Ranker: Early Ransomware Detection Through Kernel-Level Behavioral Analysis

Huan Zhang, Lixin Zhao, Aimin Yu, Lijun Cai, and Dan Meng

***Abstract*— Ransomware is a rapidly evolving type of malware crafted to encrypt user files, rendering them inaccessible and demanding a ransom. The impact of ransomware attacks on both enterprises and individuals is significant. However, early detection of such malware remains a formidable challenge with current detection methods. In this paper, we propose Ranker, a real-time approach designed for early ransomware detection through kernel-level behavioral analysis. Analyzing various ransomware families, we discovered that half of these attacks exhibit stealthy behaviors preceding the actual attack. Extracting insights from the pre-attack malicious behavior proves effective for early detection of ransomware. For ransomware families that encrypt files directly, considering that interacting with user files is their goal, our focus is on monitoring file changes during the attack, hoping to detect ransomware when fewer files are lost. Therefore, Ranker systematically characterizes the kernel-level behavior of ransomware during the pre-attack and attack stages, identifying general and essential characteristics. Ranker also introduces a lightweight detector for real-time ransomware detection. Extensive experiments demonstrate that Ranker achieves an average F1 score of 99.43% in ransomware detection, with a mere 0.11% false positives across 68 distinct ransomware families. Notably, Ranker detects 95% of ransomware attacks with no more than one file encrypted and attains a 97.16% accuracy in identifying 22 previously unseen ransomware families.**

***Index Terms*— Ransomware detection, early detection, behavior analysis, Kernel-level behavior.**

I. Introduction

RANSOMWARE is a class of malware that typically encrypts files on infected systems and demands substantial ransom for decryption. Such ransomware attacks have become more rampant over the past few years, causing significant financial losses, with first half 2023 payments reaching $450 million, nearing the total for all of 2022 [1]. More importantly, contemporary ransomware variants exhibit accelerated encryption rates through the utilization of robust encryption algorithms or advanced multi-process techniques. Presently, the fastest ransomware, Rorschach, demonstrates an impressive capability, encrypting 220,000 files in just 4.5 minutes—twice the speed of its predecessor, LockBit 3.0 [2]. The imperative for early detection of ransomware activities is underscored by these advancements. An ideal detection method should identify ransomware as early as possible, safeguarding critical data and minimizing economic losses for organizations and individuals.

To fight against ransomware, diverse detection methods have emerged, generally falling into two categories: *static* and *dynamic*. Static methods [3], [4], [5], [6] rely on rule-based features, such as file hashes, header information, and OPcodes of executable files. However, static rules prove ineffective against polymorphism and code obfuscation techniques [7], [8], [9]. Conversely, dynamic methods employ behavior-based features for ransomware detection. Existing methods [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] focus on monitoring the process-level behavior of ransomware to distinguish it from benign processes. This process-level detection method is rational, as ransomware programs typically perform different operations compared to benign processes, such as frequent read and write activities during encryption, and a large number of file deletions and renames after encryption.

However, process-level detection methods have notable vulnerabilities that render them susceptible to evasion by multi-process techniques, junk code injection, or other evasive tactics. For instance, a ransomware program can achieve its goal by spawning multiple child processes, each executing a small portion of the overall task while mimicking benign processes to evade detection [20]. Additionally, most existing methods [10], [11], [12], [13], [14], [15] detect ransomware offline or during the attack stage, often after multiple files have already encrypted. They lack profiling the behavior of ransomware in the pre-attack stage. Recently, several methods [16], [17], [18], [19] focus on early detection by relying solely on features in the pre-attack stage. But they lack universality and are ineffective against traditional ransomware employing direct encryption. Therefore, there is an urgent need to find an *effective*, *efficient* and *universal* solution for early detection of ransomware or its variants.

To accomplish this objective, we start with a thorough analysis of diverse ransomware families to gain a deeper insight into their characteristics. We first observed that numerous ransomware families abstain from initiating encryption attacks directly. Instead, they often engage in stealthy and evasive

Manuscript received 18 November 2023; revised 4 March 2024; accepted 28 May 2024. Date of publication 6 June 2024; date of current version 11 June 2024. This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDC02040200. The associate editor coordinating the review of this article and approving it for publication was Prof. Shouling Ji. *(Corresponding author: Lixin Zhao.)*

Huan Zhang, Aimin Yu, and Dan Meng are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China, and also with the School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China (e-mail: zhanghuan@iie.ac.cn; yuaimin@iie.ac.cn; mengdan@iie.ac.cn).

Lixin Zhao and Lijun Cai are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China (e-mail: zhaolixin@iie.ac.cn; cailijun@iie.ac.cn).

Digital Object Identifier 10.1109/TIFS.2024.3410511

operations, such as deleting volume backups and terminating security protection software, actions markedly distinct from normal software behavior. This observation inspired us that mining the malicious behavior of ransomware before the attack is promising for early detection. For traditional ransomware employing direct encryption, we further explored the behavioral features during its attack phase. Considering that ransomware must interact with files, our key insight is that scrutinizing ransomware from a new perspective of files may unveil essential characteristics shared across all ransomware families, offering a more resilient detection method.

Motivated by the observations, we propose **Ranker**, a real-time method for early ransomware detection through kernel-level behavior analysis. Leveraging the rich information embedded in kernel-level events and their resistance to modification, Ranker explores the *general* and *essential* behavior of ransomware in the pre-attack and attack stages, enabling early detection. In the pre-attack stage, we identified 37 malicious commands within kernel-level events, reflecting various stealthy behaviors performed by ransomware. In the attack stage, we present a *file-level* detection method that identifies ransomware by monitoring file changes. We extracted file behavior sequences for 68 ransomware families, revealing shared patterns grouped into four categories. Importantly, these patterns are distinct from those observed in normal applications. Overall, Ranker achieves early detection by matching malicious commands in the pre-attack stage or detecting the first file that matches the behavior pattern in the attack stage. In extensive experiments involving 68 ransomware families, Ranker achieves a 99.43% detection accuracy with only 0.11% false positives. Furthermore, Ranker can detect 95% of ransomware samples with no more than one file encrypted and achieves a 97.16% detection accuracy for previously unseen ransomware families.

Our contributions can be summarized as follows:

- We find that ransomware perform various stealthy behaviors in the pre-attack stage, leaving us an opportunity to detect ransomware early.
- We find that the file behavior patterns produced by ransomware encrypting files are shared by most families. We mined four types of file behavior patterns to distinguish ransomware from normal software.
- We propose Ranker, a real-time approach for early ransomware detection through kernel-level behavioral analysis. Ranker detects ransomware in real time by matching malicious commands and detecting file behavior patterns that unique to ransomware.
- Extensive experiments have demonstrated the effectiveness, robustness and practicality of Ranker in ransomware detection.

II. Background and Related Work

A. Kernel-Level Events

Event analysis is essential for ensuring the security of computing systems. The event data can be categorized into user-level and kernel-level events according to different collection layers. User-level events stem from human-computer interaction, with API calls collected to record software behavior and purpose. However, user-level events have drawbacks such as unclear semantics and susceptibility to malicious tampering. Kernel-level events capture fine-grained behaviors by collecting system calls and related parameters, providing higher reliability and a more accurate reflection of software operations. Researchers often rely on kernel-level events for intrusion and malware detection [21], [22], [23], [24], [25].

TABLE I
Existing Methods

Method	Data	Features	Approach/Model
UNVEIL [10]	IRP	I/O access sequences, entropy, write/delete operations	Suspicious score
Redemption [11]	IRP	I/O access frequency, entropy, file type, directory traversal, write/delete operation	Malice score
CryptoDrop [12]	IRP	file type, file hash, entropy, deletion	Reputation score
RWGuard [13]	IRP	file type, file hash, entropy, file size	Benign probability
ShieldFS [14]	IRP	file type, file traversal, entropy, read/write/rename operations	Random forest
ElderRan [15]	API call	occurrence of API calls, strings, registry keys, file operation, file extension, drooped files, directory operation	Logistic regression
Ricardo et al. [16]	API call	occurrence of key API calls	Random forest
Filippo et al. [17]	API call	occurrence of key API calls	ANN
PEDA [18]	API call	sequence of API calls	Learning Algorithm

A kernel-level event contains information about the type of occurred event, when and where it occurred, its source and outcome, and the identity of any individuals or subjects associated with the event [26]. We define a kernel-level event as a quadruple: (*Subject, Object, Relation, Time*), where *Subject* is a process entity, *Object* is a system entity (i.e., process, file or network socket), *Relation* is a system call function (such as *read* and *write*), and *Time* is the event timestamp. Note that system entities are associated with a set of attributes for identification, such as the PID and process name for processes, the FileKey and file name for files.

B. Ransomware Detection

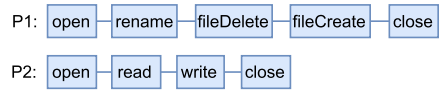
Existing crypto ransomware detection methods typically are process-level detection methods, leveraging the understanding that ransomware processes exhibit distinct behavior compared to benign processes. Such approaches commonly consist of three main components: a monitor that monitors the dynamic behavior of running processes, a feature computing module that calculates the dynamic behavior features of each process, and a classifier that labels processes with those features. Typical features used for feature computing include the sequence of I/O events or API calls, the frequency of file read/write/delete operations, and changes to files (such as file type and file content entropy). Table I summarizes the data sources, features and detection approaches used in existing methods.

