculate that access patterns on BitLocker-enabled storage devices were utterly different from those on storage devices without BitLocker.

In summary, we will not be able to employ the machine learning model trained using access patterns on storage devices without BitLocker in those on BitLocker-enabled storage devices as it is. There are two possible solutions to address the problem: (1) to design more robust feature vectors on a full drive encryption function, and (2) to create a separate model using access patterns on BitLocker-enabled storage devices. Although solution (2) is straightforward, it increases the number of models. Solution (1) is more complicated than solution (2). However, we believe that there is room for improvements in the design of five-dimensional feature vectors and the time-window-based feature extractor. The conventional machine learning algorithms we have employed need careful engineering and domain expertise to design a feature extractor. Instead of manual feature extraction, we will be able to exploit deep learning techniques such as one-dimensional convolutional neural networks (CNN) to extract hidden features from the dataset in some automated way (Kiranyaz et al., 2021).

6. Coexistence of two detection methods

This section shows how the thin-hypervisor method can complement the weakness of OS-centric methods. In the experiments, the proposed thin-hypervisor method and an OS-centric method were executed simultaneously. We employed Windows File System Minifilter Drivers (Microsoft, a) as the OS-centric method. We used Minispy, a minifilter driver included in Windows Driver Kit (WDK) (Microsoft, b); the implementation is equivalent to the architecture of the I/O access monitor in UNVEIL (Kharaz et al., 2016) that captures I/O Request Packets (IRPs) on storage devices in OS (i.e., kernel) layer.

We monitored access patterns of the 7 ransomware samples shown in Table 4 on 250 GB SSD under condition (b) in 3 min. Fig. 12 shows storage access patterns on a computer infected by Sodinokibi in 150 s obtained from the minifilter driver (above) and those from BitVisor (below). Although the two different methods were used, we were able to obtain access patterns with the same characteristics. Since the minifilter driver creates access logs consisting of I/O operation types (e.g., IRP_MJ_READ, IRP_MJ_WRITE, and IRP_MJ_CREATE), file names, timestamps, and process IDs, we were able to identify when Sodinokibi started enumerating decoy files, encrypting the files, creating ransom notes. We were also able to identify when the Windows indexing service (i.e., SearchIndexer.exe) started indexing the encrypted and newly created files. In contrast, the thin hypervisor method creates access logs consisting of only I/O operation types (i.e., read and write), accessed LBAs, and entropy of written 4 KiB blocks. Thus, the OS-centric methods can obtain more accurate information than the thin-hypervisor

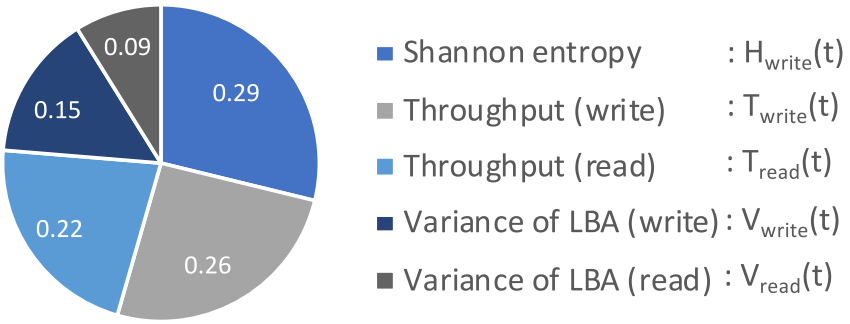


Fig. 11. Feature importance in the random forest model.

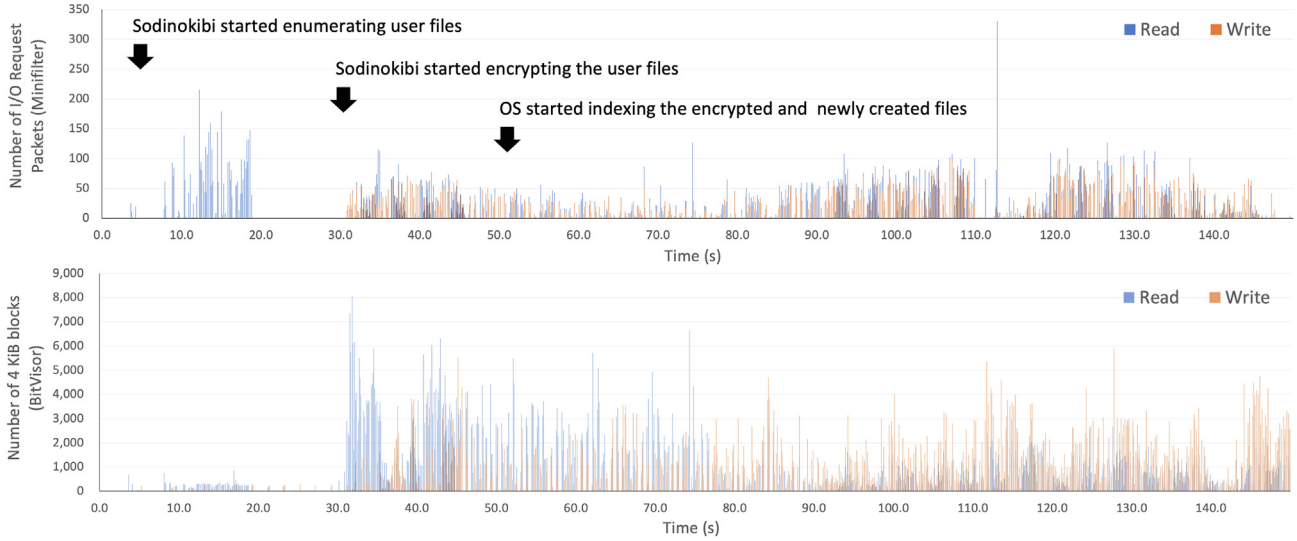


Fig. 12. The number of I/O Request Packets (IRPs) on a computer infected by Sodinokibi obtained from Minispy (above) and those of 4 KiB blocks obtained from BitVisor (below).

method. If that is the case, why should we use the detection method using hypervisor?

In the experiments, Darkside forcibly terminated the Minispy program; we, therefore, could not monitor access patterns at all. Ryuk deleted the access log files. Ryuk and Sodinokibi disappeared in the output of the tasklist.exe program. Although the OS-centric method was attacked and disabled by these ransomware samples, the thin-hypervisor method was able to monitor access patterns continuously. While the advantage of the OS-centric methods is to collect more accurate information, the disadvantage is that they are vulnerable to attacks. While the advantage of the thin-hypervisor method is stealthy features against ransomware, the disadvantage is to collect less accurate information. We, therefore, concluded that the proposed thin hypervisor method does not replace the OS-centric methods but effectively complements the weakness of the OS-centric methods especially when the OS-centric method was attacked. Such multiple protection design will build a stronger foundation for the entire ransomware detection platform.

