ERW-Radar: An Adaptive Detection System against Evasive Ransomware by Contextual Behavior Detection and Fine-grained Content Analysis

Lingbo Zhao, Yuhui Zhang, Zhilu Wang, Fengkai Yuan*, Rui Hou*
Institute of Information Engineering, Chinese Academy of Sciences
{zhaolingbo, zhangyuhui, wangzhilu, yuanfengkai, hourui}@iie.ac.cn

***Abstract*—To evade existing antivirus software and detection systems, ransomware authors tend to obscure behavior differences with benign programs by imitating them or by weakening malicious behaviors during encryption. Existing defense solutions have limited effects on defending against evasive ransomware. Fortunately, through extensive observation, we find I/O behaviors of evasive ransomware exhibit a unique repetitiveness during encryption. This is rarely observed in benign programs. Besides, the χ^2 test and the probability distribution of byte streams can effectively distinguish encrypted files from benignly modified files. Inspired by these, we first propose ERW-Radar, a detection system, to detect evasive ransomware accurately and efficiently. We make three breakthroughs: 1) a contextual *Correlation* mechanism to detect malicious behaviors; 2) a fine-grained content *Analysis* mechanism to identify encrypted files; and 3) adaptive mechanisms to achieve a better trade-off between accuracy and efficiency. Experiments show that ERW-Radar detects evasive ransomware with an accuracy of 96.18% while maintaining a FPR of 5.36%. The average overhead of ERW-Radar is 5.09% in CPU utilization and 3.80% in memory utilization.**

I. Introduction

Crypto Ransomware is a type of malware that encrypts the data of infected hosts. It demands ransom from victims in exchange for decryption keys [1]. Over the past decade, it has inflicted economic losses exceeding 20 billion dollars across various sectors of society, such as governmental agencies [2], healthcare institutions [3] and educational establishments [4]. Therefore, considerable efforts are devoted to detecting ransomware accurately to minimize economic losses [5], [6], [7], [8], [9], [10], [11], [12]. Existing defense mechanisms mainly focus on detecting anomalous I/O behaviors of ransomware during encryption [13], [14], [15], [16], [17], [18]. Their typical approaches include i) detecting anomalous behavior features, such as the high frequency of I/O requests associated with read, write, and other file access operations during encryption, ii) detecting specific behavior patterns based on predefined rules, such as a read-encrypt-overwrite behavior pattern generated by a specific sequence of operations during encryption, and iii) calculating and analyzing entropy values of written buffers based on the insight that encryption algorithms result in encrypted content exhibiting high entropy values.

However, with the evolution of attack strategies, existing defense mechanisms have limited effects on defending against ransomware efficiently. Their limitations are manifested in three aspects. a) They are vulnerable to evasive attackers, who adjust I/O strategy to make operations less intensive or regular [19], [20], [21]. Most of them rely heavily on predefined features or specific patterns. However, attackers can easily bypass them by keeping the operation frequency within detection thresholds or avoiding operations exhibiting specific patterns. For example, they split short-duration encryption tasks and run these sub-tasks intermittently. b) They are powerless to uncover hidden encryption behaviors of attackers who achieve encryption goal by imitating benign programs [13], [22]. They assume that ransomware behaves very differently from benign programs due to frequently encrypting files and erasing original content. However, the discrepancies between the two can be obscured by encrypting files following benign behavior patterns. c) They suffer from potential attacks targeting the high entropy values of encrypted content [23], [24], [25]. Attackers can use partial encryption or pad low-entropy data to reduce entropy values, thereby bypassing entropy-based detection. Besides, some benignly modified files (compressed files) also exhibit high entropy values. This compromises the ability of detectors to distinguish malicious and benign writes.

In this paper, we aim to figure out how ransomware successfully evades detection and fill the gap in defending against evasive ransomware with advanced mechanisms. To achieve this goal, we investigate and analyze the I/O behaviors and encrypted files of numerous evasive samples both in the literature and the wild. We first observe that ① *the I/O behaviors of evasive ransomware exhibit a unique repetitiveness characteristic over the long term, even though they are dispersed or wrapped as benign behaviors.* The reason is that attackers evade detection by slowing down the encryption speed or adjusting the encryption operations under the guidance of their customized evasive strategies. It means that fewer encryption tasks are completed by a process in the short term. Consequently, attackers have to repetitively use evasive strategies to encrypt numerous files. This ultimately results in multiple sets of similar I/O behavior segments being exhibited in the long term. We also observe that ② *the* χ^2

*Corresponding authors: Rui Hou and Fengkai Yuan.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230349
www.ndss-symposium.org

results of byte values for encrypted files and benignly modified files show significant differences. Additionally, the probability distribution of bytes is more uniform in encrypted files, yet it slightly fluctuates in files padded with low-entropy data, while it exhibits numerous peaks in benignly modified files. This indicates the χ^2 test and the probability distribution can serve as reliable indicators of encrypted content.

Based on these observations, we make three main breakthroughs: First, motivated by observation ①, we propose a *Correlation* mechanism to detect I/O behaviors of evasive ransomware. It analyzes behavioral correlations by combining the operation details in local behaviors and contextual information in global behaviors. Our insight is that if local behaviors and global behaviors are highly correlated, this correlation can reflect that similar behavior segments have repeatedly appeared over the past period. Therefore, by targeting the unique behavioral repetitiveness and identifying it through correlation analysis, we can detect evasive ransomware accurately. In contrast, previous behavior-based detection mechanisms regard predefined behavioral features/patterns as the main criteria for decision-making. They fail to detect evasive attackers who weaken features or disrupt specific patterns by distributing malicious operations. Additionally, our detection mechanism exhibits greater robustness in preventing adversaries who attempt to tamper with the correlation, as its complex decision-making process considers both local and global behavioral information rather than relying solely on individual data points.

Second, inspired by observation ②, we propose a fine-grained content analysis mechanism to detect whether modified files are encrypted. It considers the overall randomness and the detailed probability distribution of byte streams to classify modified files. Compared to previous detection which is easily bypassed by low-entropy attacks, these indicators enable a fine-grained analysis of whether data follows a random distribution, so they can distinguish encrypted files from benignly modified files more accurately. In addition, we believe content analysis is necessary as it can resist potential threats, since all ransomware encrypts files and even evasive one is no exception.

Finally, we propose adaptive mechanisms to achieve a better trade-off between detection accuracy and efficiency. First, we adaptively adjust the behavior detection window size based on the above two detection results. Compared to existing detection systems with fixed-size detection windows, this approach enables us to capture as many behaviors as possible to provide richer contextual information for accurate decision-making, especially when attackers weaken encryption behaviors. Additionally, it avoids excessive decision delays caused by analyzing overly long behavior sequences. Second, we assess the I/O busyness in real-time and adaptively adjust the start time of content analysis to ensure it runs during idle I/O cycles. With adaptive mechanisms, we can seamlessly integrate the two detection mechanisms into a hybrid detection system, thereby, reducing FPs/FNs, enhancing detection accuracy, and reducing detection overhead.

Overall, we develop ERW-Radar, a hybrid detection system, to detect evasive ransomware accurately and efficiently. To implement it, we deploy a filesystem driver to collect I/O behaviors in the kernel and perform behavior detection and content analysis in user space. To evaluate ERW-Radar, we collect 910 ransomware samples (132 evasive samples and 778 traditional samples) ranging from 2018 to 2024, and run them in VMwares equipped with Windows 10. Results show that ERW-Radar can successfully detect evasive ransomware with an accuracy of 96.18% and traditional ransomware with an accuracy of 96.12%. We also evaluate the overhead of ERW-Radar. Results show that the average CPU and memory utilization are 5.09% and 3.80%, respectively. Additionally, we evaluate the resilience of ERW-Radar against adversaries who attempt to overflow the detection system and tamper with the behavioral correlation or the χ^2 test.

In summary, this paper makes the following contributions:

- We explore the limitations of existing detection systems in defending against evasive attacks. Through extensive observation of evasive attacks, we find the unique behavioral repetitiveness exhibited over the long term is an ideal candidate for characterizing evasive ransomware. Motivated by this, we propose a *Correlation* mechanism, which is a context-based approach, to detect ransomware by analyzing the correlation of I/O behaviors.
- We also observe that the χ^2 test results and the probability distribution of bytes are more reliable indicators of encrypted files. Inspired by this, we develop a fine-grained content analysis mechanism to precisely estimate whether files are encrypted by ransomware.
- We propose adaptive mechanisms that dynamically adjust the detection window and initiate content analysis. They help seamlessly integrate the two detection mechanisms into a hybrid detection system, achieving a better trade-off between detection accuracy and efficiency. Results show ERW-Radar detects evasive ransomware with an accuracy of 96.18%. The overhead of running ERW-Radar is 5.09% and 3.80% in CPU and memory utilization.

II. Understanding of evasive ransomware

In this section, we detail how evasive ransomware is designed and how prevalent the evasive techniques are in real ransomware. Then we highlight the necessity of enhancing existing solutions in defending against evasive ransomware.

Currently, the most widely used approach to detect ransomware is monitoring I/O behaviors and employing predefined features or patterns to discriminate malicious behaviors [5], [7], [6], [8]. Because traditional ransomware performs numerous operations to encrypt files and erase original contents continuously and rapidly. It often writes high-entropy content to files due to the nature of encryption algorithms. To evade detection, attackers expend considerable efforts in encrypting covertly. They focus on obscuring discrepancies with benign programs by imitating them or weakening malicious behavioral features. In their designs, a series of instruction lists, which define the types, orders, and time intervals of operations, are needed to guide the encryption. They are

ingeniously pre-designed based on various evasive techniques and can be applied to encrypting files of different types and sizes. Typically, this is completed during the deployment phase. During an attack, evasive ransomware spawns processes according to the pre-designed instruction lists. The processes then follow their instruction lists to access files and write encrypted data.

So far, we've found hundreds of ransomware samples that report successfully evading detection. They come not only from novel evasive families [26], [27], [28], [29], [30], [31] but also from well-known traditional families [32], [33], [34], [35], [36] and academic papers [13], [19], [20], [37], [38], [39]. This indicates evasive techniques are becoming more and more prevalent both in the literature and in the wild. Currently, there are four mainstream evasive techniques widely used by ransomware authors. They are:

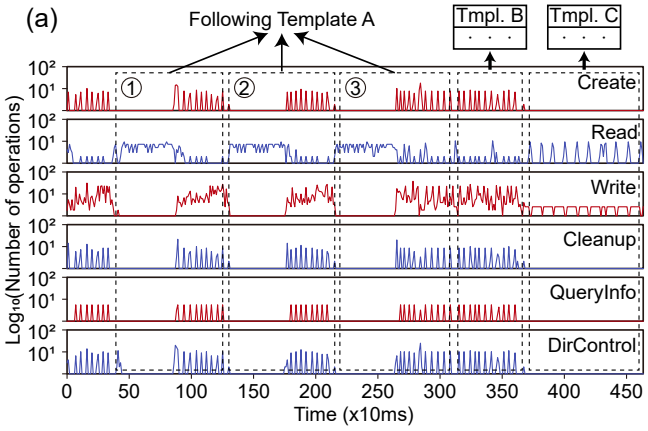
Splitting: Ransomware authors distribute malicious payloads (encryption operations) across multiple processes. In this way, none of the sub-payloads contain recognizable malicious behavior features (high frequency and specific patterns) known to single-process-based ransomware detectors, yet those sub-payloads as a whole can still fulfill the original malicious functionality [37], [39], [38], [19]. Such a splitting technique is widely used in real ransomware such as Conti [35], [36] and Ryuk [32]. They perform encryption in one process and auxiliary tasks (e.g., collecting system information) in other processes. Authors of a splitting attack [19] demonstrate that the accuracy of ShieldFS [5] decreases after a single split, going from 98.6% down to 65.5% on a two-process ransomware. And by dividing 18 processes into 3 functional groups: $\{directory\ list\}$,$\{read, rename\}$ and $\{write\}$, ransomware is able to completely evade RWGuard [7].