Most existing methods focus on the runtime behavior of ransomware during the attack stage. UNVEIL [10] and Redemption [11] compute a suspicious score for every process, considering features such as the sequence of file I/O, write/delete operations, and entropy changes. CryptoDrop [12]

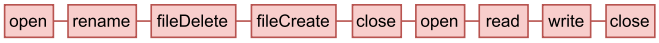
calculates a reputation score based on file type, file hash, and entropy, while RWGuard [13] similarly calculates a benign probability. ShieldFS [14] employs the relative frequencies of various disk I/O operations, including read, write, file traversal, as features fed into a random forest model to classify ransomware and normal software. EldeRan [15] utilizes similar features, along with the occurrence of API calls, fed into a logistic regression classifier. Several methods rely solely on features from the pre-attack stage for early detection but have only been tested on a very limited number of ransomware families. Molina et al. [16] selected 23 pre-attack API calls from 5 families and used a random forest model to classify ransomware. Coglio et al. [17] picked similar API calls from 12 families fed to an artificial neural network (ANN) classifier. Kok et al. [18], [19] proposed a pre-encryption detection algorithm (PEDA) to capture and analyze API calls generated by 10 families.

Time	Process	Event	FileName	Prop.FileKey
T1	P1	open	F1.pdf	---
T2	P1	rename	F1.pdf	3151528304
T3	P1	fileDelete	F1.pdf	3151528304
T4	P1	fileCreate	**F1.Vohuk**	3151528304
T5	P1	close	**F1.Vohuk**	3151528304
T6	**P2**	open	**F1.Vohuk**	3151528304
T7	**P2**	read	**F1.Vohuk**	3151528304
T8	**P2**	write	**F1.Vohuk**	3151528304
T9	**P2**	close	**F1.Vohuk**	3151528304

(a) The kernel-level events generated by Vohuk ransomware.



(b) Process-level event sequence.



(c) File-level event sequence.

Fig. 1. A motivating example for file-level detection.

C. Motivation

In general, existing methods achieve early detection only for specific ransomware families, lacking a universally effective early detection approach for broader ransomware families and their variants. In this paper, we consider two types of ransomware attacks according to their behaviors: (1) Evasive ransomware attack, corresponding to the emerging sophisticated ransomware that attempts to escape detection of existing methods. (2) Traditional ransomware attack, corresponding to the typical ransomware families that directly carry out encryption attacks.

For the first type of ransomware, existing methods typically rely on the presence of key APIs before the attack for early detection. However, recent research indicates that classifiers based on API calls significantly degrade over time as malware evolves [27]. The underlying reason for this phenomenon is that malware adapts by switching to different APIs while maintaining the same semantics. By contrast, we exploit the more stable kernel-level system calls to explore ransomware characteristics. We observed that these evasion ransomware attacks often involve stealthy behaviors before the actual attack. *This insight inspires us to utilize these pre-attack behavioral features, enabling early detection of ransomware with zero file loss*.

For the traditional ransomware attacks, existing methods typically rely on process-level features. However, this approach proves ineffective against ransomware attacks employing multi-process techniques [20], [28]. For example, Fig. 1(a) shows the kernel-level events generated by the Vohuk ransomware when encrypting the file `F1.pdf`. We can see that Vohuk first renames `F1.pdf` using the process *P1*, and then spawns the process *P2* to encrypt and write to the renamed file `F1.Vohuk`. If we observe Vohuk on a process-level basis, the two processes correspond to the two event sequences in Fig. 1(b). It can be seen that each sequence contains limited semantic information, unable to fully reveal the intention of Vohuk's behavior. Furthermore, each sequence appears to exhibit normal behavior and does not raise suspicion. However, Vohuk performs simultaneous operations of renaming, reading and writing on `F1.pdf`, making its behavior distinctly suspicious. Therefore, *our key insight is to associate all events related to each file to comprehensively observe the behavior of ransomware from a file-level perspective, as exemplified by the sequence in Fig. 1(c)*. Renaming a file does not change its *FileKey* property, as shown in Fig. 1(a), allowing us to associate events with a specific file like `F1.pdf` through *FileKey*. *FileKey* is a property of the FileIO event within the Microsoft Event Tracing for Windows component [29], [30]. Modifying this property poses a challenge for attackers, given the difficulty of acquiring kernel-level privileges on the user's system and the substantial cost associated with such modifications. Importantly, we observed consistent encryption behavior across different files for the same ransomware, inspiring us to mine distinct file-level behavior pattern for ransomware detection. We can detect the first file exhibiting this pattern during an attack to identify ransomware early and promptly block it to minimize file losses.

Through customized analyses of both ransomware attacks, focusing on behavior exploration in the pre-attack and attack stages, we can design a universally effective early detection method.

D. Threat Model

Similar to related work [11], [13], [14], [31], our design aims to safeguard Windows OS computer systems from the threat of crypto-ransomware as Windows is the primary target for most ransomware attacks [32]. In our threat model, we also assume that the underlying operating system and system audit framework are both trusted. The kernel-level events collected from kernel space are assumed not to have been tampered with. We also assume that the behaviors of ransomware are audited as kernel-level events. These assumptions are in line with prior work on kernel-level auditing [21], [22], [23].

III. Ransomware Behavior Analysis

A ransomware attack unfolds in three stages: pre-attack, attack, and post-attack. In the pre-attack stage, the ransomware

performs various stealthy operations, such as disabling security software, deleting backups, removing system logs, etc., to evade detection. The attack stage involves traversing and encrypting files within the infected system aggressively. In the post-attack stage, the ransomware displays a ransom note, notifying the victim that his/her files have been encrypted and instructing the victim to pay for decryption.

We collected kernel-level events from diverse ransomware and benign applications, analyzing them to gain insights into the distinctive behaviors of ransomware. For early detection, our focus lies on analyzing the behavior of ransomware in the first two stages: the pre-attack and attack stages.

A. Pre-Attack Stage Analysis

Analyzing diverse ransomware families revealed a common pattern: a series of stealthy actions precedes the encryption attack. These actions serve two main purposes: evading detection and enhancing the attack impact.

1) Evading Detection: Ransomware usually aims to evade detection by antivirus services on the victim's system to ensure a successful attack. Commonly, they utilize Windows OS programs like `net.exe` and `sc.exe` to locate and halt antivirus services. For example, the Hardbit ransomware family employs the command `net.exe stop avpsus /y` to disable the Kaspersky Seamless Update Service. Ransomware may also employ `taskkill.exe` to terminate specific tasks, disabling services like security protection, databases, volume management, and automatic backups. Additionally, they may delete the system logs to prevent auditing by using command `wevtutil cl system` or turn off User Account Control (UAC) to conceal their activities.

2) Enhancing the Attack Impact: For a ransomware attack to truly impact victims, attackers often target and delete volume shadow copies and system backups, preventing file recovery. Recent data indicates that over 93% of ransomware attacks attempt to destroy backup data [33]. For example, `vssadmin` is a default Windows OS utility that manages volume shadow copies, which are often used as recovery points to restore files if damaged or lost. Ransomware, such as Babuk, Avaddon, and LockBit, executes commands like `vssadmin.exe delete shadows /all /quiet` to delete these copies, rendering file restoration impossible. Other methods to delete shadow copies include using utilities like `wbadmin`, `wmic`, `PowerShell` etc. Additionally, certain ransomware families such as Phobos, Sodinokibi, and Dharma, may modify registry information to maintain permissions or elevate privileges.

These pre-attack stealthy behaviors are commonly expressed as commands in kernel-level events. Importantly, these commands are seldom utilized by benign applications, even though they are default programs in the Windows OS. Our extensive analysis of numerous ransomware families identified 37 malicious commands, detailed in Table II.

B. Attack Stage Analysis

In the attack stage, ransomware interacts extensively with user files, involving actions such as reading file contents, cryptographic operations, writing encrypted information, and erasing original content. Given these facts, our focus is on examining ransomware encryption behavior from a file-level perspective, aiming to identify its general and essential characteristics. Our investigation led to two crucial findings:

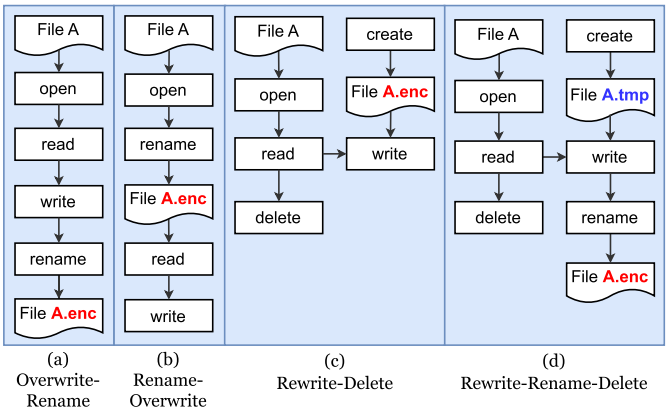


Fig. 2. The file behavior patterns of ransomware.

> **Finding 1:** The file encryption behavior of the majority of ransomware exhibits stability. Distinctive patterns in ransomware file behaviors can be mined for automated detection of known families.

We noted a consistent pattern in ransomware behavior, where a ransomware sample tends to perform identical actions on all files. Specifically, it generates similar kernel-level event sequences for encryption operations on each file. This finding inspired us to mine this consistent behavioral pattern, enabling the automated detection of these known ransomware families.

> **Finding 2:** The file encryption behavior across diverse ransomware families exhibits similarities. The behavioral patterns identified in known ransomware may possess universality, making them applicable to unknown ransomware families.

In our comprehensive analysis of 68 ransomware families, we observed a general trend: most ransomware families tend to follow similar steps when encrypting files. This finding naturally suggests that the behavior patterns unearthed in known ransomware families may also applicable to unknown ransomware families.

All ransomware conducts encryption attacks, but the specific encryption strategies can differ across various ransomware families. Our extensive experiments distilled these encryption strategies into four *general* file behavior patterns, outlined in Fig. 2. We provide a brief description for each below.