7. Evaluation in more realistic usage

The dataset consists of access patterns when a single sample was executed in the controlled environment. The computers in real environments, however, execute multiple applications and services simultaneously. This section shows experimental results to examine how storage accesses of ransomware samples are dominant than other programs such as Office applications and Windows background services.

Fig. 13 shows the number of I/O Request Packets (IRPs) per process obtained using Minispy (Microsoft, b), a Windows File System Minifilter driver. The 7 ransomware samples presented in Table 4 were executed and the logs of IRPs were collected in 3 min. We were not able to collect the logs of Darkside and Ryuk because Darkside disabled the monitoring program and Ryuk deleted the log file. During the execution of WannaCry, the number of processes was between 35 and 38. During the execution of Sodinokibi, the process ID of Sodinokibi disappeared in the output of tasklist.exe; the number of processes we could confirm was between 36 and 37. The top 8 processes that issued the most IRPs are shown in (a) and (b). We confirmed that executions of ransomware samples increase the file accesses of the System process and the SearchIndexer.exe process; these access patternsare also an important part of the features for detecting ransomware. The other ransomware samples including TeslaCrypt, Cerber, and GandCrab4 have the same trend as WannaCry (a) and Sodinokibi (b).

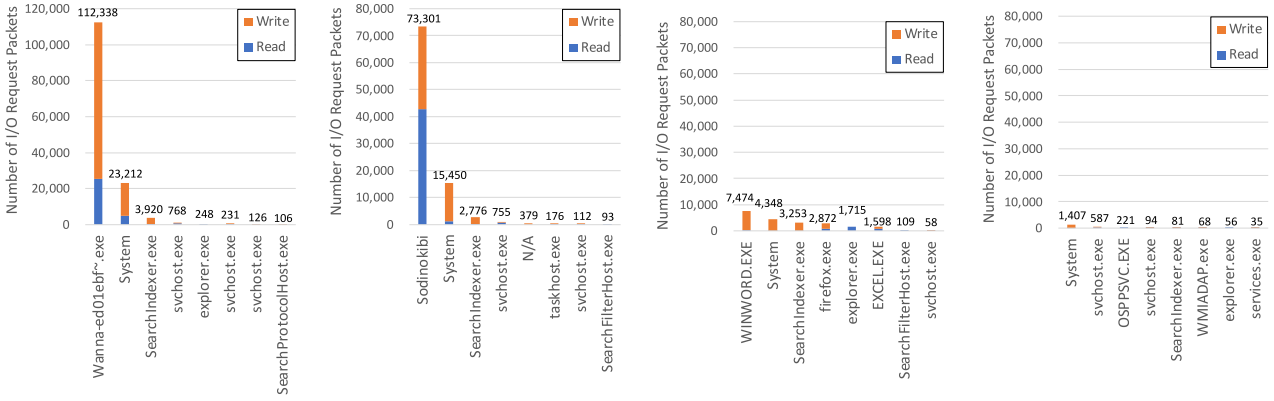
We also collected logs of IRPs in more realistic usage, a computer in office environments (c) in 15 min. To reproduce the actual user environment, we created Visual Basic Application (VBA) macros of Word and Excel that copy the way office workers behave such as typing texts, creating tables and graphs, and saving the file. A test presentation file of PowerPoint was played repeatedly. Moreover, YouTube video was played in Firefox web browser simultaneously. All the files used in the experiment are included in the dataset. Office Professional 2016 was used in the experiment. The number of processes during the experiment was between 44 and 46; the top 8 processes that issued the most IRPs are shown in (c). Although WINWORD.EXE (Word) and other processes issued thousands of IRPs, they are enough small compared to those of ransomware samples with tens of thousands of IRPs.

The number of IRPs when a computer is in an idle state (d) in 15 min was measured to confirm storage access patterns of background processes of Windows 7. In the idle state, the number of processes was between 30 and 33; the top 8 processes that issued the most IRPs are shown in (d). The numbers of IRPs in the idle state were small enough to ignore compared to those of ransomware samples.

The total numbers of IRPs of ransomware sample, System, and SearchIndexer.exe were 139,470 (WannaCry), 91,527 (Sodinokibi), 66,008 (TeslaCrypt), 168,920 (Cerber), and 145,817 (GandCrab4). On the other hand, the total number of IRPs of the top three processes in office environments (c) in 15 min was 15,075. We confirmed that execution of a ransomware sample increases storage accesses 4 times to 11 times higher than office environments. Considering the experimental periods (i.e., 3 min and 15 min), we can estimate that execution of a ransomware sample increases storage accesses 20 times to 55 times higher than office environments.

We tested our classifier on these access patterns. The classifier was trained using access patterns of 250 GB SSD with $T_d = 90$ s and $T_{window} = 10$ s. Accuracy of access patterns of a computer in office environments (c) in 2 classes was 0.818 and that of a computer in idle state (d) was 0.987; they were correctly identified in the benign class. This result indicates that the classifier can distinguish ransomware from many applications that have low I/O operations.

We also tested the classifier against access patterns when Sodinbokibi was executed simultaneously with Word, Excel, PowerPoint, and Firefox web browser playing YouTube (i.e., a computer



(a) WannaCry in 3 min (b) Sodinokibi in 3 min (c) Word, Excel, PowerPoint, and Firefox browser playing YouTube video in 15 min (d) Idle state in 15 min

Fig. 13. The number of I/O Request Packets per process.

in office environments (c)); the accuracy of the mixed access patterns decreases to 0.503. Through the series of experiments, we confirmed that the accuracy of detecting ransomware using only storage access patterns is sufficient under conditions that a single sample is executed, it is, however, limited under more complex conditions that multiple applications are executed simultaneously. To improve the accuracy of the classifier, we are examining an application of deep learning instead of conventional machine learning algorithms. In addition, we are developing a novel function that collects memory access patterns in the hypervisor layer in addition to storage access patterns; the additional features will be examined in our next papers.

8. Limitations of the proposed dataset

The current dataset mainly consists of access patterns on the older Windows operating systems (i.e., Windows 7 and Windows Server 2008 R2). The latest Windows operating systems, such as Windows 10 and Windows Server 2018, however, have different internal storage access patterns and internal security mechanisms that have changed within the last years. Despite the improvement in the security mechanisms of the latest OSes, many attackers continue to target an older version of OSes. Ransomware attacks on nation-state-scale infrastructure such as energy systems are on the rise (Nicol, 2021). Legacy cyber-physical systems use old human−machine interface (HMI) applications that communicate with legacy programmable logic controllers (PLCs) on an older version of Windows (e.g., Windows 7) without the latest patches. The protection of such legacy systems is still an important research area even if we can obtain the latest, more secure systems.