Intermittent: Attackers introduce idle operations, i.e., sleeping after encrypting for a certain period. By repeating this work-sleep pattern during encryption, they reduce the operation frequency within a period, thereby circumventing detection. After developers of LockFile [40] adopted intermittent encryption in 2019, novel families such as BlackCat [26], BlackBasta [27] and PLAY [29] have successfully followed it.

Imitating: Attackers imitate benign programs to encrypt files. They first extract behavioral templates from benign programs. Then, they spawn processes according to the templates, and the processes follow their own instruction lists to access files and write encrypted data. For example, DoppelPaymer [30], [31] and ANIMAGUS [13] evade detection by imitating file operations and network traffic patterns of benign programs during encryption. This eliminates the differences between encryption and benign behaviors. With this technique, ransomware can achieve its encryption goal while behaving like a benign program.

Low-entropy [23]: Attackers implement distributed partial encryption. They encrypt partial file content and distribute it throughout the entire file to maximize file corruption while incurring a smaller local entropy increase. For example, attackers[23] reduce entropy values from 7.86 to 6.96, thereby bypassing ShieldFS [5] with a detection threshold of 7.82.

We investigate the effectiveness of some evasive samples against state-of-the-art defense mechanisms, including two mainstream Antivirus (AV) software, MalwareBytes [12] and Windows Defender [11], as well as two representative detection systems in the literature, RWGuard [7] and ShieldFS [5]. We summarize that samples using splitting or imitating techniques evade all the defense mechanisms. A few intermittent attacks are detected by AV software due to their binary signatures. Additionally, by using low-entropy techniques, attackers can fully evade the entropy indicators of RWGuard and ShieldFS. This indicates both existing academic detection systems and AV software fail to defend against evasive ransomware. So how to detect it effectively becomes an urgent problem. We hope our preliminary study will stimulate further discussion and research to enhance current ransomware detectors, thereby addressing this potential threat before it becomes a major security problem.



Template A

Time	Proc	Operation	Path	Entropy	R/W Size
T_0+t_i	P_i...	QueryInfo	Dir_1...	/	/
T_0+t_j	P_j...	Read	F_1...	/	4096B
T_0+t_k	P_k...	DirControl	F_2...	/	/
T_0+t_l	P_l...	Create	F_3...	/	/
T_0+t_m	P_m...	Write	F_4...	6.12	1024B
T_0+t_n	P_n...	CleanUp	F_5...	/	/
...	...	...	...	...	...

Fig. 1: I/O behaviors and an instruction list of an imitating attack, namely $ANI^{MSOffice}$.

III. Observation & Motivation

The evasive techniques have been well studied and summarized in §II. They are implemented through a set of instruction lists. We exemplify one of them employed by imitating attacks in Fig. 1 (b). We regard these instruction lists as building blocks of evasive ransomware because they can be combined to encrypt files and evade detection. For example, attackers use a splitting instruction list to set four functional groups: $\{directory\ list\}$, $\{read\}$, $\{write\}$, and $\{rename\}$, then apply the intermittent technique to introducing sleep into operations. To explore the details of evasive ransomware, we

collect 132 samples from public repositories [41], [42], [43] and run them for ten minutes to collect their I/O behaviors. To ensure that each sample encrypts effectively, we do not collect its I/O behaviors until at least one file is observed being encrypted. Then we analyze the frequency of various I/O operations over time and show a representative imitating attack $ANI^{MSOffice}$ (encrypting files by imitating Microsoft Office) in Fig. 1 (a). This sample poses a great challenge in accurately distinguishing it from benign programs because they both perform intensive I/O operations.

We can intuitively observe a series of similar behavior segments (marked with boxes ①②③) in Fig. 1 (a). Through analyzing their behavioral details, we find processes within these boxes follow the same instruction list, which is shown in Template A in Fig. 1 (b). This demonstrates the sample executes a series of similar operations over a period of time. We also notice other different templates such as B and C in this sample, with each one repetitively employed a various number of times. The discrepancy arises from the differing types and sizes of files encrypted with templates A, B and C. Besides, repetitive behavior segments are also observed in other samples of imitating attacks, splitting attacks and intermittent attacks. We show them in Fig. 8 and Fig. 9 in Appendix A. Therefore, we conclude that *evasive ransomware repetitively executes similar operations based on various templates, which results in its I/O behaviors exhibiting a unique repetitiveness characteristic over the long term.*

We then analyze the reason behind this and demonstrate that *the above behavioral repetitiveness is inevitable*. As mentioned in §II, evasive attackers rely on pre-designed instruction lists to reduce encryption speed or operation types. However, this would result in fewer encryption tasks being completed in the short term. Meanwhile, the available lists are limited during encryption, because they need to be cleverly designed to evade detection and are usually packaged as a library before malicious payloads are loaded onto victim systems. Therefore, to encrypt numerous file blocks, attackers have to repetitively utilize various instruction lists, which inevitably leads to behaviors exhibiting a unique repetitiveness. For example, in imitating attacks, considering that benign programs rarely access files intensively and continuously, I/O operations are relatively limited in instruction lists, so attackers have to repetitively utilize them to encrypt files. In contrast, such a behavior regularity is rarely observed in widely-used benign programs (Chrome and Firefox). Because they are manipulated by users who rarely perform repetitive tasks continuously and intensively, e.g., users may utilize Chrome to perform actions like browsing web pages, downloading files and so on. A few benign programs with repetitive work patterns (WinRAR) can be excluded easily by combining content analysis (§V-D).

We further collect 1,000 files of various types (jpg/png, pdf, etc.) from well-known benchmarks[44], [45] to analyze encrypted files. We use AES 256, a representative symmetric encryption algorithm, to encrypt 50% of files and make benign modifications to the others, e.g., compressing files into zip formats. Then we extract segments of 512 bytes, 1024 bytes and 2048 bytes in size from these files. For each segment size, we randomly select 10,000 encrypted file segments and 10,000 other file segments. An obvious fact is that the encryption algorithm results in byte streams exhibiting high randomness. Therefore, we utilize χ^2 to measure the deviation between the distribution of byte streams in a file and a uniform discrete distribution. We choose absolute values (Abs.) to represent the χ^2 results of the above segments and normalize them. Results show that the differences in normalized Abs. between encrypted and other segments of 512 bytes fall between 105.4 and 150.6; of 1024 bytes between 108.24 and 153.76, and of 2048 bytes between 112.43 and 157.57. This indicates *the χ^2 test results can distinguish between encrypted files and benignly modified files to some extent*. On the other hand, since χ^2 test "compresses" the distribution of bytes into a single scalar value, it loses detailed information of the overall distribution shape, so if we pad some structured data (low-randomness data) to encrypted files, the differences in χ^2 results can be reduced. Therefore, to capture its fine-grained distribution regularity, we further analyze it as a whole and find that *the probability distribution of bytes is more uniform in encrypted files, yet it slightly fluctuates in files padded with low-entropy data, while it exhibits numerous peaks in benignly modified files*. This indicates that combining the probability distribution of bytes with the χ^2 test helps distinguish encrypted files more accurately, especially when the differences in the χ^2 test are not significant.

In summary, we observe that evasive ransomware exhibits inevitable characteristics both in long-term I/O behaviors and in encrypted contents. These have not been found in previous ransomware defense solutions, thus providing us with an opportunity to defend against evasive ransomware.

IV. Threat Model

In this paper, we focus on the encryption phase of ransomware because it is when actual malicious actions take place and when permanent file losses occur. Therefore, we mainly study defense solutions against ransomware that evades detection during encryption. To ensure successful attacks, ransomware needs to be developed following three conditions: First, ransomware evades detection systems by adopting mainstream evasive techniques in §II rather than simplifying encryption operations, such as directly deleting files. Second, it encrypts the majority of file content to ensure that files cannot be read or recovered by victims, rather than using non-encryption content hiding techniques, such as setting passwords for files. Finally, it does not encrypt at extremely slow speeds, e.g., a 1 byte/h speed, because this would be considered harmless to systems.

A potential threat to ERW-Radar is that adversaries may escalate privilege [46] and terminate detection in the user space. This is an ongoing security problem. Since we rely on it for ransomware defense, we need to ensure that it cannot be maliciously shut down. Note that to address this problem, several promising approaches have been proposed [47], [48], [49]. They deny operations that attempt to disable the detection

system by performing security authentication, access control, integrity check, process isolation, memory protection and so on. We detail them in §VII. Taking Active Security Processor (ASP) [47] as an example, we can entrust the management of ERW-Radar to the ASP, which has the highest privilege (Super Root) of the system. In this way, the detection processes are monitored and managed in a secure and isolated environment which can prevent privileged attacks from terminating them. Thus, by integrating such an approach into our system, we ensure ERW-Radar's resilience against termination.

V. Design of ERW-Radar

A. Design Overview

In this paper, we aim to detect evasive ransomware accurately. Traditional solutions rely heavily on predefined behavior features and patterns or file entropy, thus, they are easily bypassed by evasive ransomware which breaks original behavior patterns or reduces the entropy of encrypted content. To tackle it, we design ERW-Radar, an adaptive detection system against evasive ransomware. As Fig. 2 shows, it utilizes *I/O Monitor* to extract behavioral information from I/O Requests, *Behavior Detector* to detect I/O behaviors in real-time and *Content Analyzer* to analyze modified files at idle time. The insight behind our design is that the unique behavioral repetitiveness is an ideal candidate for identifying evasive ransomware. The χ^2 tests and the probability distribution of bytes values are also reliable indicators of encrypted files.

The workflow is as follows: When a process accesses the filesystem, I/O requests are generated and encapsulated as IRPs by the *I/O manager*. Subsequently, they are forwarded by *Filesystem Driver* to *I/O Monitor*. *I/O Monitor* parses IRPs to extract behavioral information and caches it in a queue located in kernel space. The behavioral information is periodically sent to *Behavior Detector* and *Content Analyzer*. After receiving it, ② *Preprocessor* first saves it as behavior logs, then embeds it into a behavior sequence and sends it to ① *Detector*. An overly long sequence needs to be compressed in ③ *Compressor*. Finally, ① *Detector* encodes the behavior sequence to obtain contextual information and analyzes it by combining the latest behavior sequence and the whole sequence. Meanwhile, ⑤ *Timer* continuously evaluates the I/O busyness and triggers ④ *Analyzer* to analyze recently modified files at idle I/O cycles.

One solution to extract behavioral information is to parse IRPs. However, there are over 30 types of filesystem-related IRPs and the frequency of some IRPs often reaches up to 3,000/s. This indicates that parsing all IRPs incurs a huge overhead. Furthermore, since behavioral information is extracted in the kernel space, frequently transmitting it to user space results in excessive context switching, heavily burdening the system. Thus, We seek a lightweight and customized solution that reduces the parsed IRPs to ensure extraction efficiency while still providing the necessary ransomware-related behavioral information. Additionally, we design a queue to temporarily cache the information and send it periodically.