(a) **Overwrite-Rename:** Ransomware initially overwrites the target file with encrypted data and then proceed to rename the file. This pattern is adopted by the majority of ransomware families, including 40 out of 68 families, such as Avaddon, LockBit, Cuba, and Pandora.

(b) **Rename-Overwrite:** In contrast to pattern (a), ransomware initially renames the target file, and then

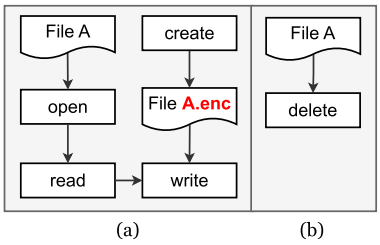


Fig. 3. The file behavior patterns of normal applications.

overwrites it with encrypted data. This pattern is observed in 29 ransomware families, including samples from AstraLocker, Babuk, Rook, and Locky ransomware families.

(c) **Rewrite-Delete:** Some ransomware first reads the target file, creates a new file, writes encrypted data to this new file, and then deletes the original file. This pattern is observed in 13 ransomware families, including Phobos, CoronaCrypt, Dharma, Mole and Darkbit, among others.

(d) **Rewrite-Rename-Delete:** This pattern resembles pattern (c), but the difference lies in creating a temporary file (such as `A.tmp`) to write the encrypted data, and then renaming it to the encrypted file (`A.enc`). This pattern is observed in Sage and EternityStealer ransomware family.

Patterns (a) and (b) generate one sequence per file, while patterns (c) and (d) involve creating a new file, resulting in two corresponding sequences. We conclude that most ransomware families follow one of these four file behavior patterns. Additionally, some families may exhibit multiple behavioral patterns simultaneously during a single run.

To validate the utility of these behavior patterns for ransomware detection, we extended our pattern mining to include normal applications with behavior resembling ransomware, such as encryption, compression, and deletion applications. Our analysis revealed two common file behavior patterns, as depicted in Fig. 3. These sequences are noticeably shorter than those of ransomware, indicating that normal software performs fewer operations on files in a brief timeframe. Pattern (a) is a common pattern observed in encryption and compression applications. It can be seen that normal software usually avoids altering the original files, refraining from overwriting or deletion operations. Additionally, normal software usually pre-creates the final encrypted or compressed file (such as `A.enc`) based on user input before encryption or compression, without involving renaming operations. Pattern (b) originates from deletion applications and behaves distinctly from ransomware. Normal software usually directly deletes the target file as per the request of the user, while ransomware encrypts the file before deleting, introducing read behavior.

Based on the extensive analysis, we believe that leveraging the distinctions in file behavior patterns between ransomware and normal applications tends to be a reliable method for ransomware detection.

IV. System Design

In this section, we outline the design of Ranker, depicted in Fig. 4, comprising two main modules: 1) Behavior Pattern Mining and 2) Real-time Detection. The Behavior Pattern Mining module explores the most distinct behavioral patterns for ransomware in the pre-attack and attack stages. The Real-time Detection module utilizes these patterns to detect ransomware in real-time from unknown applications.

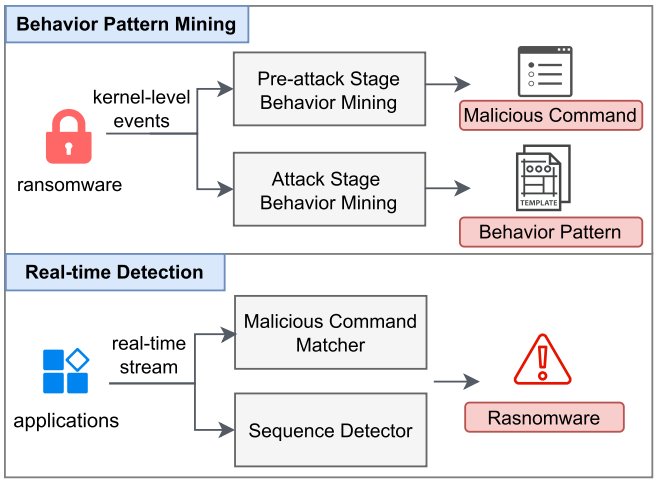


Fig. 4. An overview architecture of ranker.

A. Behavior Pattern Mining

This module aims to extract ransomware behavioral patterns from kernel-level events. Given kernel-level logs of a ransomware sample, we initially parse these logs into quadruples (i.e., kernel-level events as defined in Section II-A). The primary focus is on the `ProcessStart` event, containing commands executed by software or users, and FileIO events (e.g., `read`, `write` and `rename`) to mine file behavior patterns. This module includes two sub-modules for exploring ransomware behavior patterns in the pre-attack and attack stages, respectively.

1) Pre-Attack Stage Behavior Mining: As explained in Section III-A, we found two types of stealthy behaviors observed in ransomware. Additionally, these behaviors typically manifest themselves as execution commands in kernel-level events. Therefore, we extracted the execution commands from samples of 68 ransomware families, forming a collection of malicious commands. We have performed deduplication to remove the duplicates among these commands. However, within these commands, there might be some normal ones. Thus, we further extracted commands commonly executed by benign applications and removed these benign commands from the ransomware execution command set. For the remaining commands, we manually reviewed them. We investigated the functionality and purpose of each command through tool manuals and relevant materials, excluding commands that do not impact system security and integrity. In the end, we found 37 malicious commands most relevant to ransomware, summarized in the Table II. These commands are mainly used to evade antivirus detection and enhance the attack impact, detailed in Section III-A.

2) Attack Stage Behavior Mining: As explained in Section III-B, we found four categories of file behavior

TABLE II
MALICIOUS COMMANDS

no.	Command	Description
1	vssadmin.exe delete shadows /all /quiet	Delete volume shadow copies
2	vssadmin resize shadowstorage /for=C: /on=C:/maxsize=401MB	Resize the maximum size for a shadow copy storage association
3	wmic shadowcopy delete /nointeractive	Delete a volume shadow copy
4	bcdedit /set {default} recoveryenabled no	Disable recovery options
5	bcdedit /set {default} bootstatuspolicy ignoreallfailures	Set ignoreallfaiures option when startup
6	wbadmin delete systemstatebackup -quiet	Delete one or more system state backups
7	wbadmin delete backup -keepVersions:0 -quiet	Disable the daily backups
8	wbadmin delete catalog -quiet	Delete the backup catalog on the local computer
9	taskkill /f /im process.exe	Forcefully terminate the specified process
10	fsutil file setzerodata	Overwrite the file with zero data
11	net stop severname /y	Terminate services
12	net1 stop severname /y	Terminate services
13	netsh advfirewall set currentprofile state off	Disable Windows firewall
14	netsh firewall set opmode mode=disable	Disable Windows firewall
15	sc delete servicename	Delete a service
16	sc config servicename start=disabled	Disable the startup of a service
17	cipher /w:C	Erase all data on drive C
18	wevtutil cl security	Clear security logs from the Windows event logs
19	wevtutil cl system	Clear system logs from the Windows event logs
20	wevtutil cl application	Clear application logs from the Windows event logs
21	defrag /C /H	Clean disk fragmentation
22	icacls C:* /grant Everyone:F /T /C /Q	Grant everyone full access to all files in the C drive directory
23	attrib Default.rdp -s -h	Close the system properties and hidden properties of the remote desktop protocol program creation file Default.rdp
24	powershell -e Get-WmiObject Win32_Shadowcopy | ForEach-Object {$_.Delete();}	Obtain and delete all shadow copies
25	powershell -e Get-WmiObject Win32_Shadowcopy | Remove-WmiObject	Obtain and remove all shadow copies
26	powershell -e Get-CimInstance Win32_ShadowCopy | Remove-CimInstance	Obtain and remove all shadow copies
27	powershell -command Get-EventLog -LogName * | ForEach {Clear-EventLog $_.Log}	Obtain and clear all system event logs
28	powershell Set-MpPreference -disablerealtimeMonitoring $true	Disable Windows Defender real-time protection
29	powershell Set-MpPreference -disableBehaviorMonitoring $true	Disable Windows Defender behavior monitoring
30	powershell Set-MpPreference -disableArchiveScanning $true	Disable Windows Defender scanning archive files
31	powershell Set-MpPreference -disableBlockAtFirstSeen $true	Disable Windows Defender block at first seen
32	powershell & Enable-WindowsOptionalFeature -Online -FeatureName SMB1Protocol	Enable Windows SMB1 protocol
33	reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v Pentestlab /t REG_SZ /d C:\Users\pentestlab\pentestlab.exe	Add the maintenance rights registry to current user
34	reg add HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce /v Pentestlab /t REG_SZ /d C:\Users pentestlab\pentestlab.exe	Add the maintenance rights registry to current user
35	reg add HKCU\Software\Microsoft\Windows CurrentVersion\RunServices /v Pentestlab /t REG_SZ /d C:\Users\ pentestlab\pentestlab.exe	Add the maintenance rights registry to current user
36	reg add HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\ RunServicesOnce /v Pentestlab /t REG_SZ /d C:\ Users\pentestlab\pentestlab.exe	Add the maintenance rights registry to all user
37	reg add HKCU\Control Panel\Desktop /v Wallpaper /t REG_SZ /d	Add a registry for changing Windows desktop wallpaper

patterns of ransomware. However, extracting and generating these patterns from kernel-level events pose challenges. There are two major challenges in this procedure:

- How to accurately extract all relevant events that ransomware acts on a file from massive kernel-level events?
- Once event sequences of files are extracted, how to generate the most distinct behavioral patterns exhibited by ransomware?

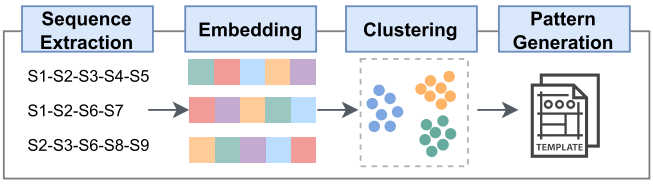


Fig. 5. The framework of attack stage behavior mining module.