Attackers always seek unpatched and obsolete critical system-level components such as operating and network management systems. For example, Owaida reported that 40.5% of ransomware detections in 2020 were still related to WannaCry (Amer, 2020) even though Microsoft immediately released patches to stop the worldwide WannaCry outbreak in 2017. Williams has described this phenomenon using a metaphor; the people who live in glass houses are happy the stones weren't thrown at them. Unfortunately, academics and industries continue to fail to solve such security problems that superficially seem trivial and obvious (e.g., unpatched and obsolete operating systems) (Williams, 2021).

Although we are developing a new surveillance system that can monitor the latest Windows, it is still under development. We, of course, will add storage access patterns of the latest Windows operating systems in our future dataset. The authors, however, believe that the dataset of older Windows operating systems will still play a vital role because of the existence of many legacy information systems.

9. Discussion

9.1. Availability of an open dataset

Deep learning and machine learning have become essential components of today's security systems. However, the lack of a standard and realistic open dataset has made the development of such components slower and harder. Although there have been many state-of-the-art open datasets for static analysis, there have been few open datasets for dynamic analysis. For example, Harang and Rudd have released the state-of-the-art dataset named SOREL-20M (Sophos and ReversingLabs 20 Million) for static analysis consisting of nearly 20 million Windows Portable Executable files of malware (Harang and Rudd, 2020). In addition, Anderson and Roth have released the Ember dataset for static analysis consisting of 1 million files (Anderson and Roth, 2018).

On the other hand, Berrueta et al. have released one of the few open datasets for dynamic analysis of ransomware (Berrueta et al., 2020). The dataset contains dynamic features extracted from packet capture data of ransomware. To our best knowledge, our new dataset is one of the few open datasets consisting of dynamic features focusing on ransomware storage access patterns.

Table 14 presents related work on ransomware detection using dynamic features such as entropy and file operations; and the availability of an open dataset of dynamic features. Kharaz et al. have first presented a dynamic ransomware detection system named UNVEIL that focuses on input and output (I/O) data buffer entropy and access patterns obtained from file system activity monitor (Kharaz et al., 2016). Around the same time, Scaife et al. have presented CryptoDrop, an early-warning system of ransomware that employs dynamic features including file type changes, similarity measurement, Shannon entropy obtained using a kernel driver (Scaife et al., 2016). Continella et al. have developed ShieldFS (Continella et al., 2016), a file system that is immune to ransomware attacks. In addition, they have publicly released a dataset of input and output request packets that represent ransomware file system activities. McIntosh et al. have employed the dataset to compare

Table 14
Related work of a ransomware detection system and availability of dataset using behavioral features.

Reference	Year	Entropy	File operations	Availability of dataset
Kharaz et al., (2016)	2016	Yes	Yes	
Scaife et al., (2016)	2016	Yes	Yes	
Sgandurra et al., (2016)	2016		Yes	
Continella et al., (2016)	2016	Yes	Yes	Yes (via email)
Kharraz and Kirda, (2017)	2017	Yes	Yes	
Shaukat and Ribeiro, (2018)	2018	Yes	Yes	
Hampton et al., (2018)	2018		Yes	
Morato et al., (2018)	2018		Yes	Yes (Berrueta et al., 2020)
Hull et al., (2019)	2019		Yes	
Lee et al., (2019)	2019	Yes		
Tang et al., (2020)	2020	Yes	Yes	
McIntosh et al., (2021)	2021		Yes	

their proposal (McIntosh et al., 2021) with ShieldFS. Finally, Morato et al. have developed a ransomware detection algorithm (Morato et al., 2018), and they have released a dataset consisting of dynamic features in public (Berrueta et al., 2020). The dataset contains input and output operations on a file server; they were created using packet capture data of Windows Server Message Block (SMB) protocols. Even though most of the related work in Table 14 provide sufficient information of the dataset, such as a collection of ransomware samples and that of decoy files, few of them provide a dataset consisting of dynamic features.

While most of the related work in Table 14 have employed features obtained from the operating system layer, few of them have employed features obtained from the hypervisor layer. For example, an advantage of using a hypervisor is to protect a detection function from being attacked by ransomware. Tang et al. have employed a hypervisor to collect system call numbers issued by ransomware while protecting its function from attacks (Tang et al., 2020). In contrast, we collected low-level storage access patterns using the developed hypervisor without OS-dependent information such as system call numbers. The developed hypervisor, therefore, can be employed on other operating systems by design. A limitation of our system is that the developed hypervisor works only on computers using Intel processors because it uses Intel Virtualization Technology.

9.2. Comparison with other dynamic analysis methods

We now compare our hypervisor-based approach with other sandbox systems. Malware sandboxes have become the de facto standard to extract malwares' dynamic behaviors. Cuckoo, for example, is a popular open-source automated malware analysis system that can retrieve traces of malware such as API calls, file operations, memory dumps, network traffic traces (Oktavianto and Muhardianto, 2013). On the other hand, our hypervisor-based approach cannot retrieve OS-level information without adding to a function that understands the internal of an operating system. In the Cuckoo architecture, a special agent software installed on an operating system collects the traces. A typical deployment of Cuckoo sandbox also employs a hypervisor to restart a clean environment using its snapshot mechanism. In contrast, our hypervisor-based method obtains ransomware behaviors without using any agent programs on an operating system layer to collect more accurate behavioral features than conventional sandboxes requiring agent programs.

Malware such as ransomware, however, often behaves differently when it detects sandbox environments. Yokoyama et al. presented a tool to exfiltrate characteristics of such malware sandboxes to avoid attackers' evasion techniques (Yokoyama et al., 2016). They first fingerprinted Internet-connected sandboxes using a hostname, installation date of OS, disk space, RAM size, processor type, desktop icons, and network configuration. They then create a classifier using supervised machine learning techniques to show that adversaries can automatically generate a classifier that can reliably tell a sandbox and a real system apart. Yokoyama et al. also reported that the Cuckoo sandbox could be easily detected by checking a particular file named agent.py, which must be running upon the analysis of a sample.

An additional advantage of a hypervisor-based surveillance system (i.e., virtual machine introspection system) over conventional sandboxes is stealthy features. Dynamic malware analysis relies heavily on the use of hypervisors for functionality and safety. Modern malware, however, checks for the presence of a hypervisor and changes its behavior when it was detected. Brengel et al., for example, have presented a timing-based virtualization detection method using VM Exit overhead and Translation-Lookaside Buffer (TLB) eviction overhead (Brengel et al., 2016). Their method successfully detected 95.95% of the virtualized environment on 31 public malware sandboxes.