To achieve accurate behavior detection, there are two major challenges to address. First, how we accurately detect the behavioral repetitiveness, which may be disrupted and lack periodicity? Since other programs in the system, especially those with heavy tasks may compete for system sources (e.g., CPU and I/O), I/O behaviors of ransomware may be disrupted. This leads to repetitive behavior segments not always being entirely consistent. Thus it's not feasible to detect repetitive behavioral segments by pre-defining their patterns or features, which are widely used by existing solutions [7], [8]. Another approach to detecting behavioral repetitiveness is determining whether I/O operations exhibit periodicity. However, given that time intervals are not always identical among various behavior segments, behaviors of evasive ransomware do not exhibit precise periodicity. This exceeds the analysis capabilities of periodic functions [50], [51]. To address this challenge, we propose a *Correlation* mechanism, a context-based detection approach, to identify repetitiveness. It breaks down the latest behavior sequence into progressive sub-sequences and conducts sequence-wise analysis with historical sequences.

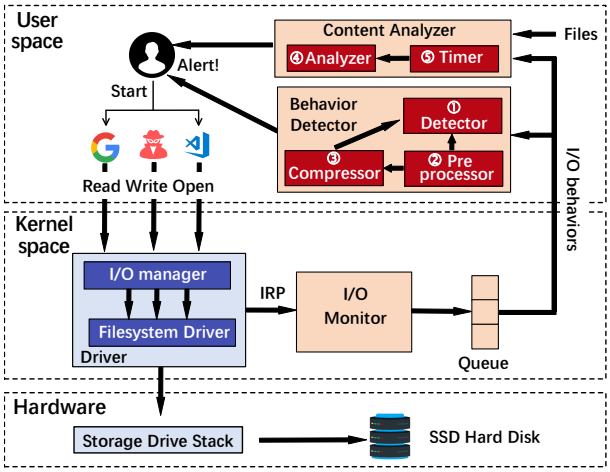


Fig. 2: Overview of ERW-Radar. The orange modules represent components designed to detect evasive ransomware; the blue modules represent existing components in the system.

Second, how to determine a feasible size for the detection window (the detected behavior sequence) to detect repetitiveness? A short detection window fails to provide enough contextual information for accurate decision-making, especially when attackers weaken encryption behaviors to evade detection. However, analyzing long sequences incurs a high computational latency. This causes more file loss. To precisely identify ransomware without causing too much analysis latency, we propose a dynamic detection window, which adaptively adjusts the window size based on recent detection results.

Existing content-based detection targets the entropy of write buffers of IRPs. It is typically performed in the kernel. However, due to limited computing resources and storage space, fine-grained content analysis in the kernel faces two challenges. First, conducting the χ^2 test and probability distribution evaluation for each write buffer requires substantial computing resources, especially for I/O-intensive programs

that generate numerous IRPs. This leads to excessive overload of the kernel, impacting system stability and responsiveness. Second, a larger content size allows for more accurate differentiation of encrypted files. But it's unfeasible to store excessive content in limited kernel space. To tackle these, we conduct content analysis in user space. We also propose a simple but effective mechanism that assesses the I/O busyness and adaptively triggers content analysis at idle I/O cycles.

Next, we present the implementation of behavioral information extraction, behavior detection and content analysis in §V-B, §V-C and §V-D, respectively.

B. Lightweight and Customized Information Extraction

The behavioral information extraction function of ERW-Radar is located in *I/O Monitor*, which is built on a standard kernel-based framework, Windows Filesystem Driver [52]. When users access the filesystem, their I/O requests will be packed as I/O request packets (IRPs) in *I/O manager* and forwarded to *I/O Monitor*. Then, by setting callbacks on IRPs, we can parse their attributes to extract the detailed behavioral information of these operations. Due to the asynchronous nature of the callback mechanism in handling I/O requests, it allows the CPU to handle other tasks without waiting for I/O operations to complete. This makes it widely used for extracting I/O information [5], [8], [13].

As mentioned in §V-A, parsing all IRPs and frequently transmitting the extracted behavioral information result in significant overhead. To tackle it, we lightweight and customize the *I/O monitor* with two techniques: (1) We only parse IRPs most relevant to ransomware behaviors to reduce the parsing overhead. First, we let most IRPs pass through directly except for file-related operations. Then we gradually filter out those IRPs with low frequency due to their negligible impact on characterizing ransomware behaviors, until we find a balance between performance and overhead (see §VI-B). Ultimately, only eight IRP types are retained, which can be considered specific in characterizing ransomware (see Appendix B). Afterward, we extract behavioral information of these IRPs from the FILE_OBJECT structure, an open instance of a file [53] containing the file-related information. We extract *timestamp, processID, type of IRP, full path of file/directory, and size of write/read buffer* from it, and employ them to construct I/O behavior logs in user space.

(2) We cache the behavioral information in the queue to reduce transmission overhead. Here are two problems that need to be solved. One is how to transmit the cached information efficiently. An asynchronous solution is transmitting data upon specific events (e.g. when the queue is full). It reduces overhead during idle periods but leads to more file loss because behaviors cannot be detected timely. Thus, we choose the polling mechanism with higher security. It enables behaviors to be sent and detected timely even during idle periods (§VI-B). To achieve this, the queue is set to transmit information periodically. Meanwhile, it checks the status to avoid empty transmissions. The other problem is how to make the queue long enough to be resilient against overflow. We conduct stress tests in scenarios where I/O-intensive applications are running to determine the maximum length. Results show that, with a sending interval of 16ms, a queue length value of 1000 ensures that legitimate I/O operations will not cause overflow. Such a setting is conservative because the actual length is usually half of the maximum during runtime. Besides, such a queue also ensures resilience against the DoS attack [54], which performs dummy I/O to overflow the queue (§VI-E).

C. Efficient and Accurate Behavior Detection

Previous observations in §III demonstrate that I/O behaviors exhibit distinctive repetitive patterns in the long term when the OS suffers an evasive attack. Accordingly, ERW-Radar makes full use of the contextual information of I/O behaviors to implement efficient and accurate evasive ransomware detection.

1) Dynamic Detection Windows: Given that the unique repetitive pattern of I/O behaviors is exhibited after a sufficient accumulation of I/O behaviors, to identify it accurately, we must expand the detection window to capture a long period of I/O behaviors for enough behavioral information. However, longer detection windows typically incur higher detection overhead. For widely-used models such as Transformer [55], its time complexity grows exponentially with the input size, limiting its ability to analyze a large number of I/O behaviors quickly for real-time detection.

Thus, we seek a balanced solution with acceptable costs and strong security guarantees. To this end, we propose an adaptive mechanism and embed it into *Preprocessor* to enable a dynamic detection window. It adjusts the window size according to recent results from the *Detector* and *Analyzer*. The insight behind our design is that: ① If both behavior detection and content analysis prompt positive results consistently, it means the current window size is sufficient to distinguish between malicious and benign behaviors effectively. A shorter but more feasible size is probably needed to detect ransomware. ② If the two results remain inconsistent, it implies the current window size is too small, either making malicious behaviors hard to identify or resulting in too many false positives. Therefore, expanding the detection window might capture more contextual information to help identify ransomware accurately.

A concerning case is that to capture subtle malicious behaviors, the detection window will be too large, affecting the detection efficiency. One solution to tackle this is compressing excessive behavioral information. To this end, we design *Compressor* to assist *Preprocessor* in implementing the dynamic detection window. It is built on a decoder unit of the Transformer [55]. It can remove unimportant information and retain key information by utilizing the cross-attention mechanism. This is a practical solution for two reasons: First, benefiting from the cross-attention mechanism, the decoder unit can transform an $m \times C$ input sequence into an $n \times C$ output sequence in $O(n^2C + nmC)$ time complexity, while retaining key behavioral information. If $n \ll m$, the time complexity is linear to m, making it an ideal tool for compressing large amounts of behavioral information [56], [57]. Second, the decoder unit is lightweight; introducing it to pre-process the

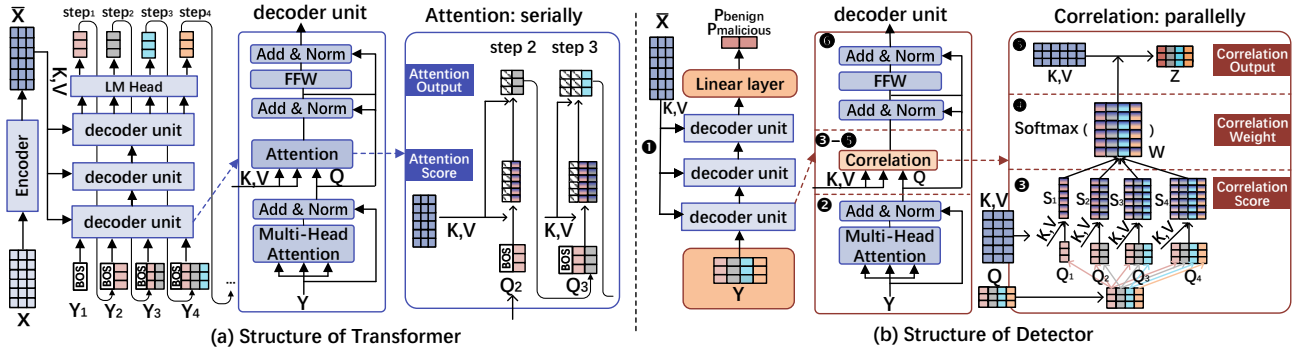


Fig. 3: Structure comparison of Detector. Input X represents the historical behavior sequence; Input Y represents the latest behavior sequence; $\overline{\mathrm{X}}$ represents the encoded behavior sequence. The orange modules represent modifications made to Transformer; the blue modules represent existing components in Transformer.

input data does not make the system cumbersome and difficult to maintain. The detailed implementation of *Compressor* is provided in Appendix C.

Concretely, the dynamic detection window is implemented as follows: The *Preprocessor* extracts *timestamp, processID, type of IRP and size of write/read buffer* from the behavior log and encodes them as behavior sequences. The sequence length t is adjusted as follows in the above two cases, ① to obtain a smaller t, it searches for the smallest length t' from 0 to t, at which the *Detector* can still detect malicious behavior sequences; ② otherwise, it increases t until results of *Detector* and *Analyzer* are consistent. The behavior sequence will be compressed if t exceeds 512. The upper limit of *Compressor* is conservatively set to 2048, which is derived from large amounts of real ransomware detection.

2) Behavior Detection Mechanism: To identify the repetitive behavior segments, a straightforward solution is predefining their specific patterns or features. However, as mentioned in §V-A, these behavior segments may be disrupted by other programs, causing them not to match the patterns or features, thus affecting the detection robustness in different environments. Another commonly utilized solution that detects the periodicity of operations is also not feasible due to the lack of precise periodicity. Therefore, we propose a *Correlation* mechanism, which detects behavioral repetitiveness by analyzing the **contextual correlation** between the latest and historical behaviors. This is derived from the intuition that behaviors observed recently provide precise information about the ongoing operations, while behaviors over an extended period offer contextual references for recent operations. If they are highly correlated, this correlation can reflect that similar behavior segments have repeatedly appeared in the past period.

We track global file access behaviors originating from multiple processes. The rationale is that analyzing the correlations between historical and recent behaviors based on a single process allows us to detect intermittent or traditional attacks, where a process may repeat certain I/O operations. Analyzing the correlations between different processes enables us to identify splitting attacks or imitating attacks, where different groups of processes may exhibit repetitive behavioral patterns. Therefore, we first break down the latest behavior sequence into a series of sub-sequences, which include behaviors from both single processes and multiple process groups. Then, we conduct a sequence-wise comparison between sub-sequences and the historical behavior sequence. After that, each sub-sequence is assigned a correlation score. A high score indicates a suspicious sub-sequence. We then make decisions by aggregating these scores. If the aggregated results show a high correlation, it indicates that the majority of recent operations have been repetitively occurring in the past period, i.e., behaviors exhibit repetitiveness in the past period.