To solve these challenges, Ranker devised an automated pipeline for mining the file behavior patterns of ransomware, outlined in Fig. 5. The pipeline comprises four sub-modules: 1) Sequence Extraction, 2) Embedding, 3) Clustering, and 4) Pattern Generation. To overcome the first challenge, the Sequence Extraction sub-module designs an algorithm to extract all events related to each file from kernel-level events, grouping them into a sequence. For the second challenge, considering ransomware aims to encrypt numerous files, potentially in the thousands, Ranker utilizes an embedding algorithm and clustering model to automatically identify consistent behaviors exhibited by the ransomware across these files. Subsequently, Ranker selects the most representative behavior patterns from diverse ransomware families, generating *distinct* file behavior patterns. The details are provided below.

a) Sequence extraction: To capture comprehensive semantic information revealing ransomware intentions, our approach goes beyond simple correlation of FileIO events by filename. We leverage the observation that renaming a file does not change the *FileKey* property, a unique identifier that is used to associate a file (see the motivating example in Fig. 1(a)). We designed a sequence extraction algorithm to extract and associate events with each file from kernel-level events, presented in Algorithm 1.

In Algorithm 1, given the kernel-level events $\mathcal{E}$ generated by ransomware, Ranker prioritizes FileIO events. When a file,

Algorithm 1 Sequence Extraction Algorithm

Output: Kernel-level events of a ransomware sample $\mathcal{E}$
Input: File behavior sequence set $\mathcal{S}$
1: $\mathcal{S} \leftarrow \emptyset$; PropertySet $\mathcal{A} \leftarrow \emptyset$
2: **for** each FileIO $e \in \mathcal{E}$ **do**
3: **if** Event is 'open' **then**
4: **if** FileName is new **then**
5: create an event list $L_{FileName}$ for FileName
6: **end if**
7: add e to list $L_{FileName}$
8: **else if** FileKey is new **then**
9: record $\langle FileKey, FileName \rangle$ to PropertySet $\mathcal{A}$
10: add e to list $L_{FileName}$
11: **else**
12: $OrigFileName \leftarrow \mathcal{A}$.get($FileKey$)
13: add e to original file's list $L_{OrigFileName}$
14: **end if**
15: **end for**
16: **for** each $\ell \in \mathcal{L}$ **do**
17: $s \leftarrow clearList(\ell)$
18: $\mathcal{S} \leftarrow \mathcal{S} \cup s$
19: **end for**
20: **return** $\mathcal{S}$

FileName, is accessed (i.e., `open` event received), an event list $L_{FileName}$ is created, and the `open` event is added to this list (Lines 3-7). For subsequent FileIO events (`read`, `write`, `rename`, `delete`, `fileCreate` and `fileDelete`), if the *FileKey* is new, Ranker adds the $\langle FileKey, FileName \rangle$ pair to PropertySet $\mathcal{A}$ and appends the event to list $L_{FileName}$ (Lines 8-10). If the *FileKey* corresponds to *OrigFileName* in $\mathcal{A}$, the event is added to the event list of *OrigFileName*, irrespective of whether the event filename matches OrigFileName (Lines 11-13). This ensures that the `rename` events are added to the event list of the original file. After consuming all events in $\mathcal{E}$, Ranker cleans the event list of each file to obtain a concise event sequence. The *clearList* function converts each event list $\ell = \{e_1, e_2, \cdots, e_n\}$ into a sequence s, merging and deleting consecutive recurring events or subsequences (Lines 16-19). Finally, each file corresponds to a sequence of events $s = \{e_1, e_2, \cdots, e_m\}, (m \leq n)$.

b) Embedding: After obtaining the event sequence set $\mathcal{S}$ of all files, which typically includes sequences for thousands of files, we identify the most common file behavior sequence as the behavior pattern of ransomware. Ranker achieves this by converting all file sequences into vectors using Doc2Vec model, a document embedding model in natural language processing [34]. The insight here is that the vector difference between two sequences in the embedding space reflects their semantic similarity. By group semantically similar sequences, Ranker can uncover the most common behavior sequences. Specifically, Ranker treats an event sequence as a sentence and the collective sequences of all files as a document, employing the Doc2Vec model to embed these sequences into a numerical vector space, generating their vector representations. Formally, an event sequence s is represented as an d-dimensional vector: $\mathbf{s} = [\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_d}]$.

c) Clustering: Next, Ranker clusters sequence embeddings into different groups using Mean Shift algorithm [35]. Given that ransomware not only encrypts user files but also loads system files like dynamic link libraries, these sequences will be grouped into multiple clusters. Because we collect data when ransomware completes the encryption of all files, the number of encrypted files far exceeds the number of loaded files. Therefore, Ranker selects the top two clusters with the most sequences and extracts the representative sequence in each cluster as the behavior pattern of ransomware. The reason for selecting the top two clusters is that patterns (c)Rewrite-Delete and (d)Rewrite-Rename-Delete generate two independent event sequences, one for the original file and another for the newly created file. The representativeness of each sequence in a cluster is measured by its average cosine similarity to other sequences. Given the vector representation of two sequences s_i and s_j in a cluster, Ranker calculates their cosine similarity $S(s_i, s_j)$ as follows:

$$S(s_i, s_j) = \frac{s_i \cdot s_j}{\|s_i\| \times \|s_j\|} = \frac{\sum_{v_i \in s_i} \sum_{v_j \in s_j} v_i \cdot v_j}{\sqrt{\sum_{v_i \in s_i} (v_i)^2} \times \sqrt{\sum_{v_j \in s_j} (v_j)^2}}.$$

The representativeness of a sequence s_i in cluster C is calculated by $R(s_i) = \frac{\sum_{k=1}^{n} S(s_i, s_k)}{n}$, where n is the number of sequences in cluster C. The sequence with the highest representativeness is picked out as the representative sequence, i.e., the behavior pattern of ransomware.

d) Pattern generation: Following the above steps, we mined behavior patterns for 68 ransomware families. As analyzed in Section III-B, different ransomware families may exhibit similar behavior patterns. Therefore, Ranker groups these behavior patterns using the Ratcliff-Obershelp similarity algorithm [36]. The algorithm calculates the similarity of two strings as the ratio of matching characters to the total number of characters in the strings. Let $\mathbf{P} = (p_1, p_2, \cdots, p_n)$ denote the set of behavior patterns from the 68 families. For the two behavior patterns p_i and p_j, their similarity is calculated as follows:

$$Sim(p_i, p_j) = 2 \times \frac{len\left(LCS\left(p_i, p_j\right)\right)}{len(p_i) + len(p_j)},$$

where the matched characters is computed by the *LCS* (Longest Common Substring) algorithm [37], and $len(p_i)$ represents the total number of characters in p_i. The resulting similarity value ranges from 0 to 1, where identical sequences yield a similarity value of 1, and sequences with no common characters result in a value of 0. Empirically, Ranker combines sequences with similarity values greater than 0.90. This process ultimately generates four file behavior patterns, as depicted in Fig. 2.

B. Real-Time Detection

Our goal is to detect ransomware attacks early in real-time. To achieve this, Ranker employs two detection modules: 1) Malicious Command Matcher and 2) Sequence Detector, which enable instant detection through command matching and analysis of file behavior sequences. The Malicious Command Matcher uses the 37 identified malicious commands to

check for the execution of such commands by ransomware. The Sequence Detector correlates kernel-level FileIO events for each file, identifying suspicious file behavior patterns displayed in ransomware.

In practice, Ranker continuously monitors the kernel-level event stream, performing malicious command matching and sequence detection. Upon receiving a `ProcessStart` event, Ranker extracts the `commandLine` attribute containing the executed command and checks for matches with known malicious commands. If a match is found, Ranker promptly terminates the executing process and issues an alert to the user. The malicious command matcher will detect evasive ransomware attacks before encryption, ensuring zero file loss. At the same time, Ranker extracts FileIO sequences and checks them for suspicious file behavior patterns (i.e., the mined four patterns). It terminates the malicious process when the first file exhibiting one of the behavior patterns appears. For dynamic analysis, we contend that sacrificing only a few files represents our most effective strategy.

V. Data Collection And Experimental Setup

A. Data Collection

We rely on Event Tracing for Windows (ETW) framework [29], a native event tracing framework in Windows OS, to collect the kernel-level events. We used an ETW-based open-source kernel event collector *Kellect* [38] to collect data and monitor the system in real-time.

1) User System Simulation: To conduct our experiments, we set up several virtual machines (VMs) running 64-bit Windows 10 OS. These VMs were equipped with the kernel-level event collector and various common utilities such as Adobe Reader, Microsoft Office, alternative Web browsers, and media players. To simulate a real user system, we randomly selected 2,040 files from publicly available document corpus [39], encompassing file types like `pdf`, `doc`, `xls`, `ppt`, `html`, `csv`, `jpg`, `json`, `txt`, which were common targets for ransomware. These files were distributed across different directories within the VMs to mimic a real user system. For real-time detection experiments, we deployed the Ranker prototype in VMs and ran ransomware and normal samples for evaluation.