Shi et al. have proposed a detect-and-hide approach, which systematically addresses anti-hypervisor techniques in malware (Shi et al., 2017). Attackers can detect the presence of hypervisor by using specially crafted fuzzing test-cases, which were initially proposed by Martignoni et al. (2009), to verify whether the CPU is properly virtualized or not. Shi et al. have presented VM Cloak that detects the fuzzing test-cases and modifies the command's outcomes to match those generated by a physical machine. Of course, we might not be able to remove every trace of the hypervisor on our surveillance mechanism; however, the stealth performance of our system can be improved by using the technique.

Another advantage of a hypervisor-based surveillance system over sandboxes that needs agent programs on operating systems is its ability to construct an isolated environment inaccessible from the outside of the hypervisor. For example, Mishra et al. have presented VMShield that performs virtual memory introspection from the hypervisor (i.e., trusted domain) to collect the run-time behavior of processes, making it impossible for malware to evade the security tool (Mishra et al., 2021). Kiperberg et al., for example, have presented a specially crafted hypervisor that is responsible for decryption, execution, and protection of the crucial codes to prevent reverse engineering attacks (Kiperberg et al., 2019). They have employed Intel's virtual machine extensions as well as we utilized in our hypervisor.

10. Use cases of the proposed dataset and detection method

The first use case is multiple-layered protection. The most important artifacts related to ransomware behavior are on an operating system, network, HDD, and other subsystems. Many malware

detection systems such as anti-virus software, intrusion detection systems, and firewall software use these artifacts. The recent articles, however, address some evasion techniques of these conventional malware detection systems. To address the evasion techniques, we believe that a multiple-layered protection mechanism will be needed. Even if attackers could evade the first protection layer of an operating system (e.g., anti-virus, sandbox), the following protection layer of a hypervisor (i.e., the proposed hypervisor-based system) will detect the attack. The experimental results shown in this paper have indicated that the proposed method can detect ransomware attacks using only low-level behavioral features. If we need to consider that attackers can compromise the first protection layer, the additional security layer would be helpful.

The second use case is a system to recover from ransomware attacks. If the system could collect a complete set of read and write operations of ransomware, we can restore original files before it encrypts or destroyed user files. The experimental results show that ransomware detection can be done in the first 30 s immediately after execution. Accordingly, such a restoration system will need to preserve read and written sectors, at least, in the last 30 s. If ransomware were detected, the system would stop the guest operating system immediately, and the encrypted or deleted sectors would be recovered using the original data sectors the system recorded.

A similar system, FlashGuard, a ransomware-tolerant SSD, which has a firmware-level recovery system without relying on explicit backups, has been presented by Huang et al. (2017). FlashGuard leverages the observation that the existing SSD already performs out-of-place writes to mitigate the long erase latency of flash memories. Although their proposal can be used only on SSD drives, our system can be used on any type of serial ATA drive. On the other hand, Hirano et al. have presented LogDrive, a proactive data collection and forensic analysis framework, in particular, for virtual storage devices in Infrastructure as a Service (IaaS) cloud environments (Hirano et al., 2018).

We first employed WebHDFS (Restful API of Hadoop Distributed File System) protocol over TCP to send complete storage access patterns to our cluster (Hirano et al., 2017). Since the throughput of the previous TCP-based system was not sufficient on faster SSD drives, we replaced it with a UDP-based system despite potential packet losses so that we can collect access patterns on faster SSD drives. Considering potential packet losses in UDP, we implemented a mechanism to add some reliability as follows. The developed hypervisor writes a sequence number on each UDP packet. The sequence number is checked on the UDP server to avoid the significant number of UDP packet losses that spoil our dataset's quality. In addition to checking packet losses during UDP transfer, the header information of each read and write operation is checked, for example, whether each timestamp has a valid date and time or not. The broken UDP packets are then discarded, and only the access patterns having no errors were saved on the dataset. The current system presented in this paper uses storage access patterns only for machine learning; we, therefore, can tolerate a few packet losses during UDP transfer.

11. State-of-the-art

11.1. Static analysis

Raff et al. have presented MalConv, a deep-learning-based malware detection method using raw byte sequences (Raff et al., 2018). Gibert et al. have processed a malware sample as an entropy stream; features are extracted using wavelet transforms, and then used to train deep learning models (Gibert et al., 2018). Sun and Qian have identified malware families using a feature images generation method that combines recurrent neural networks (RNNs) and convolutional neural networks (CNNs) (Sun and Qian, 2021). Kim et al. have presented a multimodal deep learning method that converts multiple static features of malware samples such as embedded strings, API calls, and shared libraries into a single feature vector (Kim et al., 2019). Onwuzurike et al. have presented MaMaDroid, a malware detection method, that is more resilient to API changes using the sequence of abstracted API calls in Markov chains to train a model (Onwuzurike et al., 2019).

Cozzi et al., on the other hand, have reported that many Linux malware samples on IoT devices, binaries of a variety of CPU architectures (e.g., MIPS, PowerPC, ARM, and dominant $\times$86), can not be parsed using major static analysis tools (Cozzi et al., 2018). The differences in CPU architecture make static analysis difficult and, therefore, increases the need for malware detection using dynamic features. If we could implement our system on other CPU architecture in addition to $\times$86, the access patterns of standard serial ATA devices can be used for machine learning since the system does not use any CPU-architecture-dependent information.

11.2. Dynamic analysis

Smith et al. have surveyed existing datasets used for classifying malware by machine learning algorithms and demonstrated the importance of datasets that reflects behaviors of real-world systems. They, in particular, pointed out that extracted features from the datasets are primarily syntactic, not behavioral; moreover, datasets generally contain extremes exemplars that can be easily discriminated (Smith et al., 2020).

Many researchers have, therefore, presented a detection method using behavioral features of malware. Zhang and Wang have used a combination of API call arguments and API names to detect malware (Zhang et al., 2020). They employed one-dimensional CNN and bidirectional long-short term memory (LSTM) networks to learn sequential correlation among API calls. Wüchner et al. have presented a behavior-based malware detection approach that uses compression based mining on quantitative data flow graphs to increase the detection rate of malware (Wüchner et al., 2019). Chen et al. have proposed a real-time detection system, called RansomProber, that uses users' finger movements as a dynamic feature (Chen et al., 2018). Kwon et al. presented a method for cybercrime scene reconstruction using library calls obtained from an operating system layer (Kwon et al., 2021). Saracino et al. have presented MADAM, a host-based malware detection system for Android devices which simultaneously analyzes and correlates features at four levels (i.e., kernel, application, user, and package) to detect malicious behaviors (Saracino et al., 2018). Features of storage access patterns presented in this paper might be able to use in a classifier that uses multiple features such as MADAM.