Since our detection mechanism targets the inevitable behavioral repetitiveness rather than the pre-defined behavioral patterns or features, it maintains detection effectiveness even when attackers weaken, confuse or distribute their encryption behaviors. Additionally, it exhibits greater robustness in preventing adversaries who attempt to tamper with the correlation. This arises from its contextual and dynamic decision-making process, which considers both local and global behavior information rather than relying solely on individual data points.

Considering that the encoder-decoder structure of Transformer [55] is naturally suited to capture the relationships between two behavior sequences, we can maximize the use of its existing structures and avoid introducing excessive components that would make the model "fat" and difficult to maintain. Therefore, we implement the *Correlation* mechanism by embedding it into the decoder. To achieve this, we make the following modifications to the original structure: First, as Fig. 3 (a) shows, one of the inputs of the decoder, Y, is from the outputs at previous time steps. We set it to be the latest behavior sequence to analyze the correlation between the latest and historical behavior sequences. Second, we replace the Attention mechanism with the *Correlation* mechanism, as shown in Fig. 3 (b). Third, we add a Linear Layer to classify the correlation results. Given that there is no dependency between sub-sequences in the *Correlation* mechanism, they can be analyzed **in parallel**, thereby greatly improving analysis efficiency. Instead, outputs are generated serially in Fig. 3 (a) because the outputs generated in previous time steps need to be concatenated to form the input of the

current time step, resulting in an efficiency bottleneck.

Specifically, behavior detection includes encoding and decoding phases. During encoding, ① the historical behavior sequence X is encoded as contextual information $\bar{X}$ and sent to decoder units. This process is similar to that in the Transformer. We take the first decoder unit as an example to illustrate the decoding phase. It takes $\bar{X}$ (the contextual information) and Y (the latest behavior sequence) as inputs. During decoding, ② Y is first mapped to Q, a high-dimensional space, via the linear transformation matrix W^Q (obtained through model training). This aims to capture the internal relationships within the latest I/O behaviors, because the multi-head attention mechanism affords diverse representations for behaviors, with each head focusing on distinct behavior features in its dimensional subspace. Similarly, $\bar{X}$ is mapped to K and V via transformation matrices W^K and W^V. Then, ③ Q is broken down into a series of sub-sequences Q_i, each calculated with K **in parallel** to obtain the Correlation Score, S_i, represented as $S_i = \frac{Q_i K^T}{\sqrt{d_k}}$, $\sqrt{d_k}$ is the dimension of Q and K. Then, ④ weightedly averaging S_i to obtain S, represented as $S = \sum_{i=1}^{n} \alpha_i S_i$, where α_i are the weight coefficients, then getting Correlation Weight W_{ij} by $W_{ij} = \frac{\exp(S_{ij})}{\sum_k \exp(S_{ik})}$. After that, ⑤ obtaining the Correlation Output, Z, by $Z = W \cdot V$. Finally, ⑥ normalizing and applying a feedforward network to Z. The output of the last decoder unit is processed by a linear layer to obtain the detection result, i.e., the probability of being benign or malicious.

D. Adaptive and Fine-grained Content Analysis

As mentioned in §II, existing entropy-based detection is vulnerable to low-entropy attacks. Our observation in §III shows that the χ^2 tests and the probability distribution of byte values can effectively distinguish encrypted files from benignly modified ones. Accordingly, ERW-Radar conducts content analysis based on them to detect these attacks.

1) Adaptive Adjusting Start Time: As illustrated in §V-A, analyzing numerous write buffers of IRPs in kernel space requires large amounts of computation and storage, which overburdens the kernel and affects its stability. Therefore, we perform content analysis in userspace. This mitigates the kernel's burden while allowing us to conduct a more flexible and fine-grained analysis. Further, to lessen I/O traffic in a busy system, we can trigger content analysis at idle I/O cycles. Thus, we design *Timer* that adaptively adjusts the start time according to the I/O busyness of the system.

Usually, sustained high intervals between I/O requests imply the arrival of an appropriate analysis time. To assess the I/O busyness, we define *I/O Trend* as $\frac{x_t+(1-\beta)x_{t-1}+(1-\beta)^2 x_{t-2}+...+(1-\beta)^n x_{t-n}}{1+(1-\beta)+(1-\beta)^2+...+(1-\beta)^n}$, where n, the number of historical I/O request intervals, needs to be considered; β is the exponential smoothing factor; $\mathrm{x_i}$, $\mathrm{i = t-n, t-n+1, ..., t}$, is the interval between I/O requests at time i and $i-1$. Once *I/O Trend* is larger than a defined threshold (10s in ERW-Radar derived from statistical analysis), content analysis is started to check files. Meanwhile, *Timer* resets the time. If it doesn't work for a long time (10 min in ERW-Radar), it will be forcefully started to mitigate potential threats. Unlike other metrics used to assess I/O traffic such as I/O Throughput, *I/O Trend* does not simply reflect the average level over a period but measures the trend of intervals over the past period of time. Given that the latest values of time intervals carry greater weight relative to historical values, we assign a series of exponentially decreasing weights, $(1-\beta)^i$ to the values. Experiments (§VI-C) indicate that *I/O Trend* is more **sensitive** to reflecting the recent trend of I/O busyness.

2) Targets of Content Analysis: Ransomware either overwrites original files with encrypted data or writes encrypted data to other files and destroys the original files. Thus, only files newly created and recently modified are suspicious targets that need to be analyzed. We select segments of different sizes from these files as analysis targets distributively and randomly. This makes the detection process more unpredictable, preventing adversaries from pinpointing the analyzed file segments to tamper with the χ^2 test or probability distribution, thus ensuring the resilience of ERW-Radar. Additionally, it avoids the significant overhead of analyzing all files.

3) Fine-grained Content Analysis: The content analysis function lies in the *Analyzer* of ERW-Radar, which considers both the χ^2 test and the probability distribution of byte values. The χ^2 test is used to measure the deviation of an observed distribution from an expected distribution. It is defined as $\sum_{i=0}^{k} \frac{(N_i - E_i)^2}{E_i}$, where N_i and E_i are the actual number and expected number of samples assuming value i. In content analysis, the samples are the byte values of file content, thus i is the byte value, and k is 255. Typically, encryption algorithms result in encrypted content exhibiting high randomness. Therefore, we set the expected distribution of byte values to a discrete uniform distribution, i.e., $\forall i,\ E_i = \frac{L}{256}$, where L is the file content length being considered. Accordingly, the probability distribution of byte values is defined as $P(i) = N_i/L, i = 0, 1, ..., 255$. The features fed to *Analyzer* for training/classifying are derived from the χ^2 test results and a series of probability of byte values. Compared to entropy value (a rough estimation of encrypted content's disorderliness), these features are **fine-grained** indicators because they consider both the overall randomness of file content and the detailed probability distribution of its bytes. *Analyzer* is trained as a simple binary classifier to distinguish encrypted files from benignly modified ones. During runtime, it analyzes the above suspicious targets and classifies their feature values to determine whether they are encrypted. If more than half of the classifying results are malicious, ERW-Radar informs users that ransomware may have infected certain files.

E. Reduction of FP and FN cases of ERW-Radar

A common issue in detection systems is the frequent FP cases caused by individual positive results. To tackle this, we consider a delayed alert after receiving a certain number of notifications about malicious behaviors or encrypted files. Concretely, the *Detector* calculates the ratio of positive detection results over a period of time. Only when the ratio exceeds

a certain value does it alert that an attack has occurred. We define this value as a detection threshold, represented as α, and evaluate it in §VI-B.

Further, integrating *Detector* and *Analyzer* also contributes to reducing FP and FN cases. Particularly, *Detector* identifies and terminates malicious processes promptly, but if their behaviors are too subtle or too dispersed, they may be ignored. However, *Analyzer* can mitigate such potential attacks, but the fine-grained analysis limits its ability to analyze files timely. Therefore, it is feasible to conduct content analysis during idle time to complement the vulnerabilities of behavior detection. We demonstrate it in § VI-B. The following details how to reduce FP and FN cases by integrating them.

FP case: A common FP case is that some benign programs compress files into a certain format continuously, making their behavior segments highly similar. In this case, *Detector* frequently reports FPs, causing an extra system overhead. To mitigate the issue, we propose a solution that integrates the two results. Concretely, when *Detector* continuously alerts, if they are denied by users, *Detector* marks detected behaviors in the behavior log. But if they are ignored for a long time, *Analyzer* is forced to start to prevent potential file loss caused by an unattended system. When continuous alerts reoccur, ERW-Radar first queries whether the current behaviors match the marked behaviors in the log. If they match, it ignores the alerts; otherwise, it marks them. By doing so, ERW-Radar avoids frequent false alerts.

FN case: To avoid the potential FNs of *Detector*, we initiate the *Analyzer* if no behavioral alerts are triggered for a sustained period. In the case above malicious behaviors are too subtle or too dispersed to detect, *Analyzer* can tackle the potential threat by identifying the encrypted files.

VI. Experiments

A. Methodology

1) Experimental setup: The experiments are conducted on a desktop with an 8-core AMD Ryzen 7 CPU and 16 GB memory. Since ransomware encrypts user files for extortion, we build a user document directory consisting of various types of files, including JPGs, DOCs, XLSs, etc. To evaluate if ERW-Radar can detect ransomware in the real world, we collect 910 ransomware samples ranging from 2018 to 2024. They include 132 evasive ransomware samples and 778 traditional ransomware samples from MalwareBazaar [41], VirusShare [42], GitHub [43], [58] and Prototypes [19], [37], as shown in Tab. I. Following the best practices for malware experiments suggested by [59], each sample is tested in advance to ensure its function to encrypt files. Then, each sample is identified post-run. We let them run for minutes on virtual machines (VMs) equipped with Windows 10 and assigned 4 cores and 8 GB memory to evaluate the detection performance of ERW-Radar. Our evaluation is based on several metrics, including Accuracy, False Positive Rate, Recall, Precision and F1-Score. After a round of experiments, VMs are rolled back to a clean state to prevent any interference across executions.

TABLE I: Details of ransomware samples.

Evasive	**Ransomware**	**Samples**	**Traditional**	**Ransomware**	**Samples**
Family	Number	Rate	Family	Number	Rate
• $ANI^{MSOffice}$	10	1.10%	Revil	205	22.53%
• ANI^{WPS}			Cerber	201	22.09%
• ANI^{MSEdge}			Chaos	196	21.54%
• $ANI^{Firefox}$			Darkside	57	6.26%
• ANI^{Chrome}			Mespinoza	24	2.64%
• ANI^{WinRAR}			Mountlocker	19	2.09%
• ANI^{7zip}			Wannacry	19	2.09%
• ANI^{Golang}			Xorist	9	0.99%
• ANI^{Rustc}			HelloXD	9	0.99%
• ANI^{VS}			Virlock	7	0.77%
⋆ $Blackcat$	81	8.90%	Diavol	6	0.66%
⋆ $Blackbasta$	14	1.54%	Karma	5	0.55%
⋆ $Lockfile$	10	1.10%	Voidcrypt	5	0.55%
⋆ $Play$	5	0.55%	Badrabbit	5	0.55%
⋆ $Lockergoga$	5	0.55%	Zepplin	3	0.33%
⋄ $Conti$	3	0.33%	Other 8 families	8	0.88%
⋄ $Ryuk$	2	0.22%			
⋄ $SplittingProto.$	2	0.22%			
Tot. 18 families	132	14.51%	Tot. 23 families	778	85.49%

• refers to imitating attacks, e.g., $ANI^{Firefox}$ refers to a sample of the ANIMAGUS family, which encrypts files by imitating Firefox; ⋆ and ⋄ refer to intermittent and splitting attacks, respectively.