2) Ransomware Data Collection: To the best of our knowledge, there is currently no publicly available kernel-level dataset for ransomware that can be used to evaluate Ranker. And we tend to evaluate the robustness and adaptability of Ranker, which requires diverse ransomware over a long period. To this end, we compiled a dataset ourselves comprising 90 real-world ransomware families spanning from 2008 to 2023, with 88% emerging after 2018. This dataset, collected from sources like VirusTotal [40], VirusShare [41], MalwareBazaar [42], and online repositories, is the most extensive dataset to date and twice the size of the largest dataset in previous research [43]. For each family, we ran at most fifty samples, and for families with fewer samples, we ran all available ones. The purpose here is to ensure diversity of ransomware families in our dataset, rather than duplication of a single family. Previous work also emphasizes the importance of diversity across ransomware families in evaluating detectors [12], [43]. Considering that diversity within families is limited, a small number of representative samples from each family suffices. Each ransomware sample was executed in VMs for ten minutes or until all user files were encrypted. After each run, we manually labeled the sample by its family type, and the VM was reset to an earlier snapshot to eliminate any impact from previous ransomware executions. The process took over 60 days to run all samples and collect data. We focused on samples that actively encrypted user files, excluding those with no file modifications. As a result, we obtained 1,030 active samples from 90 ransomware families. For the behavior pattern mining, we used 854 samples from 68 ransomware families to evaluate detection accuracy, listed in Table III. The remaining 176 samples from 22 unseen ransomware families were used to evaluate the robustness of Ranker, listed in Table IV. We divided the dataset according to the chronological appearance of these sample families to ensure that the ransomware families in the unseen dataset emerged after those in the pattern mining dataset, avoiding the temporal bias problem caused by the dataset split proposed by TESSERACT [44].

3) Benign Data Collection: To evaluate the robustness of Ranker against false positives, we collected two categories of benign applications, behaving similarly to ransomware and commonly used applications, listed in Table V. The first

TABLE III
RANSOMWARE FAMILIES AND SAMPLES

Family	#Samples	Family	#Samples	Family	#Samples
Adhubllka	13	Dalexis	2	Mole	6
AESRT	1	Darkside	5	Mydoom	3
AstraLocker	5	DearCry	3	Nemty	3
Avaddon	35	Dharma	43	Nitro	2
Avoslocker	18	Doydo	3	Pandora	13
Azov	42	Elbie	2	Phobos	14
Babuk	50	Eyedocx	2	PwndLocker	1
Bitlocker	3	GandCrab	10	Pony	3
BlackBit	2	GlobeImposter	31	Ragnar	6
BlackMatter	40	GoldenEye	3	Ranzy	5
BlackOut	7	Hades	2	Rook	5
Blaze	3	Hive	12	Ryuk	11
Cerber	50	HolyGhost	2	Sage	3
Chaos	35	keygroup	2	Seven	5
Chernobyl	1	LockBit	50	Sodinokibi	38
Cipher	4	LockerGoga	19	Solidbit	3
Clop	2	LokiLocker	12	Teslacrypt	9
Conti	40	Locky	10	Thanos	15
CoronaCrypt	5	Matrix	11	VasaLocker	2
Cryakl	11	Makop	34	Venus	15
Crypt360	2	MafiaWare666	2	Xorist	21
CryptFile2	4	Mespinoza	19	ZeusSec1337	1
Cuba	11	Medusa	8	-	
Total samples: 854					

TABLE IV
UNSEEN RANSOMWARE FAMILIES AND SAMPLES

no.	Family	#Samples	no.	Family	#Samples
1	Animagus*	5	12	Mallox	21
2	Bianlian	11	13	MoneyMessage	11
3	BlackBasta	37	14	Nef1lim	6
4	BlackSnake	4	15	nokoyawa	10
5	Cylance	2	16	Play	13
6	CryptXXX	2	17	PlutoCrypt	8
7	Darkbit	6	18	RansomEXX	5
8	DarkyLock	5	19	REDTM	3
9	DATAFLocker	2	20	Targeted	4
10	EternityStealer	5	21	Vohuk	10
11	Hardbit	6	22	Zeppelin	3
Total samples: 176					

TABLE V
BENIGN APPLICATIONS

Tool	Applications	Version	#Samples
Encryption	Crypt4Free	8.20	450
	FlashCrypt	1.0	200
	EncryptFiles	1.5.0.3	250
	File Encryption XP	1.7.395	250
	MEO Encryption Software	2.18	250
	Best Folder Encryptor	16.83	200
	Advanced Folder Encryption	6.75	250
	DiskCryptor	0.8	150
	BestCrypt Volume Encryption	5.11.4	200
Compression	7-zip	22.01	250
	WinRAR	6.22.0	200
	PeaZip	9.2.0	150
	WinZip	26.0	150
	Bandizip	6.26	200
	BreeZip	1.4.29	150
Deletion	Secure File Delete	1.1	100
	Windows Delete	-	100
Tools	Adobe Reader	23.3.20284.0	200
	Notepad	10.0.19041.1865	200
	Microsoft Word	16.0.16731.20170	200
	Microsoft Excel	16.0.16731.20170	200
	Microsoft Powerpoint	16.0.16731.20170	200
Browser	Google Chrome	116.0.5845.179	100
	Firefox	114.0.2.8570	100
Interleaved Scenario	-	-	100
Total samples: 4,800			

category included 9 encryption, 6 compression, and 2 deletion applications from online repositories [45], [46]. For the second category, we collected 7 widely used applications such as Microsoft Word, Notepad, Adobe Reader, etc. Acknowledging the prevalence of normal software over malware in the real-world scenario [44], we oversampled normal samples. We ran each function of each application 50 times based on its functionality (e.g., compression and decompression 50 times each). For applications interacting with files, we considered their diverse interactions with files of different sizes and types, thus sampling multiple samples for each application. Special emphasis was placed on benign encryption software due to its behavior similarity to ransomware. For such software, we collected data on different configurations, including various encryption algorithms and whether the original file was deleted. Additionally, we examined scenarios where multiple applications ran simultaneously to test Ranker on interleaved operations. In total, we collected 4,800 kernel-level logs for benign applications.

B. Experiment Setup

We implement Ranker on Windows 10 with Python 3.8, with around 3K lines of code for all its components. For sequence embedding, we use the Gensim library(4.3.1) [47] to deploy Doc2Vec on these event sequences. The Doc2Vec model is set up with epoches to 10, windows size to 5, and embedding size to 50. For clustering, we implement Mean Shift clustering algorithm using the Sklearn library(1.3.0) [48].

Baselines: In Section II-B, we showed that various ransomware detection methods were proposed. However, they withhold their data and source codes. Table I illustrates that their data sources primarily revolve around I/O Request Packets (IRPs) and Application Programming Interface (API) calls, while ours is based on kernel-level audit logs. Given the distinct information in these logs, transferring their methods to our data is challenging. Notably, these methods commonly utilize ransomware features such as FileIO operations (used by [10], [11], [12], [14], and [15]) and API call sequences (used by [15], [17], [18], and [16]). To align with these common features, we extracted similar ones from our kernel-level data: FileIO operations and system call sequences. Furthermore, combining two of their latest detection models: Random Forest (RF) and Artificial Neural Network (ANN), we implemented four baseline detection methods to compare with Ranker. The four baseline detection methods are detailed as follows:

- Syscall+RF: We extract the system call (syscall) sequence from the ransomware process, generating a binary vector based on syscall occurrence as the feature vector of ransomware. Subsequently, we use a random forest model to classify ransomware and benign software.
- Syscall+ANN: Similarly, we extract the system call sequence vector generated by the ransomware process as its feature vector. We then employ an artificial neural network classifier to train these vectors.
- FileIO+RF: Leveraging the statistics of file I/O operations (such as read, write, delete, rename, and directory traversal operations) performed by the ransomware process as features, we train a random forest model to classify ransomware and benign software.
- FileIO+ANN: Similarly, using the statistics of file I/O operations as ransomware features, we train an artificial neural network model based on these features.

VI. EVALUATION

In this section, we evaluate the effectiveness and efficiency of Ranker for ransomware detection. Specifically, our evaluation answers the following research questions:

- **RQ1:** How effective is Ranker at detecting ransomware?
- **RQ2:** Is Ranker robust to unseen ransomware families?
- **RQ3:** How fast is Ranker for ransomware detection? How many files were lost when Ranker detected ransomware?
- **RQ4:** Is the proposed components effective in enhancing the performance of Ranker?
- **RQ5:** Is the runtime overhead of Ranker low enough to make it efficient and suitable for practical scenarios?

A. Detection Accuracy (RQ1)

We use the ransomware samples from 68 families in Table III and all benign samples to evaluate the detection accuracy of Ranker and baseline methods. We chose 20% of ransomware samples and benign samples for exploring behavior patterns, reserving the remaining 80% for a test set to evaluate the robustness of Ranker. The dataset division also follows the temporal criteria proposed by TESSERACT [44]. The training set includes ransomware samples from 2020 and earlier, while the test set comprises samples from 2020 and beyond. Here, the test set includes variant samples of ransomware families already present in the training set, unlike the unseen family dataset. We employed a small training set (only 20% of the entire dataset) to better align with real-world scenarios, where training sets are typically limited compared to the extensive test samples. The detection accuracy is measured using false negative rates (FNR), false positive rates (FPR), precision, recall and F1-score metrics.