11.3. Malware sandboxes

Sartea and Farinelli have pointed out a key problem of dynamic analysis methods; sandboxes can miss important malicious behaviors visible only if triggered by specific actions on the infected system (Sartea and Farinelli, 2017). You et al. have developed a state-of-the-art lightweight forced execution technique (You et al., 2020). Forced execution is an effective method to penetrate malware self-protection mechanisms such as sandbox detectors and expose hidden behavior, by forcefully setting certain branch outcomes. Although we believed that the proposed RANSAP dataset provides access patterns representative of typical configurations, it may not cover all of the possible behaviors of ransomware payloads that depend on execution environments. We might be able to collect more storage access patterns in various configurations using the forced execution technique.

Küchler et al. conducted an extensive set of experiments to evaluate the impact of the analysis time on the results collected by a malware analysis sandbox, and they have confirmed that a threshold of 2 min is sufficient in the vast majority of the cases for the analysis of collected malware samples (Küchler et al., 2021).

11.4. Evasion attacks and countermeasures

Runtime packing is a conventional technique designed to store a compressed, encrypted, or encoded copy of the original program, to evade static malware analysis. Aghakhani et al. have demonstrated that the signals extracted from packed executables are not rich enough for ML-based malware detection to (1) generalize their knowledge to operate on unseen packers, and (2) be robust against adversarial examples (Aghakhani et al., 2020). Mantovani et al. have investigated the hypothesis that packing and entropy are strongly correlated and their results show that, despite all samples have a low entropy value, over 30% of them adopt some form of runtime packing (Mantovani et al., 2020).

Adversarial machine learning attacks have become a major threat to recent security systems. Chen et al. have developed an automated tool to generate the adversarial examples of Android malware without human intervention to evade ML-based detection (Chen et al., 2020). Pierazzi et al. have presented an automated generation system of adversarial malware which takes less than 2 min to mutate a given malware example into a variant that can evade a hardened state-of-the-art classifier (Pierazzi et al., 2020).

Ye et al. have presented a structured heterogeneous graph for modeling relationships between Android malware and the runtime API call sequences to make evasion harder (Ye et al., 2019). Hou et al. have presented HinDroid, an Android malware detector that has robustness on evasion attacks using a structured heterogeneous information network of API calls (Hou et al., 2018). Ding et al. have presented an assembly code representation learning model Asm2Vec that is more robust against changes introduced by obfuscation and optimizations on static malware analysis (Ding et al., 2019).

D'Elia et al. have presented BluePill that offers new intervention and customization capabilities for dissection on evasive malware samples. They have employed the Dynamic Binary Instrumentation (DBI) approach instead of the Virtual Machine Introspection (VMI) approach (D'Elia et al., 2020) because of the difficulty in building a transparent hypervisor.

11.5. Hypervisor-based security systems

Virtualization is designed to be transparent, that is, unprivileged users should not be able to detect whether a system is virtualized. Such detection, therefore, can result in serious security threats on virtualization-based dynamic analysis systems. Zhang et al. have shown a new stealthy detection method of the hardware-assisted virtualization using TLB, Last-Level Cache (LLC), and Level-1 Data (L1D) Cache (Zhang et al., 2021). Shie et al. have proposed a detect-and-hide approach, which systematically addresses anti-VM techniques of malware (Shi et al., 2017). Schumilo et al. have presented HYPER-CUBE, a novel fuzzer that aims explicitly at testing hypervisors to find vulnerabilities (Schumilo et al., 2020).

Trusted Execution Environments (TEE), on the other hand, can be used to create an isolated environment as well as a hypervisor. Paccagnella et al. have presented CUSTOS that provides a tamper-evident logging layer; They presented a prototype of Linux's audit system that uses Intel Software Guard Extensions (SGX) (Paccagnella et al., 2020).

11.6. Federated learning

Another interesting research area on ML-based malware detection is federated learning. It is a type of distributed machine learning to collectively train a deep learning model while preserving privacy on training data (Jere et al., 2021). Since our system uses users' storage access patterns that include their privacy (i.e., written data on sectors), federated learning is a possible solution to resolve the problem.

Xu et al. have proposed VerifyNet, the privacy-preserving, and verifiable federated learning framework by using a double-masking protocol to guarantee the confidentiality of users' local gradients during the federated learning (Xu et al., 2020). Wei et al. have presented a method to effectively prevent information leakage using the concept of differential privacy, in which artificial noises are added to the parameters at the clients' side before aggregating (Wei et al., 2020). Sav et al. have addressed the problem of privacy-preserving training and evaluation of neural networks in an N-party, federated learning setting (Sav et al., 2021). Cao et al. presented FLTrust, Byzantine robust federated learning, to enable a service provider to learn an accurate global model when a bounded number of clients are malicious (Cao et al., 2022).

12. Future work

Another problem of the proposed dataset is the lack of access patterns accessed from remote computers. For example, Windows-based file servers and Linux-based network-attached storage (NAS) systems are widely used in office environments, and they are the most crucial target for cyber-criminals. We also need to consider cloud storage services such as OneDrive and Google Drive. Our new dataset currently includes access patterns only on a local machine; hence, we need to collect more access patterns on the network-based storage services as well as those on the latest versions of the Windows operating system.

In addition, the proposed detection mechanisms have not been integrated with the hypervisor. While we have implemented the monitoring function of storage access patterns in the hypervisor, the machine learning model is created and used only on another machine. Therefore, we need to implement the same machine learning function in the hypervisor to detect ransomware on the machine. Moreover, federated learning allows collaboration with distributed machines to create an ML model while preserving each user's privacy.

Although deep learning models trained using static features of malware binaries for identifying malware families have been proposed, those using dynamic or behavioral features have rarely been proposed. Our results show that we can identify ransomware families using storage access patterns. Such large-scale characterization of modern ransomware using distributed measurement terminals or cloud-based sandboxes will be a challenging topic. On the other hand, recent adversarial ML attacks on malware detection have mainly focused on static analysis. As the ML-based ransomware detection using dynamic or behavioral features increases, the attackers will use adversarial ML attacks. Therefore, countermeasures against the new attacks will be needed.