2) Models & Datasets: In behavior detection, we set the feedforward dimension of *Detector* as 64, the attention head as 16, the input sequence feature as 16 and the activation function as ReLU. We set search space for other hyper-parameters and select the optimal through the security and feasibility trade-off in §VI-B. We collect malicious I/O behaviors during ransomware samples running. To gather benign behaviors, we deploy *I/O Monitor* on 10 normally used host machines and collect their I/O behaviors for one week. 80% of I/O behaviors are used for training and 20% are reserved for testing. In content analysis, we design *Analyzer* based on a ransom forest model with 10 decision trees and a depth of 5. To build an unbiased dataset, we partition the file dataset in §III into 80% training and 20% test. To prevent over-fitting, we utilize a 10-fold cross-validation.

B. Evaluation of Behavior Detection of ERW-Radar

1) Effectiveness analysis: To evaluate the contribution of each component to the overall performance improvement, we compare ERW-Radar with four behavior detectors: (1) ERW-Fixed, which shields the *Compressor* and sets the detection window to a fixed size of 512. (2) ERW-Feat, which detects ransomware based on behavior features in RWGuard and Unveil [8], [9]. (3) ERW-Trans, which uses the original Transformer [55] with the attention mechanism. (4) ERW-ShieldFS, which replicates ShieldFS [5] with our dataset. Table II presents the detection accuracy, recall and FPR. Experiments show that ERW-Radar achieves high detection accuracy for both evasive and traditional ransomware, with accuracies of 96.18% and 96.12%, respectively. To measure the FPR of ERW-Radar, we deploy it on 10 normally used host machines and run it for a week. During this period, we receive 173 false alarts from *Detector*. The false positive behavior segments account for 5.36%. The undetected samples account for 3.76%. However, through content analysis, the number

TABLE II: Performance comparison between the *Detectors* of ERW-Radar and other behavior detection tools.

	Ransomware Families	ERW-Radar			ERW-Fixed			ERW-Feat			ERW-Trans			ERW-ShieldFS		
		Recall		Overall	Recall		Overall	Recall		Overall	Recall		Overall	Recall		Overall
Imitating	ANI^{MSEdge}	94.25%	Averaged detection recall: **95.23%**	Recall **96.24%** FPR **5.36%** Accuracy	94.03%	Averaged detection recall: 94.85%	Recall 94.98% FPR 6.79% Accuracy	69.88%	Averaged detection recall: 79.49%	Recall 88.82% FPR 7.95% Accuracy	88.35%	Averaged detection recall: 88.51%	Recall 91.10% FPR 8.79% Accuracy	37.38%	Averaged detection recall: 47.58%	Recall 73.85% FPR 8.92% Accuracy
	$ANI^{Firefox}$	91.01%			89.39%			64.65%			82.97%			27.21%		
	ANI^{Chrome}	94.21%			93.95%			77.77%			87.03%			42.34%		
	$ANI^{MSOffice}$	97.98%			96.57%			82.98%			90.03%			42.99%		
	ANI^{7zip}	93.79%			94.02%			80.20%			88.87%			50.82%		
	ANI^{WinRAR}	96.94%			96.03%			83.27%			89.79%			44.78%		
	ANI^{VS}	96.33%			95.93%			90.97%			89.96%			74.61%		
	ANI^{Rustc}	97.00%			95.79%			79.38%			88.53%			51.87%		
	ANI^{WPS}	98.33%			97.97%			86.28%			91.07%			56.23%		
Intermittent	Play	96.07%	Averaged detection recall: **97.00%**	**96.65%**	94.97%	Averaged detection recall: 95.59%	95.54%	97.12%	Averaged detection recall: 93.22%	88.36%	93.00%	Averaged detection recall: 92.21%	91.33%	89.23%	Averaged detection recall: 77.96%	74.49%
	Blackcat	96.84%			94.35%			89.03%			89.13%			70.98%		
	Blackbasta	97.94%			96.44%			96.99%			92.65%			90.85%		
	Lockfile	96.85%			95.33%			87.61%			93.67%			69.63%		
	Lockergoga	97.28%			96.88%			95.36%			92.58%			69.12%		
Splitting	Conti	96.16%	Avg. recall: **96.31%**		94.23%	Avg. recall: 95.36%		90.85%	Avg. recall: 90.76%		91.95%	Avg. recall: 91.80%		86.43%	Avg. recall: 80.31%	
	Ryuk	97.46%			96.75%			92.48%			93.21%			82.57%		
	SplittingProto.	95.30%			95.10%			88.96%			90.25%			71.94%		
*	Tradit. RW	96.12%			94.11%			91.79%			91.89%			89.56%		

of false alerts is reduced to 17, an almost negligible FPR, and the number of undetected samples is reduced to 0. This demonstrates that integrating *Detector* and *Analyzer* (See §V-E) is effective in reducing the FPs and FNs. Below we detail why the four detectors are not as effective as ERW-Radar.

ERW-Fixed: We evaluate the impact of the dynamic detection window on performance. Compared to ERW-Fixed, ERW-Radar enhances recall by 1.26% and decreases the FPR by 1.43%. It indicates when detection results of *Detector* and *Analyzer* remain consistent, shortening the detection window does not result in significant semantic loss or a decrease in accuracy. Besides, expanding it allows for capturing more malicious behaviors, thereby avoiding potential FNs and FPs. This confirms the necessity of using a dynamic window to adapt to varying malicious behaviors.

ERW-Trans: We evaluate the effectiveness of the *Correlation* mechanism in identifying behavioral repetitiveness. Compared to ERW-Trans, our *Detector* achieves an accuracy increase of 5.32%. This confirms our design in §V-C, i.e., analyzing the contextual correlation of behaviors enables more accurate detection of evasive ransomware. In contrast, although Transformer is also a contextual analysis framework, it only calculates relationships among scattered points to discover point-level dependency, but fails to understand subsequences as a whole. Thus, it's not suitable for analyzing the repetitiveness of behavior segments. Note that although using DL models (ERW-Trans) obtains better results than using ML models (ERW-ShieldFS), with recall increasing by 17.25%, it is still insufficient to fully mitigate the threat of evasive attacks.

ERW-Feat & ERW-ShieldFS: We compare ERW-Radar with three representative detection tools: ShieldFS [5], RWGuard [7] and Unveil [8]. We summarize their features used to detect malicious behaviors, including the number of traversed directories, the number of read/written/renamed files, the frequency of file operations, the number of accessed file types and the average entropy/length of written buffers. We extract these feature values from the behavior log to form the dataset. Since the authors of ShieldFS do not release its kernel driver nor its trained model, we train a decision tree model, ERW-ShieldFS, based on our dataset and regard it as an alternative to ShieldFS to facilitate our in-depth analysis. We notice that compared to ERW-ShieldFS, ERW-Radar exhibits a significant advantage in identifying both traditional and evasive ransomware, with recall increasing by 6.56% and 27.56%. Because ERW-ShieldFS treats the number of renamed files and traversed directories as essential criteria for decision-making. In terms of these features, evasive ransomware behaves similarly to benign programs. Therefore, the decision tree considers it as a benign program. As a result, ERW-ShieldFS can only distinguish traditional ransomware from benign programs but fails to detect evasive ransomware. In contrast, analyzing the behavioral correlation exhibits greater robustness.

Besides, we train ERW-Feat based on the above statistical features but adopt our contextual approach for decision-making rather than threshold/pattern-based judgment. Compared to ERW-Feat, ERW-Radar does improve the recall of evasive attacks by 8.36%. This improvement can be attributed to the repetitive characteristic. In contrast to features such as I/O frequency and patterns used by other detection tools, the repetitiveness characteristic used by ERW-Radar better distinguishes evasive ransomware from benign programs. We show the visualization comparison between the repetitiveness and these features in Fig. 11 in Appendix A.

2) Detection Cost of ERW-Radar: To evaluate the detection cost, first, we start ERW-Radar after the ransomware begins encrypting files and measure the time taken to detect it; then, to evaluate the detection delay, we measure the number of substantively encrypted files (see Note of Tab. III), encrypted bytes and IRP_MJ_WRITE requests generated during this period. We also conduct the same tests on another representative detector, ERW-ShieldFS. As Tab. III shows, the average detection time of traditional samples for ERW-Radar is 0.24s.

TABLE III: Detection time as well as the number of substantively encrypted files, encrypted bytes and IRP_MJ_WRITE requests generated during the detection of ERW-Radar and ERW-ShieldFS.

Evasive RW.	Time(ms)	IRPs(Num.)	Files(Num.)	Bytes(KB)	Traditional RW.	Time(ms)	IRPs(Num.)	Files(Num.)	Bytes(KB)
	ERW-Radar					ERW-Radar vs. ERW-ShieldFS			
$ANI^{MSOffice}$	383	5	1	4	Badrabbit	370 vs. 290	65 vs. 40	3 vs. 3	67 vs. 42
ANI^{WPS}	392	3	1	3	Darkside	27 vs. 135	1 vs. 27	1 vs. 2	1 vs. 27
ANI^{MSEdge}	690	10	3	9	Diavol	400 vs. 269	963 vs. 599	8 vs. 5	1009 vs. 628
$ANI^{Firefox}$	695	4	1	4	HelloXD	137 vs. 970	1 vs. 175	1 vs. 10	1 vs. 175
ANI^{Chrome}	625	4	1	6	Karma	132 vs. 367	10 vs. 54	1 vs. 3	9 vs. 49
ANI^{WinRAR}	855	10	3	9	Mespinoza	101 vs. 189	34 vs. 67	3 vs. 4	37 vs. 73
ANI^{7zip}	847	120	6	131	Mountlocker	57 vs. 340	18 vs. 79	2 vs. 4	20 vs. 88
ANI^{Golang}	820	45	4	40	Revil	135 vs. 450	5 vs. 98	1 vs. 4	3 vs. 59
ANI^{Rustc}	945	9	3	12	Cerber	104 vs. 378	1 vs. 47	1 vs. 3	1 vs. 47
ANI^{VS}	1918	6	1	12	Virlock	398 vs. 279	890 vs. 563	8 vs. 5	874 vs. 553
Lockergoga	1350	1290	8	1252	Voidcrypt	126 vs. 568	17 vs. 124	2 vs. 5	35 vs. 255
Blackcat [1]	410	17	1(+3)	27(+0.009)	Xorist	1600 vs. 1879	703 vs. 862	11 vs. 16	726 vs. 890
Blackbasta	752	789	6	680	Zeppelin	1536 vs. 2075	693 vs. 839	6 vs. 9	712 vs. 862
Play [1]	260	19	1(+2)	20(+0.006)	Wannacry	121 vs. 561	1 vs. 130	1 vs. 10	1 vs. 130
Lockfile	320	34	2	33	Chaos	79 vs. 138	4 vs. 167	1 vs. 3	2 vs. 84
Conti	475	990	7	1249					
Ryuk	325	895	7	800	others	154 vs. 223	21 vs. 79	2 vs. 3	34 vs. 128
SplittingProto	255	85	2	62					

Note: Since ERW-shieldFS cannot detect evasive ransomware, only its detection cost for traditional ransomware is evaluated. As for the number of substantively encrypted files, we examine each file and perform content similarity evaluations between the corrupted and original versions. For corrupted files with a similarity to the original file of less than 99%, we consider them to be substantively encrypted and count them. For corrupted files with a similarity of more than 99%, we confirm the location of the tampered bytes and exclude files that are essentially unencrypted (i.e., only structural bytes are damaged). The rest of these files are also considered as substantively encrypted and are included in the statistics.