TABLE VI
DETECTION RESULTS OF BASELINES AND RANKER

Methods	FNR(%)	FPR(%)	Pre.(%)	Rec.(%)	F1(%)
Syscall+RF	20.48	11.97	79.33	79.52	79.42
Syscall+ANN	40.36	4.21	90.37	59.64	71.86
FileIO+RF	5.81	4.82	95.39	94.19	94.75
FileIO+ANN	7.64	15.34	73.87	92.36	82.09
Ranker	**1.02**	**0.11**	**99.89**	**98.98**	**99.43**

TABLE VII
DETECTION RESULTS ON UNSEEN RANSOMWARE FAMILIES

Methods	#Detected	Precision(%)
Syscall+RF	138	78.41
Syscall+ANN	112	63.64
FileIO+RF	159	90.34
FileIO+ANN	146	82.95
Ranker	**171**	**97.16**

Table VI compares the performance of different methods. As can be seen, Ranker achieves the best performance. Specifically, Ranker achieved 99.89% precision, 98.98% recall, and 99.43% F1-score, with a low false negative rate of 1.02% and an impressively low false positive rate of 0.11%. While FileIO+RF performs better than other baseline methods, it is still worse than Ranker, e.g., 94.75% compared to 99.43% in F1-score, reflecting a 4.68% lower performance. Syscall+RF and FileIO+ANN demonstrate relatively close detection accuracy around 80%, while Syscall+ANN has the worst performance with an F1-score of 71.86%, 27.57% lower than Ranker. It is noteworthy that we used 80% of the data to train these baselines, while Ranker only used 20% of the data and achieved better performance. This indicates that Ranker has greater detection effectiveness for known ransomware families. The kernel-level behavioral features from ransomware samples up to 2020 remain highly applicable in samples emerging post-2020. Additionally, among the baseline methods, FileIO operation features achieve higher detection accuracy than Syscall sequences, highlighting the effectiveness of statistical FileIO features in characterizing ransomware behavior. Ranker, which also focuses on mining the file I/O features, emphasizes the significance of this factor in its success.

1) False Positive Analysis: Minimizing false positives is critical for practical ransomware detectors, as excessive false positives can disrupt users and compromise system effectiveness. Ranker maintains a low false positive rate of 0.11%. Upon investigation, Ranker misidentified 4 benign samples from EncryptFiles due to a configuration that deletes the original file during encryption. These samples initially created the final encrypted file (e.g., `A.enc`), read the original file, wrote to `A.enc`, deleted the original file, and finally renamed `A.enc` to the name of original file, resembling the ransomware pattern (c)Rewrite-Delete. However, it is crucial to highlight that other benign encryption software refrains from executing the final renaming step, keeping only the encrypted file. This distinction allows Ranker to effectively distinguish them from ransomware. Despite these instances, the false positive rate of 0.11% remains acceptable in practice, outperforming other baselines by an average 9%.

2) False Negative Analysis: The false negative rate of Ranker is 1.02%, with a total of 7 false negatives. We manually investigated them and summarized two reasons. Firstly, 5 false negatives only performed read and write operations during file encryption, lacking renaming or other distinctive actions, allowing them to go undetected. However, this behavior was observed in only two ransomware families, as most ransomware typically involves file renaming to attract the attention of victims. Secondly, the remaining 2 false negatives directly overwrite files with random content, without performing any read operations on the files, thereby evading detection. This behavior was observed in just one ransomware family.

B. Robustness Against Unseen Families (RQ2)

We use ransomware families listed in Table IV to further evaluate the robustness of Ranker against unseen families. A total of 176 samples from 22 unseen families are tested.

Tablel VII presents the detection results on these unseen families for both baselines and Ranker. Ranker effectively identified 171 out of 176 new and unseen ransomware samples, achieving a 97.16% detection rate. This rate is notably higher than Syscall+ANN by 33.52%, Syscall+RF by 18.75%, FileIO+ANN by 14.21%, and FileIO+RF by 6.82%. The result affirms the strong generalization and robustness of Ranker against unknown ransomware families. We attribute this success to Ranker's superior ability in identifying the *general* and *essential* behavior characteristics of ransomware compared to other methods. This confirms our hypothesis that, despite the adoption of diverse encryption strategies and new evasion technologies by new ransomware families, they still share many common kernel-level behaviors, such as common malicious commands and file behavior patterns. Therefore, we can anticipate Ranker's continued effectiveness in detecting future ransomware attacks.

1) Case Study: Ransomware family Animagus, labeled as number 1 in Table IV, is an elaborately designed ransomware attack by Zhou et.al [20]. Exploiting the limitations of existing process-level detection methods, they crafted Animagus as an imitation-based ransomware attack. Animagus imitates the behaviors of benign programs to disguise its encryption tasks and employs multi-process technology to divide the encryption task into sub-tasks, evading process-level detection. Zhou et.al [20] provided an executable sample of Animagus, along with several behavior patterns imitating normal software. We ran the sample five times, imitating five different benign software to evaluate Ranker. The results show that Ranker successfully detected all five samples. The key factor lies in the file-level detection method of Ranker, which monitors changes in each file rather than focusing on the behavior of individual processes. The file behavior of Animagus aligns with the typical behavior pattern of ransomware, making it detectable by Ranker.

C. File-Loss Analysis (RQ3)

In this section, we assess the efficacy of Ranker by evaluating its speed in ransomware detection and quantifying the

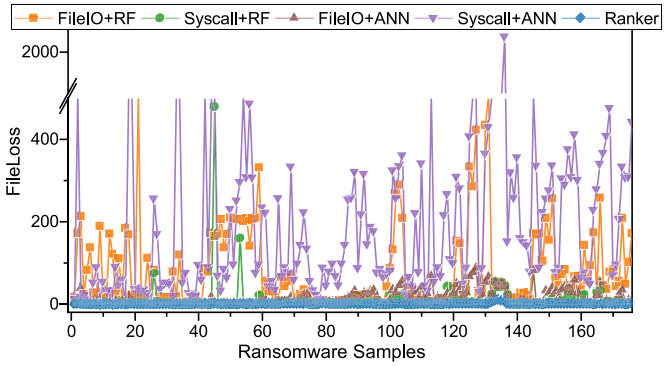


Fig. 6. The statistics of file loss.

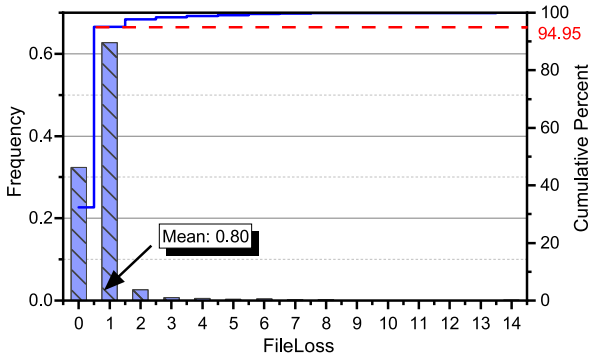


Fig. 7. The statistics of the file loss of Ranker.

extent of file losses. In our experiments, we observed that 70% of ransomware exhibits a delayed onset of attacks following execution, frequently remaining inactive for a period before encrypting files. Consequently, the utilization of detection time, defined as the interval from the initiation of ransomware execution to its detection, may not be a suitable criterion for evaluating detector efficiency. Given that the goal of ransomware is to encrypt as many files as possible, we contend that if a ransomware detector can detect ransomware when fewer files are encrypted, it also demonstrates its efficient detection speed. Therefore, here we adopt the number of file losses as the criterion to evaluate the efficiency of Ranker.

During real-time detection tests involving 176 ransomware samples outlined in Table IV, Fig. 6 depicts file loss statistics (y-axis) across the samples (x-axis) for Ranker and baseline methods. Ranker exhibits the smallest average file loss, with only 0.81 files lost, fluctuating between 0 and 10. Unlike Ranker, the file loss for other methods fluctuates significantly with changes in samples. The worst-performing method is Syscall+ANN, with an average file loss of 435, and a maximum file loss of 2037. The average file losses for FileIO+RF, Syscall+RF, and FileIO+ANN are 96, 49, and 41, respectively, fluctuating between 5 and 643. This result aligns consistently with the detection performance of these methods (as shown in Table VII). For instance, Syscall+ANN, displaying the poorest detection performance, also incurs the highest file loss. Notably, methods relying on syscall features tend to detect ransomware earlier (with fewer file losses) compared to those relying on FileIO features. This difference may arise from the fact that syscall features only need to check if specific syscalls are invoked, while FileIO features require the accumulation of statistical features over time, resulting in longer detection times. In contrast, Ranker mines behavioral features in the pre-attack and early stages, relying on rapid matching of malicious commands and detecting file sequences for early detection.

To thoroughly evaluate the impact of Ranker on file loss, we employed it for real-time detection across all ransomware samples (totaling 1,030). The results are shown in Fig. 7, with the x-axis representing the number of file losses. Notably, Ranker achieved a minimum loss of 0 files and a maximum loss of 14 files (0.58% of total files). The left y-axis represents the frequency of each loss number. On average, Ranker detected ransomware with a loss of 0.80 files. Specially, over 60% of ransomware samples were detected with only 1 file loss, and 32.23% of samples were detected without any file loss. The right y-axis represents the cumulative percentage, indicating that Ranker detected 94.95% of samples with no more than 1 file loss. These results highlight the ability of Ranker to quickly detect ransomware with very little file loss, showcasing the effectiveness of its real-time detection component. Ranker underlines pre-attack behavioral characteristics to ensure zero file loss, and its file-level detection method allows swift detection when the event sequence of one file matches the behavior patterns.

D. Ablation Study (RQ4)

In this section, we examine the impact of two key components of Ranker: behavior pattern mining and real-time detection. For behavior pattern mining, we highlight key observations and mining effects. For real-time detection, we evaluate the performance of two detectors independently.

1) Observation of Ransomware Family: In Section III-B, we discovered that samples within a family share almost identical behavioral patterns, and multiple families may exhibit similar patterns at runtime. To prove it, we randomly choose five samples from the Pandora family and one sample each from the Play, Ryuk, Xorist, Venus, Rook, and Vohuk families, totaling 11 samples. We evaluated the similarity of their runtime file behavior sequences using the Ratcliff-Obershelp Algorithm, as illustrated in Fig. 8. Notably, all five Pandora samples showed a similarity of 1, indicating an exact match in their file behavior sequences. Moreover, samples from Play, Ryuk, and Xorist exhibited similarities between 0.9 and 1 compared to Pandora, suggesting comparable or identical runtime file behavior sequences. Similarly, the three samples from Venus, Rook and Vohuk also demonstrated similar sequences. However, it is essential to note that different ransomware families can still manifest distinct file behavior sequences. For example, the sequences in Fig. 8 can be clearly divided into two groups, corresponding to two types of file behavior patterns. This clear distinction validates the effectiveness of Ranker in accurately extracting file behavior sequences, proving the efficacy of our sequence extraction algorithm.