13. Conclusion

In this paper, we have presented RANSAP, our new open dataset of ransomware storage access patterns for solving the problem of few realistic open datasets for dynamic analysis. Furthermore, we have employed a hypervisor to collect storage access patterns of ransomware because it is operating system independent, and the developed thin hypervisor can monitor those on any operating system with the minimum observer effects. Our new open dataset includes storage

access patterns of ransomware and benign software, those of ransomware variants, those on a different version of operating systems, and those on storage devices with a full drive encryption function enabled. We have evaluated the dataset using a proto-type feature extractor and analyzed the results of a machine-learning-based ransomware detection system under various conditions. Our experiments confirmed the average F_1 score of 96.2% in detecting ransomware, 94.1% in detecting variants, 81.8% on a different version of an operating system, and that of 31.0% on storage devices with a full drive encryption function enabled. Even though our new dataset has a few limitations, including performance degradation on mixed access patterns of multiple applications and a lack in access patterns of network-based storage services and those of the latest operating systems, to our best knowledge, the RANSAP dataset is one of the few realistic open datasets consisting of dynamic features for training ML-based ransomware detection systems.

CRediT authorship contribution statement

Manabu Hirano: Writing – original draft, Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Project administration, Software, Supervision, Visualization. **Ryo Hodota:** Software, Data curation. **Ryotaro Kobayashi:** Conceptualization, Methodology, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP20K11825 and JP17K00198. The authors would like to thank Ryota Koyanagi, Hana Takahashi, Hayato Toyama, Hijiri Ito, Koki Ikeda, Noriaki Kawane, and Minami Asaoka for the support in data collection; and Takamichi Omori for the support in developing the hypervisor-based monitoring system. The authors gratefully acknowledge the extremely helpful and constructive comments by the editor and several anonymous reviewers.

References

Aghakhani, H., Gritti, F., Mecca, F., Lindorfer, M., Ortolani, S., Balzarotti, D., Vigna, G., Kruegel, C., 2020. When malware is packin'heat; limits of machine learning classifiers based on static analysis features. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2020.24310.

Al-rimy, B.A.S., Maarof, M.A., Shaid, S.Z.M., 2018. Ransomware threat success factors, taxonomy, and countermeasures: a survey and research directions. Comput. Secur. 74, 144–166. https://doi.org/10.1016/j.cose.2018.01.001.

Altman, N.S., 1992. An introduction to kernel and nearest-neighbor nonparametric regression. Am. Statistician 46, 175–185. https://doi.org/10.2307/2685209.

Amer, Owaida, 2020. WannaCryptor Remains a Global Threat Three Years on. https://www.welivesecurity.com/2020/05/12/wannacryptor-remains-global-threat-three-years-on/. (Accessed 9 August 2021).

Anderson, H.S., Roth, P., 2018. Ember: an Open Dataset for Training Static pe Malware Machine Learning Models arXiv. https://arxiv.org/abs/1804.04637v2.

ANY.RUN, . Interactive malware hunting service. https://any.run. Accessed 9 August 2021.

Berrueta, E., Morato, D., Magaña, E., Izal, M., 2020. Open repository for the evaluation of ransomware detection tools. IEEE Access 8, 65658–65669. https://doi.org/10.1109/ACCESS.2020.2984187.

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory. (COLT '92), pp. 144–152. https://doi.org/10.1145/130385.130401.

Brengel, M., Backes, M., Rossow, C., 2016. Detecting hardware-assisted virtualization. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016). Springer, pp. 207–227. https://doi.org/10.1007/978-3-319-40667-1_11.

Cao, X., Fang, M., Liu, J., Gong, N.Z., 2022. FLTrust: Byzantine-robust federated learning via trust bootstrapping. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2021.24434.

Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., Ahn, G.J., 2018. Uncovering the face of android ransomware: characterization and real-time detection. IEEE Trans. Inf. Forensics Secur. 13, 1286–1300. https://doi.org/10.1109/TIFS.2017.2787905.

Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K., 2020. Android HIV: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inf. Forensics Secur. 15, 987–1001. https://doi.org/10.1109/TIFS.2019.2932228.

Chronicle, . VirusTotal. https://virustotal.com. Accessed 9 August 2021..

Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F., 2016. ShieldFS: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. ACSAC 2016), pp. 336–347. https://doi.org/10.1145/2991079.2991110.

Cozzi, E., Graziano, M., Fratantonio, Y., Balzarotti, D., 2018. Understanding linux malware. In: IEEE Symposium on Security and Privacy. SP), pp. 161–175. https://doi.org/10.1109/SP.2018.00054.

Ding, S.H.H., Fung, B.C.M., Charland, P., 2019. Asm2Vec: boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: IEEE Symposium on Security and Privacy. SP), pp. 472–489. https://doi.org/10.1109/SP.2019.00003.

D'Elia, D.C., Coppa, E., Palmaro, F., Cavallaro, L., 2020. On the dissection of evasive malware. IEEE Trans. Inf. Forensics Secur. 15, 2750–2765. https://doi.org/10.1109/TIFS.2020.2976559.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. 46, 1–37. https://doi.org/10.1145/2523813.

Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G., 2009. Bringing science to digital forensics with standardized forensic corpora. Digit. Invest. 6, S2–S11. https://doi.org/10.1016/j.diin.2009.06.016.

Garfinkel, S.L., 2010. Digital corpora, GovDocs1. https://digitalcorpora.org/corpora/files. (Accessed 9 August 2021).

Gibert, D., Mateu, C., Planes, J., Vicens, R., 2018. Classification of malware by using structural entropy on convolutional neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence 32. https://ojs.aaai.org/index.php/AAAI/article/view/11409.

Hampton, N., Baig, Z., Zeadally, S., 2018. Ransomware behavioural analysis on windows platforms. J. Inform. Secur. Appl. 40, 44–51. https://doi.org/10.1016/j.jisa.2018.02.008.

Harang, R., Rudd, E.M., 2020. SOREL-20M: a large scale benchmark dataset for malicious PE detection. arXiv:2012.07634. https://arxiv.org/abs/2012.07634v1.

Hinton, G., Roweis, S., 2002. Stochastic neighbor embedding. In: Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS 2002). MIT Press, Cambridge, MA, USA, pp. 857–864. https://doi.org/10.5555/2968618.2968725.

Hirano, M., 2021. Open Repository of the RanSAP Dataset. https://github.com/manabu-hirano/RanSAP/. (Accessed 9 August 2021).

Hirano, M., Kobayashi, R., 2019. Machine learning based ransomware detection using storage access patterns obtained from live-forensic hypervisor. In: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). IEEE, pp. 1–6. https://doi.org/10.1109/IOTSMS48152.2019.8939214.

Hirano, M., Tsuzuki, N., Ikeda, S., Kobayashi, R., 2018. LogDrive: a proactive data collection and analysis framework for time-traveling forensic investigation in iaas cloud environments. J. Cloud Comput. 7, 1–25. https://doi.org/10.1186/s13677-018-0119-2.