During this period, an average of 149.00 IRP_MJ_WRITE requests are generated. The average detection delay is about 2.26 encrypted files with 0.15M of encrypted content. In contrast, ERW-ShieldFS detects traditional samples with an average detection time of 0.40s and an average delay of 3.87 encrypted files with 0.17M of encrypted content. During this period, an average of 171.74 IRP_MJ_WRITE requests are generated. It can be observed that the detection cost of ERW-ShieldFS is higher than that of ERW-Radar.

For evasive samples, some of them (ANI^{WPS}) encrypt only one file with 0.003M of encrypted content before being detected by ERW-Radar. Even the samples from the Lockergoga family encrypt up to 8 files, they encrypt less than 1.22M of file content. Overall, ERW-Radar detects evasive samples with an average detection delay of 3.22 encrypted files and 0.24M of encrypted content, an average detection time of 0.40s, and an average of 240.83 IRP_MJ_WRITE requests generated. Such a low detection delay indicates that ERW-Radar can detect evasive attacks timely. In contrast, ERW-ShieldFS fails to detect them.

An interesting conclusion can be drawn from Tab. III: The detection time of samples imitating similar applications (e.g., $ANI^{MSOffice}$ and ANI^{WPS}) is relatively close, while it varies significantly across samples imitating different applications. That is because these samples behave identically due to employing similar behavior templates, which differ significantly among different applications. We demonstrate the behavioral similarity in Appendix A.

3) Sensitivity Analysis: To determine the optimal configuration, we conduct sensitivity analysis for different hyper-parameters based on partial dataset. As Fig. 4 (a) shows, a low α leads to a high FPR due to occasional repetitive behavior segments of benign programs, but a high α indicates a delayed alert, i.e., requiring more positive results to trigger an alert. We recommend $\alpha = 0.75$ as detection threshold as it's the turning point and accuracy levels off afterward. In Fig. 4 (b), generally, the detection accuracy increases with s_{com} but decreases with s_{de}. This indicates that the accuracy of *Detector* benefits from larger values of s_{com} and smaller values of s_{de}. We recommend choosing $1536 : 16$ (i.e., 96:1) as the ratio of behavior sequence lengths. In Fig. 4 (c), larger l_{en} leads to higher accuracy while smaller l_{de} contributes to lower inference latency. This can be attributed to the fact that deep encoder is better at capturing contextual information while shallow decoder benefits the computational efficiency. Accordingly, a structure of $9 - 3$ is a good choice. In Fig. 4 (d), we sort IRP types by frequency and gradually remove the lowest one. We observe that fewer IRPs contribute to lower CPU utilization, but lead to lower detection accuracy. An optimal balance is achieved at the inflection point, $t = 8$, where ERW-Radar sacrifices little in terms of accuracy but results in significantly lower overhead.

To demonstrate the safety of our polling queue (§V-B), we compare it with an asynchronous one that transmits information only when it is full. We test accuracies, detection delays and transmission overheads of ERW-Radar equipped with the two queues respectively. Results show their detection accuracies are similar, with the polling one of 96.65% and the other

[1] "1(+3) and 27(+0.009)" refers to an average of one file with 27 KB of data being substantively encrypted before samples are detected by ERW-Radar, and 3 other PDF files with only 9 structure-related bytes encrypted. So it is with "1(+2) and 20(+0.006)". While these structure-related bytes (0.009 KB/0.006 KB) are encrypted, the object content remains intact, indicating that the original data may be obtained from these PDF objects. Thus, they are not considered to be substantively encrypted. Note that the data recovery discussed here is only possible in such extremely specific cases. General recovery of data from partially encrypted files is not the focus of this paper and will be left for future work.

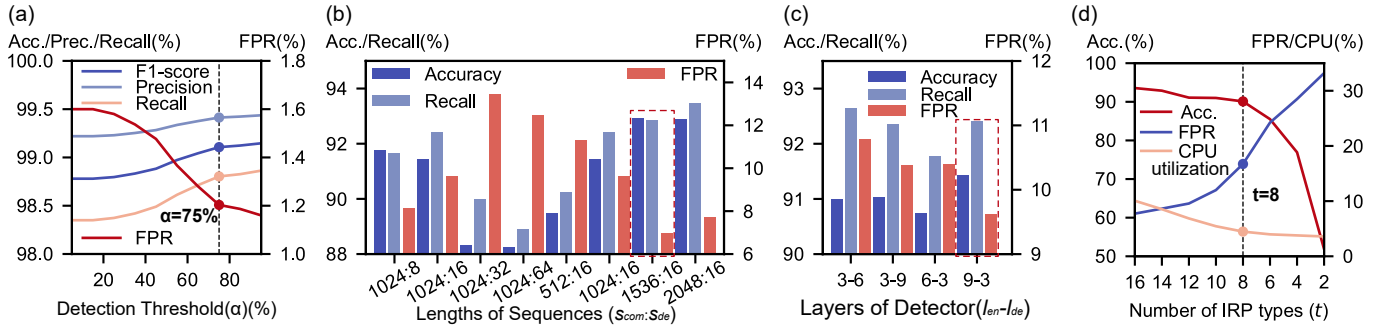


Fig. 4: Performance of Detector with varying parameters. (a) Impact of detection threshold, α, on performance. (b) Impact of the ratio of input sequence lengths on performance. s_{com}:s_{de} refers to the ratio of the historical behavior sequence length to the latest behavior sequence length. (c) Impact of Detector layers on performance. l_{en} and l_{de} denote the layers of the Encoder and Decoder respectively. (d) Impact of the number of IRP types, t, on performance.

one of 96.23%. Additionally, compared to the asynchronous one, which has an average detection delay of 0.97M and a transmission overhead of 1.21%, the polling queue reduces the former to 0.20M while only increasing the latter by 0.47%, It indicates the polling queue ensures more timely detection (higher security) without incurring significant overhead.

C. Evaluation of Content Analysis of ERW-Radar

1) Effectiveness: We utilize file dataset in §III to evaluate the effectiveness of content analysis. Results show that the *Analyzer* distinguishes encrypted files effectively, with all of them detected and the FPR being 3.00%. The FP files are misclassified because most of their analyzed segments are smaller than 1K. The dataset is constructed by randomly selecting segments of different sizes from various files, which results in the under-sampling of larger segments in some files. Fortunately, during runtime, the *Analyzer* samples segments of various sizes from each target file distributively, which avoids misclassifications due to the lack of large-sized segments.

2) Sensitivity Analysis: We conduct sensitivity analysis for the exponential smoothing parameter β and the number of historical time intervals n. Fig. 5 shows the trend curves with varying parameters. It can be observed that different values of β all exhibit a certain degree of lag when measuring the trend in time interval changes. The curve with $\beta = 0.2$ reflects changes in trends more quickly, mainly because its weight coefficients decay faster over time. The curve with $n = 25$ fits the actual data curve more accurately. Thus, we recommend a configuration of $\beta = 0.2$ and $n = 25$.

D. Overhead of ERW-Radar

To evaluate the impact on storage, CPU and memory performance, we perform benchmarks and real-world scenario tests (also performed in other ransomware defense solutions [60], [7], [61], [62], [63], [5]). We compare the overhead of ERW-Radar with that of four commonly used applications: Microsoft Word, WinRAR, PyCharm, and Firefox. Additionally, we also investigate the overheads of two representative defense solutions, ShieldFS [5] and RansomTag [60], and compare their overheads with that of ERW-Radar.

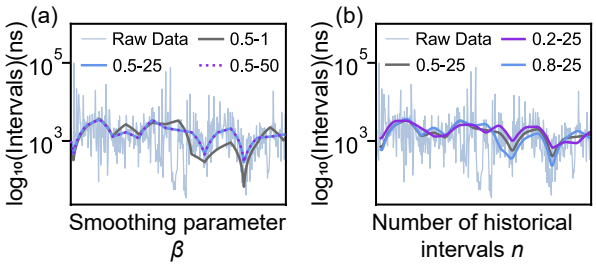


Fig. 5: Trend curves with varying parameters. 'Raw Data' refers to the original time intervals of IRPs. The '0.2-25' curve refers to calculating the *I/O trend* with $\beta = 0.2$, $n = 25$.

1) Benchmarks: We utilize IOZone [64], a popular storage performance tool, to test I/O throughput under sequential/random read and write. We set the system where the CPU is running IOZone at full load as the baseline and measure the decrease in I/O throughput caused by running ERW-Radar and other applications. As Fig. 6 (a) shows, for workloads of sequential read/write, the I/O throughput does not change greatly after running ERW-Radar, only decreasing by 0.81%/0.42%. However, for other applications, it decreases by 14.11%/8.36%. This indicates that the I/O overhead introduced by ERW-Radar is negligible. Even though the I/O throughput decreases by 1.29%/2.44% under random read/write after running ERW-Radar, it is much lower than that of RansomTag [60] which reports an I/O overhead of about 3.77%/4.00%, and other applications with an average I/O overhead of 23.21%/13.96% under random read/write.

We also employ SPEC-CPU 2017 [65] (Perlbench-mark, Blender) and Phoronix-test-suits [66] (Compress-7zip, OpenSSL) to evaluate the impact of ERW-Radar and other applications on CPU performance. We measure the execution time, MIPS, and Signs/S variations compared to running benchmarks alone. As Fig. 6 (b) shows, ERW-Radar causes an average of 3.08% performance decrease in processing computational tasks, as evaluating behavioral correlations and χ^2 tests consume computational resources. Even so, its overhead is similar to that of ShieldFS which reports an overhead

of 3.8%, and is significantly lower than that of Microsoft Word, WinRAR, PyCharm, and Firefox, whose overheads are 11.33%, 10.60%, 9.75%, and 10.25%, respectively.

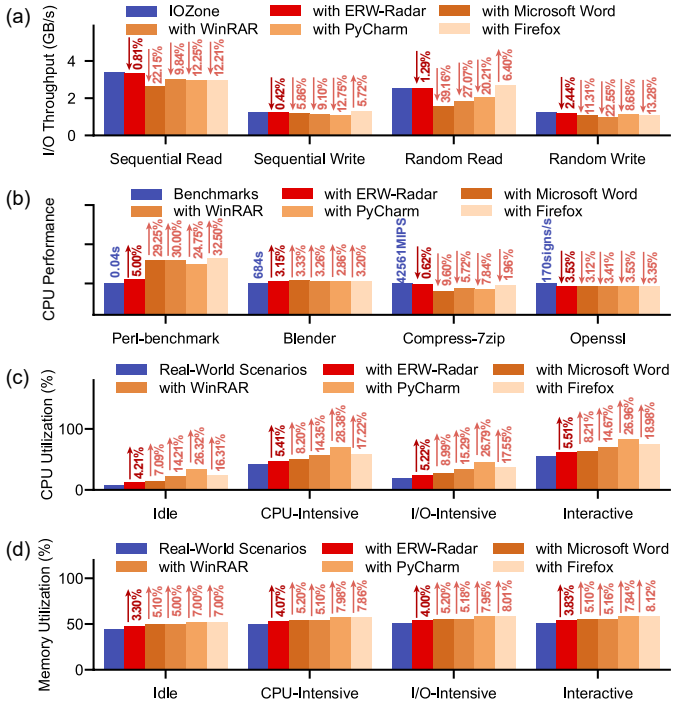


Fig. 6: Overhead of ERW-Radar and other applications.