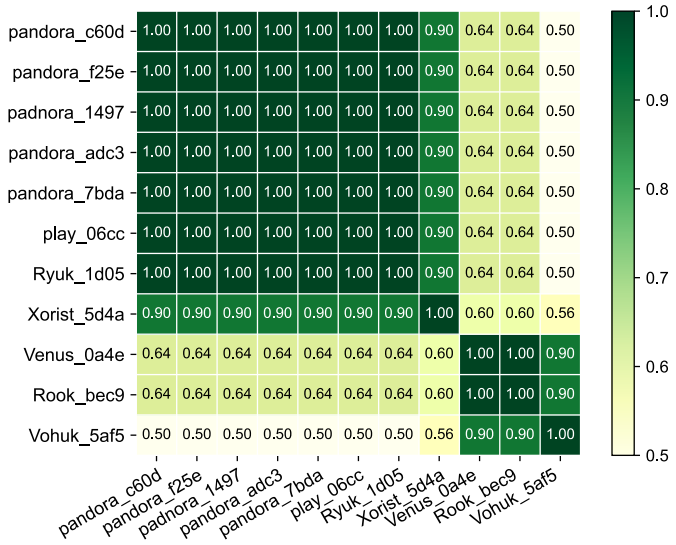
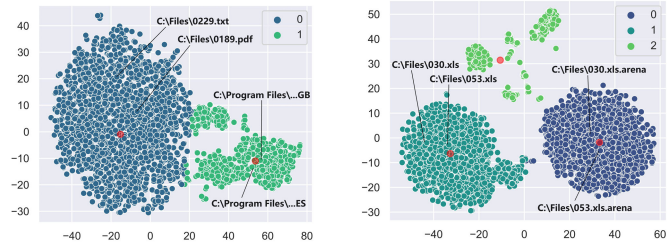


Fig. 8. The file behavior sequence similarity of ransomware samples from different families.



(a) LockBit_b2bb (b) Dharma_01e1

Fig. 9. The visualization of file behavior sequences of ransomware samples.

2) Visualization of Behavior Sequence: In this subsection, we visually evaluate the effectiveness of behavior pattern mining by mapping ransomware file behavior sequences in the embedding space using t-SNE [49] for a two-dimensional representation. Taking the two ransomware samples in Fig. 9 as examples, their behavior sequences are clearly clustered. Fig. 9(a) shows a sample from LockBit family following the pattern (a)Overwrite-Rename. Its sequences fall into two groups: one represents LockBit-encrypted files (blue), and the other denotes ordinary system files (green) accessed by LockBit. The significant operational differences result in markedly different behavior sequences, with LockBit encrypting a larger number of files. Ranker thus selects representative sequences from the cluster with the most sequences as the behavior pattern of this ransomware sample. For another sample from the Dharma family, following the pattern (c)Rewrite-Delete, its sequences are clustered into three groups due to the creation of independent sequences for original and encrypted files. Ranker appropriately selects the first two clusters with the most sequences as representative sequences. The result underscores the stability of the file behavior of ransomware, allowing Ranker to effectively mine distinctive behavioral patterns.

3) Model-Specific Detection Accuracy: Ranker employs two primary detectors: malicious command matcher (Ranker-MCM) and the sequence detector (Ranker-SD). Here, we show the effectiveness of each component in detecting ransomware.

TABLE VIII
DETECTION RESULTS OF TWO DETECTORS

Detector	FNR(%)	FPR(%)	Pre.(%)	Rec.(%)	F1(%)
Ranker-MCM	49.32	0	100	50.68	67.27
Ranker-SD	3.27	0.19	99.88	96.73	98.28

Table VIII presents the evaluation results for both detectors. Ranker-SD achieves comparable performance to Ranker, with 99.88% precision, 96.73% recall, and a 98.28% F1-score. This underscores the accuracy of the mined file behavior patterns in capturing ransomware characteristics, contributing significantly to the overall effectiveness of Ranker. In contrast, Ranker-MCM achieves 100% precision but the false negative rate is 49.32%. This is expected as Ranker-MCM relies on rule-based matching of malicious commands, accurately detecting ransomware exhibiting such behaviors. However, its effectiveness is limited to ransomware that displays these malicious commands. Overall, both detectors play crucial roles in Ranker and complement each other. Ranker-MCM excels in detecting ransomware in the pre-attack stage, ensuring zero file loss. Meanwhile, Ranker-SD proves effective against a broad spectrum of ransomware, minimizing file loss. Their collaborative operation enhances overall performance by approximately 1%, underscoring the complementary nature of these two detectors.

E. System Performance (RQ5)

In order to examine the feasibility of Ranker in a real system, we study its efficiency in terms of pattern mining overhead and real-time detection overhead. Our experiments were conducted on a computing device equipped with two Intel Core i5-1240P (1.70GHz) CPUs and 16GB RAM, running 64-bit Windows 10 operating system.

1) Pattern Mining Overhead: Since the commands executed by programs can be extracted directly through `CommandLine` field, this can be done in a shorter time during real-time detection. Here, we focus on the time overhead for attack stage behavior mining, involving four parts: sequence extraction, embedding, clustering, and pattern generation. In the sequence extraction phase, it takes an average of 0.01 seconds per file, efficiently captures event sequences. Training the doc2vec model in the embedding stage takes an average of 10.42 seconds. Clustering takes 35.18 seconds, and extracting representative sequences involves similarity calculations, which takes an average of 313 seconds. The pattern generation phase takes an average of 30.13 seconds. It is crucial to note that behavior pattern mining is an offline process, conducted once for each ransomware sample.

2) Real-Time Detection Overhead: During real-time detection, we evaluate the CPU and memory overhead of Ranker in an idle state (such as basic operations like office tasks) and an intensive state (running large software and 3D games). Due to Ranker focusing only on `ProcessStart` and FileIO events, during the idle state, the average CPU usage of Ranker is less than 1%. In the intensive state, the average CPU usage of Ranker is 7.2%. We attribute the increase in CPU usage to the need to process more events and associated event sequences.

TABLE IX
COMPARISON WITH EXISTING APPROACHES

Method	File lost	#Families	Pre-attack features	File-level/ process-level features	Real-time
UNVEIL [10]	-	8	✗	process-level	✗
Redemption [11]	5	29	✗	process-level	✓
CryptoDrop [12]	10	14	✗	file-level	✗
RWGuard [13]	partial recovery	14	✗	process-level	✓
ShieldFS [14]	recovery	11	✗	process-level	✗
ElderRan [15]	-	11	✗	process-level	✗
Ricardo et al. [16]	-	5	✓	process-level	✗
Filippo et al. [17]	-	12	✓	process-level	✗
PEDA [18]	-	10	✓	process-level	✗
Ranker	**<1**	**90**	✓	**file-level**	✓

Memory consumption is less than 1% in both states. Since Ranker cleans and merges file event sequences during real-time detection, it greatly reduces the pressure on memory storage. We believe that the performance overhead of Ranker is manageable for ransomware detection in practical scenarios and will not interfere with normal user experience.

VII. DISCUSSION AND LIMITATION

A. Comparison With Existing Solutions

Ranker and Existing solutions use different data sources, precluding a direct comparison. Instead, we compare Ranker with existing ransomware detection methods in five key aspects, as outlined in Table IX.

In ransomware detection, a critical metric is the number of file loss before detection. Ranker demonstrates excellent performance in detecting ransomware with an average encryption of no more than one file, outperforming other solutions. In addition, the number of ransomware families is also important for ransomware detection, which can evaluate the generation and robustness of detectors. These methods were evaluated on a limited number of ransomware families, with the highest being Redemption [11], encompassing 29 families. In contrast, Ranker underwent testing across 90 distinct ransomware families. Most methods neglect ransomware characteristics in the pre-attack stage, hindering early detection. Coglio et al. [17], PEDA [18] and Molina et al. [16] only consider characteristics of the pre-attack stage and are ineffective for traditional ransomware. Conversely, Ranker considers both pre-attack and attack stages, detecting ransomware with escape behavior in advance and addressing traditional ransomware. Notably, only two methods, CryptoDrop [12] and Ranker, apply file-level features, while Ranker showing fewer file losses than CryptoDrop. Additionally, Ranker enables real-time detection for rapid response to ransomware attacks.

B. Evasion Possibility

With a sufficient understanding of our detection method, attackers may attempt various methods to modify their ransomware and evade the two detectors of Ranker. First, to evade the malicious command matcher, attackers would need to avoid using the 37 malicious commands we have mined. This increases the likelihood of their activities being detected by antivirus software installed on the user's system. Additionally, users may have the option to restore files based on backups, undermining the attackers' goals. Since these malicious commands are native programs in the Windows OS, if attackers still aim to escape detection, they would need to construct similar functional functions on their own. However, this approach undoubtedly adds complexity to the ransomware and does affect the performance and stability, making it less conducive to achieving the core objective of encrypting files quickly and extensively. We will also continue to monitor the evolution of ransomware and promptly update the malicious command set.