Hirano, M., Tsuzuki, T., Ikeda, S., Taka, N., Fujiwara, K., Kobayashi, R., 2017. WaybackVisor: hypervisor-based scalable live forensic architecture for timeline analysis. In: Wang, G., Atiquzzaman, M., Yan, Z., Choo, K.K.R. (Eds.), Security, Privacy, and Anonymity in Computation, Communication, and Storage (SpaCCS 2019). Springer International Publishing, Cham, pp. 219–230. https://doi.org/10.1007/978-3-319-72395-2_21.

Ho, T.K., 1995. Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition. IEEE, pp. 278–282.

Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M., 2018. Make evasion harder: an intelligent android malware detection system. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18. International Joint Conferences on Artificial Intelligence Organization, pp. 5279–5283. https://doi.org/10.24963/ijcai.2018/737.

Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K., 2017. FlashGuard: leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017). Association for Computing Machinery, New York, NY, USA, pp. 2231–2244. https://doi.org/10.1145/3133956.3134035.

Hull, G., John, H., Arief, B., 2019. Ransomware deployment methods and analysis: views from a predictive model and human responses. Crime Sci. 8, 1–22. https://doi.org/10.1186/s40163-019-0097-9.

Intel Corporation, 2016. Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume. 3C: system Programming guide, Part 3.

Jere, M.S., Farnan, T., Koushanfar, F., 2021. A taxonomy of attacks on federated learning. IEEE Secur. Priv. 19, 20–28. https://doi.org/10.1109/MSEC.2020.3039941.

Kharaz, A., Arshad, S., Mulliner, C., Robertson, W., Kirda, E., 2016. UNVEIL: a large-scale, automated approach to detecting ransomware. In: 25th USENIX