2) Real-world scenarios: We also evaluate the overhead of ERW-Radar in real-world scenarios. We set up four scenarios with different workloads, each consisting of commonly used Windows applications. They are (1) a CPU-Intensive scenario where compilers, e.g., Microsoft Visual C++, are running; (2) an I/O-Intensive scenario where file editors, e.g., WPS, are running; (3) an interactive scenario where web browsers, e.g., Chrome, are running; (4) an idle scenario where no application is running. We test the CPU and memory utilization of the system in different scenarios as baselines. Then, we measure their increases after running ERW-Radar and the four applications for overhead comparison.

In Fig. 6 (c), we observe that CPU utilization increases by 4.21% in idle scenarios and 5.22%-5.51% in busy scenarios after running ERW-Radar, similar to RansomTag, which reports a CPU overhead of 4.00%–6.00%. However, this is obviously lower than the CPU overheads caused by other applications, which range from 8.12% to 27.11%. As Fig. 6 (d) shows, the memory utilization of ERW-Radar is about 3.30%–4.07%. It indicates that ERW-Radar is of moderate size and does not impose a heavy burden on various real-world systems during running. Compared to ShieldFS which reports a memory overhead of 0.95%–8.53%, and other applications with memory overheads ranging from 5.11% to 7.75%, ERW-Radar maintains its overhead within an acceptable range.

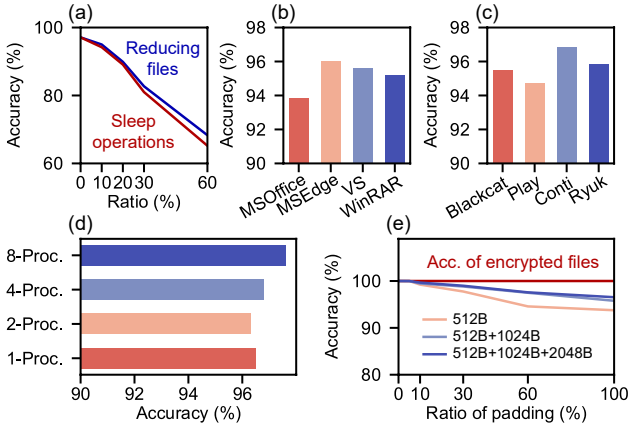


Fig. 7: Impact of various adaptive attacks on accuracy.

E. Evaluation Against Adaptive Attackers

It's important to investigate if attackers can make a simple modification to evade ERW-Radar. Theoretically, based on its functions and components, ERW-Radar faces three attack vectors: targeted attacks on the I/O monitor module, behavior detection mechanism and content analysis mechanism. Next, we evaluate their resilience against them.

1) Targeted Attacks on the I/O Monitor Module: The potential attack on this module is the DoS attack, where an adversary may perform dummy I/O access to overflow the queue. This probably results in legitimate I/O requests not being detected, thereby affecting detection performance. To evaluate the resilience against it, we make processes continuously send numerous I/O requests to access the target files, aiming to overload the system. Meanwhile, we also run ransomware. Then we test average accuracies and detection delays before and after executing the DoS attack. Results show the DoS attack does not affect the detection accuracy (96.23% before and 96.17% after). Instead, it reduces the detection delay from 0.2M to 0.13M, because performing dummy I/O access leads to a sustained high frequency of similar I/O operations, which exhibits remarkable behavioral correlation and makes it easier to detect before exhausting system resources. This proves the resilience of our proposal against the DoS attack.

2) Targeted Attacks on Behavior Detection Mechanism: Since ERW-Radar detects the behavioral repetitiveness of ransomware by analyzing the behavioral correlation, adversaries may tamper with it by weakening the correlation. There are four evasive techniques that they might adopt. Accordingly, we develop four prototypes by modifying an open-source project [67]: ① slowing down the attack speed by reducing the encrypted files or randomly introducing sleep operations during encryption, ② interfering with I/O behaviors by running ransomware samples in §VI-A with 4 benign programs, WinRAR, VS, MSEdge and MSOffice respectively, ③ employing new encryption templates by evaluating ERW-Radar with new variants of intermittent attacks, ④ creating symbolic links for files and designing multi-process attacks to access the same files via different paths. The impact of the four prototypes on

detection accuracy is shown in Fig. 7 (a)-(d) respectively.

Results show that prototypes ②③ have negligible impact on detection accuracy, with an average decrease of 1.50% and 0.93%. Because in ②, competition for I/O resources weakens encryption behaviors, making the behavior segments not as similar as before, and in ③, employing new encryption strategies diversifies the behavior segments. Since each behavior template is still repetitively utilized, ②③ do not affect the behavioral correlation. In ④, ERW-Radar detects multi-process attacks with an average accuracy of up to 96.80%. Because ERW-Radar analyzes the correlation of file access operations across global processes, it can identify similar file access patterns in multiple processes or detect multiple process groups repetitively working on different stages of the encryption chain. Path fooling does not change these operations, thus not affecting the behavioral correlation. In contrast, slowing down the attack speed affects the accuracy significantly. Fig. 7 (a) shows that, as the ratio of sleep operations increases or the encrypted files decreases by 10%, the detection accuracy only decreases by 2.91% and 2.10%. But in a worse case of more than 60%, the repetitiveness becomes hard to identify (decreases by 31.92% and 28.81%). Mainly because fewer repetitive behavior segments remain within the detected behavior sequences, thus disrupting the repetitiveness. In such cases, even if *Detector* hardly identifies these attacks, we can still rely on *Analyzer* to evaluate whether files are encrypted.

3) Targeted Attacks on Content Analysis Mechanism: As the χ^2 test targets the intrinsic characteristic, high randomness, of encrypted files, adversaries may evade it by reducing the randomness of encrypted file segments deliberately. Typically, they tend to pad structured data consisting of null characters, e.g., character '0'. Based on this, we develop prototype ⑤ to evaluate the resilience against such attacks. It encrypts files and pads low-randomness data to them. We extract segments of various sizes from the padded files and evaluate the detection accuracy. The detection accuracy of encrypted files is obtained by aggregating segment detection results. Fig. 7 (e) shows that detecting file segments of various sizes ensures significant resilience against such attacks, with accuracy decreasing by less than 3.56%. Even if some of the encrypted segments are not detected, it does not affect the detection of the whole encrypted files because the majority of their segments can be detected. While adversaries can evade the χ^2 test by padding data, the probabilities of the padded data significantly increases, making the padded files easy to detect. Moreover, ERW-Radar randomly analyzes file segments of various sizes from different positions, so adversaries cannot predict or pinpoint all the analyzed segments for tampering.

VII. Discussion

First, we are committed to developing a novel ransomware detection system in this paper. In our design, behavioral information is extracted from IRPs. We choose IRPs as targets because they are located at the closest layer to the filesystem, with access to nearly all objects of the OS. This makes it hard for adversaries to evade. While the information can also be extracted by hooking APIs, this approach may be bypassed by using customized encryption algorithms. What's more, the types of IRPs involved in the encryption process are much fewer than those of APIs, which can number in the hundreds. This avoids constructing a large-scale API encoding space to characterize behaviors, which would result in high analysis overhead. We have demonstrated that our chosen IRPs are able to detect malicious behaviors (see §VI-B).

Second, our detection system employs the behavior information extracted in the kernel to precisely detect ransomware in the user space. As mentioned in §IV, it may be ineffective when adversaries attack the OS and terminate detection. Fortunately, several promising approaches have been proposed to mitigate this issue, e.g., (1) delegating the process management of ERW-Radar to the ASP which can manage security mechanisms and is physically isolated from CPU [47], (2) building trust area with fine-grained memory protection, then performing behavior detection and content analysis in it [48], or (3) detecting OS inconsistencies over the lifetime [49]. Thus, by adopting such an approach, we can mitigate the threats posed by privileged attacks on ERW-Radar.

Third, our work is focused on the Windows system, as it's the most popular targeted OS of ransomware authors [68], [69], [70]. Since our detection system presented here is out-of-the-box, it can be readily deployed and configured on desktop PCs, servers, or modern cloud platforms as it is user-friendly and only requires a few computational resources. Similarly, we focus our study on software cases, as they are prevalent. Note that nothing prevents the deployment of the techniques presented here in hardware as modern hardware also embeds detection algorithms to enhance threat detection capabilities.

VIII. Related Work

Crypto Ransomware is a type of malware that inflicts significant economic loss [1], [71], [3], [4]. Many defense solutions have been proposed to mitigate loss [23], [7], [72], [9]. Traditional signature-based detection techniques [73], [74], such as binary feature engineering [75] and signature matching [76] have difficulty in detecting the latest ransomware and are easily bypassed by the code obfuscation technique [77], [8]. Thus, studies on ransomware detection [5], [9], [7] focus more on analyzing it dynamically. For example, PayBreak [63] strives to hook specific cryptographic APIs during encryption. Given that they fail to detect ransomware which utilizes customized cryptographic libraries, I/O behavior-based detection systems are developed [6], [78], [79], [7], [5], [72], [80], [81], [82], [83]. They rely on predefined behavioral features or patterns to model individual malicious processes, thus may be bypassed by multi-process attacks as mentioned in §II. By analyzing the behavioral correlation and the randomness of encrypted content, we find an opportunity to detect this evasive ransomware and develop a novel detection system.

Since privileged attacks may disable the detection systems, some researchers propose SSD-based solutions [84], [61] which monitor the low-level I/O behavior of disk hardware.

These solutions can deal with privileged attacks with better transparency due to isolating from the OS. However, they are vulnerable to variants that modify I/O behaviors beyond their limited observable semantics, and are difficult to deploy due to relying on customized non-commercial SSDs. In contrast, our off-the-shelf mechanism can observe rich behavioral semantics on a variety of platforms and effectively detect evasive attacks.

With the development of parallelism techniques, attackers attempt to distribute their malicious payloads into multiple small processes in a system. Each process looks unsuspicious for detectors but is malicious when acting in cooperation [37], [38], [39]. This compromises the capability of behavior-based detection by breaking the original malicious patterns of ransomware. This also makes SSD-based solutions unable to determine how long they need to retain files, thereby complicating the data recovery process. We observe that it is possible to detect such distributed attacks by collecting dispersed malicious behaviors from multiple processes and identifying correlations between behavior segments. In the future, such distributed ransomware is likely to evolve towards distributing malicious payloads across different systems/nodes within a local area network [85], [36]. In such cases, only monitoring a single node cannot detect the attack due to the lack of a full scope of attack behaviors. Therefore, we can expand our detection system by adopting a client-server model. This model deploys the detection system on a central server, which collects behavioral information from different nodes. After analyzing it, the server promptly provides feedback to each node. This solution is feasible for two reasons: First, only a minimal amount of behavioral information needs to be transmitted across the network; Second, the server has the computational power to support real-time behavior analysis.