To evade the sequence detector, an attacker may attempt to obfuscate the file sequence by intentionally introducing irrelevant events. However, we only associate FileIO events for each file, focusing specifically on a limited number of event types (8 types in this work). If attackers introduce multiple different types of events, distinct from those of legitimate software, it becomes easy to match our behavior pattern. And if attackers only add redundant events, during the Ranker's sequence extraction process, redundant events or sequences will be filtered out. Additionally, *FileKey* is a crucial attribute of Ranker, used to associate all events related to a file. As *FileKey* is a property within the kernel-level component of ETW, if attackers intend to modify this property to disrupt Ranker's extraction of file event sequences, they would need to carry out kernel-level modifications to our kernel-level tracing tool. This proves to be challenging for attackers, as obtaining kernel-level privileges on the target user's system is not easily achievable. Additionally, the cost of modifying a kernel-level tool is considerable. Meanwhile, there are also corresponding tools currently available to ensure the integrity and security of kernel-level tracing [50], [51], which can strengthen the defensibility of Ranker.

C. Limitations and Future Work

This paper achieved satisfactory detection results by focusing on two behavioral characteristics of ransomware: malicious commands and file behavior patterns. However, there are still instances of false positives and false negatives. Further exploration into the kernel-level behavioral features of ransomware could enhance the performance of Ranker. In addition, while Ranker can detect ransomware when no more than one file lost, we can explore solutions for zero file loss by incorporating additional static or dynamic features of ransomware. On the other hand, the recovery of encrypted files and predicting the future stealthy behaviors and malicious commands of ransomware will also be a focal point of our future work.

VIII. CONCLUSION

This paper proposes Ranker, a real-time approach for early ransomware detection through kernel-level behavior analysis. Ranker characterized the kernel-level behavior of ransomware in the pre-attack and attack stages, and discovered its general and essential characteristics. The experiments show that Ranker effectively distinguishes known ransomware and is also robust to unknown ransomware. Additionally, Ranker can detect most ransomware in its early stage, significantly reducing file loss.

REFERENCES

[1] Wired. (2023). *Ransomware Attacks Are on the Rise*. [Online]. Available: https://www.wired.co.uk/article/ransomware-attacks-rise-2023

[2] MonsterCloud. (2023). *The Rising Reign of Rorschach As the Fastest Ransomware in the Cybersecurity Landscape*. [Online]. Available: https://monstercloud.com/blog/2023/04/24/the-rising-reign-of-rorschach-as-the-fastest-ransomware-in-the-cybersecurity-landscape/

[3] M. Medhat, S. Gaber, and N. Abdelbaki, "A new static-based framework for ransomware detection," in *Proc. IEEE 16th Intl. Conf. Dependable, Autonomic Secur. Comput., 16th Intl. Conf. Pervasive Intell. Comput., 4th Intl. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr.(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 710–715.

[4] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A framework for analyzing ransomware using machine learning," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2018, pp. 1692–1699.

[5] S. Sheen and A. Yadav, "Ransomware detection by mining API call usage," in *Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI)*, Sep. 2018, pp. 983–987.

[6] A. Vehabovic et al., "Data-centric machine learning approach for early ransomware detection and attribution," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp.*, Jun. 2023, pp. 1–6.

[7] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 421–430.

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

[9] F. Aldauiji, O. Batarfi, and M. Bayousif, "Utilizing cyber threat hunting techniques to find ransomware attacks: A survey of the state of the art," *IEEE Access*, vol. 10, pp. 61695–61706, 2022.

[10] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 757–772.

[11] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2017, pp. 98–119.

[12] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop It): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

[13] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A real-time detection system against cryptographic ransomware," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 114–136.

[14] A. Continella et al., "ShieldFS: A self-healing, ransomware-aware filesystem," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, Dec. 2016, pp. 336–347.

[15] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," 2016, *arXiv:1609.03020*.

[16] R. M. A. Molina, S. Torabi, K. Sarieddine, E. Bou-Harb, N. Bouguila, and C. Assi, "On ransomware family attribution using pre-attack paranoia activities," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 19–36, Mar. 2022.

[17] F. Coglio, A. Lekssays, B. Carminati, and E. Ferrari, "Early-stage ransomware detection based on pre-attack internal API calls," in *Proc. 37th Int. Conf. Adv. Inf. Netw. Appl.*, vol. 2, 2023, pp. 417–429.

[18] S. Kok, A. Abdullah, N. Jhanjhi, and M. Supramaniam, "Prevention of crypto-ransomware using a pre-encryption detection algorithm," *Computers*, vol. 8, no. 4, p. 79, Nov. 2019.

[19] S. H. Kok, A. Abdullah, and N. Jhanjhi, "Early detection of crypto-ransomware using pre-encryption detection algorithm," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 5, pp. 1984–1999, May 2022.

[20] C. Zhou et al., "Limits of I/O based ransomware detection: An imitation based attack," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2023, pp. 2584–2601.

[21] W. U. Hassan et al., "NoDoze: Combatting threat alert fatigue with automated provenance triage," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–13.

[22] A. Alsaheel et al., "Atlas: A sequence-based learning approach for attack investigation," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 3005–3022.

[23] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1813–1830.

[24] Y. A. Ahmed, B. Koçer, S. Huda, B. A. Saleh Al-rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *J. Netw. Comput. Appl.*, vol. 167, Oct. 2020, Art. no. 102753.

[25] X. Zhang et al., "An early detection of Android malware using system calls based machine learning model," in *Proc. 17th Int. Conf. Availability, Rel. Secur.*, Aug. 2022, p. 92.

[26] Wikipedia. (2023). *Audit Trail*. [Online]. Available: https://en.wikipedia.org/wiki/Audit_trail

[27] X. Zhang et al., "Enhancing state-of-the-art classifiers with API semantics to detect evolved Android malware," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 757–770.

[28] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, and L. V. Mancini, "Evading behavioral classifiers: A comprehensive analysis on evading ransomware detection techniques," *Neural Comput. Appl.*, vol. 34, no. 14, pp. 12077–12096, Jul. 2022.

[29] Microsoft. (2021). *Event Tracing for Windows (ETW)*. [Online]. Available: https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows–etw-

[30] (2021). *FileIo_Info Class*. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/etw/fileio-info

[31] A. A. Elkhail et al., "Seamlessly safeguarding data against ransomware attacks," *IEEE Trans. Dependable Secur. Comput.*, vol. 20, no. 1, pp. 1–16, Jan. 2023.

[32] F. Tang, B. Ma, J. Li, F. Zhang, J. Su, and J. Ma, "RansomSpector: An introspection-based approach to detect crypto ransomware," *Comput. Secur.*, vol. 97, Oct. 2020, Art. no. 101997.

[33] (2023). *Global Report Ransomware Trends*. [Online]. Available: https://www.veeam.com/ransomware-trends-report-2023?ad=homepage-feature-banner

[34] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. 31st Int. Conf. Mach. Learn.*, vol. 32, Jan. 2014, pp. 1188–1196.

[35] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[36] P. E. Black. (2021). *Ratcliff/Obershelp Pattern Recognition*. [Online]. Available: https://www.nist.gov/dads/HTML/ratcliffObershelp.html

[37] M. Paterson and V. Dančík, "Longest common subsequences," in *Proc. Int. Symp. Math. Found. Comput. Sci.*, 1994, pp. 127–142.

[38] T. Chen, Q. Song, X. Qiu, T. Zhu, Z. Zhu, and M. Lv, "Kellect: A kernel-based efficient and lossless event log collector for windows security," 2022, *arXiv:2207.11530*.

[39] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing science to digital forensics with standardized forensic corpora," *Digit. Invest.*, vol. 6, pp. S2–S11, Sep. 2009, doi: 10.1016/j.diin.2009.06.016.

[40] (2023). *VirusTotal*. [Online]. Available: https://www.virustotal.com/

[41] (2023). *Virus Share*. [Online]. Available: https://virusshare.com/

[42] (2023). *Malware Bazaar*. [Online]. Available: https://bazaar.abuse.ch/

[43] M. E. Ahmed, H. Kim, S. Camtepe, and S. Nepal, "Peeler: Profiling kernel-level events to detect ransomware," in *Proc. 26th Eur. Symp. Res. Comput. Secur.*, 2021, pp. 240–260.

[44] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 729–746.

[45] (2023). *Portable Apps*. [Online]. Available: https://portableapps.com/

[46] (2023). *Informer*. [Online]. Available: https://software.informer.com/

[47] Gensim. (2023). *Topic Modelling for Humans*. [Online]. Available: https://radimrehurek.com/gensim/index.html

[48] Sklearn. (2023). *Scikit-Learn Machine Learning in Python*. [Online]. Available: https://scikit-learn.org/stable/

[49] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 1–12, Sep. 2008.

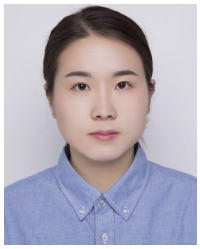
[50] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2020, pp. 1551–1574.

[51] R. Paccagnella et al., "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–16.

Huan Zhang received the B.S. degree from Yanshan University in 2019. She is currently pursuing the Ph.D. degree with the Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include malware detection, cyber security, and data security.

Lixin Zhao received the B.S. degree from Nankai University in 2013 and the Ph.D. degree in computer sciences from the University of Chinese Academy of Sciences in 2020. He is currently a Research Assistant with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include network security and data security.

Aimin Yu received the B.S. degree from Nankai University and the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, in 2011. He is currently a Professor with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include system security and trust computing.



Lijun Cai received the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, in 2016. She is currently an Associate Professor with the Institute of Information Engineering, Chinese Academy of Sciences. Her research interests include cyber security and network monitoring and analysis.

Dan Meng received the B.S., M.S., and Ph.D. degrees from Harbin Institute of Technology, Harbin, Heilongjiang, China. He is currently the Director of the Institute of Information Engineering, Chinese Academy of Sciences, and the Dean of the School of Cyber Security, University of Chinese Academy of Science, Beijing, China. His research interests include high-performance computer architecture and cyber security.