Security Symposium (USENIX Security 2016), pp. 757–772. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz.
Kharraz, A., Kirda, E., 2017. Redemption: real-time protection against ransomware at end-hosts. In: International Symposium on Research in Attacks, Intrusions, and Defenses (RAID 2017). Springer. Springer International Publishing, Cham, pp. 98–119. https://doi.org/10.1007/978-3-319-66332-6_5.
Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G., 2019. A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inf. Forensics Secur. 14, 773–788. https://doi.org/10.1109/TIFS.2018.2866319.
Kiperberg, M., Leon, R., Resh, A., Algawi, A., Zaidenberg, N.J., 2019. Hypervisor-based protection of code. IEEE Trans. Inf. Forensics Secur. 14, 2203–2216. https://doi.org/10.1109/TIFS.2019.2894577.
Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., Inman, D.J., 2021. 1d convolutional neural networks and applications: a survey. Mech. Syst. Signal Process. 151, 107398. https://doi.org/10.1016/j.ymssp.2020.107398.
Kwon, Y., Wang, W., Jung, J., Lee, K.H., Perdisci, R., 2021. C^{2}SR: cybercrime scene reconstruction for post-mortem forensic analysis. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2021.23128.
Küchler, A., Mantovani, A., Han, Y., Bilge, L., Balzarotti, D., 2021. Does every second count? time-based evolution of malware behavior in sandboxes. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2021.24475.
LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444. https://doi.org/10.1038/nature14539.
Lee, K., Lee, S.Y., Yim, K., 2019. Machine learning based file entropy analysis for ransomware detection in backup systems. IEEE Access 7, 110205–110215. https://doi.org/10.1109/ACCESS.2019.2931136.
van der Maaten, L., Hinton, G., 2008. Visualizing data using t-SNE. J. Mach. Learn. Res. 9, 2579–2605. http://www.jmlr.org/papers/v9/vandermaaten08a.html.
Mantovani, A., Aonzo, S., Ugarte-Pedrero, X., Merlo, A., Balzarotti, D., 2020. Prevalence and impact of low-entropy packing schemes in the malware ecosystem. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2020.24297.
Martignoni, L., Paleari, R., Roglia, G.F., Bruschi, D., 2009. Testing Cpu Emulators. Association for Computing Machinery, New York, NY, USA, pp. 261–272. https://doi.org/10.1145/1572272.1572303.
McAfee, . McAfee ATR analyzes Sodinokibi aka REvil ransomware-as-a-service – what the code tells Us. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/. Accessed 24 October 2021..
McIntosh, T., Watters, P., Kayes, A., Ng, A., Chen, Y.P.P., 2021. Enforcing situation-aware access control to build malware-resilient file systems. Future Generat. Comput. Syst. 115, 568–582. https://doi.org/10.1016/j.future.2020.09.035.
Microsoft, a. Filter manager concepts. https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts. Accessed 19 October 2021..
Microsoft, b. Minispy file system minifilter driver. https://docs.microsoft.com/en-us/samples/microsoft/windows-driver-samples/minispy-file-system-minifilter-driver/. Accessed 19 October 2021.
Microsoft, 2018. BitLocker. https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-overview. (Accessed 9 August 2021).
Mishra, P., Aggarwal, P., Vidyarthi, A., Singh, P., Khan, B., Haes Alhelou, H., Siano, P., 2021. VMShield: memory introspection-based malware detection to secure cloud-based services against stealthy attacks. IEEE Trans. Ind. Inform. https://doi.org/10.1109/TII.2020.3048791, 1–1.
Morato, D., Berrueta, E., Magaña, E., Izal, M., 2018. Ransomware early detection by the analysis of file sharing traffic. J. Netw. Comput. Appl. 124, 14–32. https://doi.org/10.1016/j.jnca.2018.09.013.
Nicol, D.M., 2021. The ransomware threat to energy-delivery systems. IEEE Secur. Priv. 19, 24–32. https://doi.org/10.1109/MSEC.2021.3063678.
Oktavianto, D., Muhardianto, I., 2013. Cuckoo Malware Analysis. Packt Publishing Ltd.
Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G., 2019. MaMaDroid: detecting android malware by building Markov chains of behavioral models (extended version). ACM Trans. Priv. Secur.(TOPS) 22. https://doi.org/10.1145/3313391.
Paccagnella, R., Datta, P., Hassan, W.U., Bates, A., Fletcher, C., Miller, A., Tian, D., 2020. CUSTOS: practical tamper-evident auditing of operating systems using trusted execution. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2020.24065.
Packetizer, Inc., . AES crypt - Advanced file encryption. https://www.aescrypt.com. Accessed 9 August 2021.
Penta Security, 2020. 8 Most Catastrophic Ransomware Attacks in 2020. https://www.pentasecurity.com/blog/8-most-catastrophic-ransomware-attacks-in-2020/. (Accessed 9 August 2021).
Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L., 2020. Intriguing properties of adversarial ml attacks in the problem space. In: IEEE Symposium on Security and Privacy (SP). IEEE, pp. 1332–1349. https://doi.org/10.1109/SP40000.2020.00073.
Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole exe. In: Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence. https://www.aaai.org/ocs/index.php/WS/AAAIW18/paper/view/16422.
Russinovich, M.E., Margosis, A., 2011. Windows Sysinternals Administrator's Reference. Microsoft Press.
Russinovich, M.E., Solomon, D.A., Ionescu, A., 2012. In: Windows Internals, sixth ed. O'Reilly Media. Part 1.
Saracino, A., Sgandurra, D., Dini, G., Martinelli, F., 2018. MADAM: effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secure Comput. 15, 83–97. https://doi.org/10.1109/TDSC.2016.2536605.
Sartea, R., Farinelli, A., 2017. A Monte Carlo tree search approach to active malware analysis. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI-17. AAAI Press, pp. 3831–3837. In: https://www.ijcai.org/proceedings/2017/535.
Sav, S., Pyrgelis, A., Troncoso-Pastoriza, J.R., Froelicher, D., Bossuat, J.P., Sousa, J.S., Hubaux, J.P., 2021. In: POSEIDON: Privacy-preserving Federated Neural Network Learning. Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2021.24119.
Scaife, N., Carter, H., Traynor, P., Butler, K.R., 2016. Cryptolock (and drop it): stopping ransomware attacks on user data. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS 2016). IEEE, pp. 303–312. https://doi.org/10.1109/ICDCS.2016.46.
Schumilo, S., Aschermann, C., Abbasi, A., Wörner, S., Holz, T., 2020. HYPER-CUBE: high-dimensional hypervisor fuzzing. In: Network and Distributed Systems Security (NDSS) Symposium. https://doi.org/10.14722/ndss.2020.23096.
Sgandurra, D., Muñoz-González, L., Mohsen, R., Lupu, E.C., 2016. Automated Dynamic Analysis of Ransomware: benefits, Limitations and Use for Detection arXiv. https://arxiv.org/abs/1609.03020v1.
Shannon, C.E., 1948. A mathematical theory of communication. Bell Syst. Tech. J. 27, 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.
Shaukat, S.K., Ribeiro, V.J., 2018. RansomWall: a layered defense system against cryptographic ransomware attacks using machine learning. In: 2018 10th International Conference on Communication Systems & Networks (COMSNETS). IEEE, pp. 356–363. https://doi.org/10.1109/COMSNETS.2018.8328219.
Shi, H., Mirkovic, J., Alwabel, A., 2017. Handling anti-virtual machine techniques in malicious software. ACM Trans. Priv. Secur.(TOPS) 21, 1–31. https://doi.org/10.1145/3139292.
Shinagawa, T., Eiraku, H., Tanimoto, K., Omote, K., Hasegawa, S., Horie, T., Hirano, M., Kourai, K., Oyama, Y., Kawai, E., et al., 2009. 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), pp. 121–130. https://doi.org/10.1145/1508293.1508311.
Smith, M.R., Johnson, N.T., Ingram, J.B., Carbajal, A.J., Haus, B.I., Domschot, E., Ramyaa, R., Lamb, C.C., Verzi, S.J., Kegelmeyer, W.P., 2020. Mind the gap: on bridging the semantic gap between machine learning and malware analysis. In: Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security (AISec 2020). Association for Computing Machinery, New York, NY, USA, pp. 49–60. https://doi.org/10.1145/3411508.3421373.
Sun, G., Qian, Q., 2021. Deep learning and visualization for identifying malware families. IEEE Trans. Dependable Secure Comput. 18, 283–295. https://doi.org/10.1109/TDSC.2018.2884928.
Tang, F., Ma, B., Li, J., Zhang, F., Su, J., Ma, J., 2020. Ransom Spector: an introspection-based approach to detect crypto ransomware. Comput. Secur. 97, 101997. https://doi.org/10.1016/j.cose.2020.101997.
Wei, K., Li, J., Ding, M., Ma, C., Yang, H.H., Farokhi, F., Jin, S., Quek, T.Q.S., Poor, H.V., 2020. Federated learning with differential privacy: algorithms and performance analysis. IEEE Trans. Inf. Forensics Secur. 15, 3454–3469. https://doi.org/10.1109/TIFS.2020.2988575.
Widmer, G., Kubat, M., 1996. Learning in the presence of concept drift and hidden contexts. Mach. Learn. 23, 69–101. https://doi.org/10.1007/BF00116900.
Williams, L., 2021. The people who live in glass houses are happy the stones weren't thrown at them [from the editors]. IEEE Secur. Priv. 19, 4–7. https://doi.org/10.1109/MSEC.2021.3065723.
Wüchner, T., Cisłak, A., Ochoa, M., Pretschner, A., 2019. Leveraging compression-based graph mining for behavior-based malware detection. IEEE Trans. Dependable Secure Comput. 16, 99–112. https://doi.org/10.1109/TDSC.2017.2675881.
Xu, G., Li, H., Liu, S., Yang, K., Lin, X., 2020. VerifyNet: secure and verifiable federated learning. IEEE Trans. Inf. Forensics Secur. 15, 911–926. https://doi.org/10.1109/TIFS.2019.2929409.
Ye, Y., Hou, S., Chen, L., Lei, J., Wan, W., Wang, J., Xiong, Q., Shao, F., 2019. Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, pp. 4150–4156. https://doi.org/10.24963/ijcai.2019/576.
Yokoyama, A., Ishii, K., Tanabe, R., Papa, Y., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Brengel, M., Backes, M., et al., 2016. SandPrint: fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In: International Symposium on Research in Attacks, Intrusions, and Defenses (RAID 2016). Springer, pp. 165–187. https://doi.org/10.1007/978-3-319-45719-2_8.
You, W., Zhang, Z., Kwon, Y., Aafer, Y., Peng, F., Shi, Y., Harmon, C., Zhang, X., 2020. PMP: cost-effective forced execution with probabilistic memory pre-planning. In: IEEE Symposium on Security and Privacy (SP). IEEE, pp. 1121–1138. https://doi.org/10.1109/SP40000.2020.00035.
Zhang, Z., Cheng, Y., Gao, Y., Nepal, S., Liu, D., Zou, Y., 2021. Detecting hardware-assisted virtualization with inconspicuous features. IEEE Trans. Inf. Forensics Secur. 16, 16–27. https://doi.org/10.1109/TIFS.2020.3004264.
Zhang, Z., Qi, P., Wang, W., 2020. Dynamic malware analysis with feature engineering and feature learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1210–1217. https://doi.org/10.1609/aaai.v34i01.5474.