IX. Conclusion

Evasive ransomware has successfully evaded existing detection systems. In this paper, we first discover I/O behaviors of evasive ransomware exhibit a unique repetitiveness characteristic. Besides, the χ^2 test and the probability distribution of byte streams can be used to identify encrypted files. Inspired by this, we design a hybrid detection system called ERW-Radar. To detect malicious behaviors robustly and accurately, we propose a *Correlation* mechanism and design a dynamic detection window. Besides, we also design a content analyzer embedded with an adaptive mechanism to start it at idle I/O cycles. Experiments show that ERW-Radar can detect evasive ransomware with an accuracy of 96.18% while maintaining a FPR of 5.36%. The overhead of running ERW-Radar is only 5.09% in CPU utilization and 3.80% in memory utilization.

Acknowledgment

We sincerely thank the anonymous reviewers and our anonymous shepherd for valuable feedback. This work is supported by the National Science Fund for Distinguished Young Scholars under Grant No. 62125208.

References

[1] "Ransomware: What it is & what to do about it," https://www.ic3.gov/Content/PDF/Ransomware_Fact_Sheet.pdf, 2023.

[2] T. M. T. I. team, "2023 state of ransomware," https://try.malwarebytes.com/business-2023-state-of-ransomware/.

[3] J. McKeon, "Fbi: Healthcare hit with most ransomware attacks of any critical sector," https://www.techtarget.com/healthtechsecurity/news/366594412/FBI-IC3-Victims-Racked-Up-103B-in-Losses-Tied-to-Internet-Crime-Last-Year.

[4] Sophos, "The state of ransomware 2023," https://www.sophos.com/en-us/content/state-of-ransomware.

[5] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: A self-healing, ransomware-aware filesystem," in *ACSAC '16*. ACM SIGCOMM, 2016, p. 336–347.

[6] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "Cryptolock (and drop it): Stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 303–312.

[7] S. Mehnaz, A. Mudgerikar, and E. Bertino, "Rwguard: A real-time detection system against cryptographic ransomware," in *Research in Attacks, Intrusions, and Defenses*, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds. Cham: Springer International Publishing, 2018, pp. 114–136.

[8] E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware (keynote)," in *IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2016.

[9] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *Research in Attacks, Intrusions, and Defenses*, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds. Cham: Springer International Publishing, 2017, pp. 98–119.

[10] "Eine pause einlegen, ihr unternehmen sichern und dabei noch sparen!" https://www.kaspersky.de/, 2023.

[11] "Everyday online security with microsoft defender," https://www.microsoft.com/en-us/microsoft-365/microsoft-defender-for-individuals, 2023.

[12] "Fix today. protect forever." https://www.malwarebytes.com/, 2023.

[13] C. Zhou, L. Guo, Y. Hou, Z. Ma, Q. Zhang, M. Wang, Z. Liu, and Y. Jiang, "Limits of i/o based ransomware detection: An imitation based attack," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2584–2601.

[14] R. Moussaileb, N. Cuppens-Boulahia, J.-L. Lanet, and H. L. Bouder, "A survey on windows-based ransomware taxonomy and detection mechanisms," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1 – 36, 2021.

[15] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," *CoRR*, vol. abs/2102.06249, 2021.

[16] A. Alqahtani and F. T. Sheldon, "A survey of crypto ransomware attack detection methodologies: An evolving outlook," *Sensors*, vol. 22, no. 5, p. 1837, 2022. [Online]. Available: https://doi.org/10.3390/s22051837

[17] S. Razaulla, C. Fachkha, C. Markarian, A. Gawanmeh, W. Mansoor, B. C. M. Fung, and C. Assi, "The age of ransomware: A survey on the evolution, taxonomy, and research directions," *IEEE Access*, vol. 11, pp. 40 698–40 723, 2023.

[18] K. Begovic, A. Al-Ali, and Q. M. Malluhi, "Cryptographic ransomware encryption detection: Survey," *Comput. Secur.*, vol. 132, p. 103349, 2023. [Online]. Available: https://doi.org/10.1016/j.cose.2023.103349

[19] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, and L. V. Mancini, "The naked sun: Malicious cooperation between benign-looking processes," in *Applied Cryptography and Network Security*. Cham: Springer International Publishing, 2020, pp. 254–274.

[20] F. De Gaspari, D. Hitaj, G. Pagnotta, L. De Carli, and L. Mancini, "Evading behavioral classifiers: a comprehensive analysis on evading ransomware detection techniques," *Neural Computing and Applications*, vol. 34, pp. 1–20, 07 2022.

[21] J. W. Aleksandar Milenkoski, "crimeware-trends-ransomware-developers-turn-to-intermittent-encryption-to-evade-detection," https://www.sentinelone.com/labs/crimeware-trends-ransomware-developers-turn-to-intermittent-encryption-to-evade-detection/.

[22] H. Oz, A. Aris, A. Acar, G. S. Tuncay, L. Babun, and A. S. Uluagac, "Røb: Ransomware over modern web browsers," in *USENIX Security Symposium*, 2023.

[23] T. McIntosh, J. Jang-Jaccard, P. Watters, and T. Susnjak, "The inadequacy of entropy-based ransomware detection," in *Neural Information Processing*, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham: Springer International Publishing, 2019, pp. 181–189.

[24] F. D. Gaspari, D. Hitaj, G. Pagnotta, L. D. Carli, and L. V. Mancini, "Encod: Distinguishing compressed and encrypted file fragments," *ArXiv*, vol. abs/2010.07754, 2020.

[25] F. Casino, K.-K. R. Choo, and C. Patsakis, "Hedge: Efficient traffic classification of encrypted and compressed packets," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 2916–2926, 2019.

[26] "deep-dive-into-alphv-blackcat-ransomware," https://securityscorecard.com/research/deep-dive-into-alphv-blackcat-ransomware/, 2023.

[27] "Who is black basta?" https://www.blackberry.com/us/en/solutions/endpoint-security/ransomware-protection/black-basta, 2023.

[28] "Ransom.lockergoga," https://www.malwarebytes.com/blog/detections/ransom-lockergoga, 2023.

[29] "play," https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-play, 2023.

[30] Acronis, "Threat analysis: Doppelpaymer ransomware," https://www.acronis.com/en-us/blog/posts/doppelpaymer-ransomware/, 2021.

[31] S. F. Brett Stone-Gross and B. Hartley, "Bitpaymer source code fork: Meet doppelpaymer ransomware and dridex 2.0," https://www.crowdstrike.com/blog/doppelpaymer-ransomware-and-dridex-2/, 2019.

[32] M. FIGUEROA, "An inside look at how ryuk evolved its encryption and evasion techniques," https://www.sentinelone.com/labs/an-inside-look-at-how-ryuk-evolved-its-encryption-and-evasion-techniques/, 2020.

[33] A. Zaharia, "Everything you need to know about cryptowall," https://heimdalsecurity.com/blog/cryptowall-ransomware/, 2022.

[34] CryptoWall, "What is cryptowall ransomware?" https://www.proofpoint.com/us/threat-reference/cryptowall-ransomware, 2022.

[35] I. WEIZMAN, "Conti unpacked-understanding ransomware development as a response to detection," https://www.sentinelone.com/labs/conti-unpacked-understanding-ransomware-development-as-a-response-to-detection/, 2021.

[36] VirusShare, "Conti ransomware – how it works and 4 ways to protect yourself," https://www.datto.com/blog/conti-ransomware-how-it-works-and-4-ways-to-protect-yourself/, 2023.

[37] M. Botacin, P. L. de Geus, and A. R. A. Grégio, ""vanilla" malware: vanishing antiviruses by interleaving layers and layers of attacks," *Journal of Computer Virology and Hacking Techniques*, pp. 1–15, 2019.

[38] J. Pavithran, M. Patnaik, and C. Rebeiro, "D-TIME: distributed threadless independent malware execution for runtime obfuscation," in *13th USENIX Workshop on Offensive Technologies, WOOT 2019, Santa Clara, CA, USA, August 12-13, 2019*. USENIX Association, 2019.

[39] W. Ma, P. Duan, S. Liu, G. Gu, and J.-c. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *Journal in Computer Virology*, vol. 8, 05 2011.

[40] M. Loman, "Lockfile ransomware's box of tricks: intermittent encryption and evasion," https://news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-of-tricks-intermittent-encryption-and-evasion/.

[41] "Malwarebazaar," https://bazaar.abuse.ch/, 2023.

[42] "Virusshare.com - because sharing is caring," https://virusshare.com/, 2023.

[43] C. Zhou, "Animagus," https://github.com/ChijinZ/Animagus, 2023.

[44] K. et al., "Kinetics," https://deepmind.google/, 2024.

[45] "University libraries unt digital library," https://digital.library.unt.edu/ark:/67531/metadc1757662/, 2017.

[46] T. M. Corporation, "Access token manipulation," https://attack.mitre.org/techniques/T1134/, 2022.

[47] D. Meng, R. Hou, G. Shi, B. Tu, A. Yu, Z. Zhu, X. Jia, Y. Wen, and Y. Yang, "Built-in security computer: Deploying security-first architecture using active security processor," *IEEE Transactions on Computers*, vol. 69, pp. 1571–1583, 2020.

[48] Y. Chen, S. Reymondjohnson, Z. Sun, and L. Lu, "Shreds: Fine-grained execution units with private memory," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 56–71.

[49] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion, "Sok: Introspections on trust and the semantic gap," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 605–620.

[50] N. A. Huynh, W. K. Ng, and H. G. Do, "On periodic behavior of malware: experiments, opportunities and challenges," in *2016 11th International Conference on Malicious and Unwanted Software (MALWARE)*, 2016, pp. 1–8.

[51] Y.-R. Yeh, T. C. Tu, M.-K. Sun, S. M. Pi, and C.-Y. Huang, "A malware beacon of botnet by local periodic communication behavior," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, 2018, pp. 653–657.

[52] "registering-the-minifilter-driver," https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/registering-the-minifilter-driver, 2023.

[53] "Io-request-packets," https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/i-o-request-packets, 2023.

[54] B. Lenaerts-Bergmans, "Denial-of-service (dos) attacks," https://www.crowdstrike.com/cybersecurity-101/denial-of-service-dos-attacks/, 2023.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[56] M. Xu, Y. Xiong, H. Chen, X. Li, W. Xia, Z. Tu, and S. Soatto, "Long short-term transformer for online action detection," *CoRR*, vol. abs/2107.03377, 2021.

[57] A. Jaegle, F. Gimeno, A. Brock, A. Zisserman, O. Vinyals, and J. Carreira, "Perceiver: General perception with iterative attention," in *International Conference on Machine Learning*, 2021.

[58] M. M. Botacin, "Malware.multicore," https://github.com/marcusbotacin/Malware.Multicore, 2019.

[59] "2012 ieee symposium on security and privacy prudent practices for designing malware experiments: Status quo and outlook," 2012.

[60] B. Ma, Y. Yang, J. Li, F. Zhang, W. Shen, Y. Zhou, and J. Ma, "Travelling the hypervisor and SSD: A tag-based approach against crypto ransomware with fine-grained data recovery," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, November 26-30, 2023*. ACM, 2023, pp. 341–355.

[61] S. Baek, Y. Jung, A. Mohaisen, S. Lee, and D. Nyang, "Ssd-insider: Internal defense of solid-state drive against ransomware with perfect data recovery," *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 875–884, 2018.

[62] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, "Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware," *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[63] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "Paybreak: Defense against cryptographic ransomware," *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[64] D. Capps, "Iozone filesystem benchmark," http://iozone.org/, 2016.

[65] "Standard performance evaluation corporation," http://iozone.org/, 2022.

[66] "Phoronix test suite," https://www.phoronix-test-suite.com/, 2024.

[67] "ransomware-source-code," https://github.com/topics/ransomware-source-code, 2023.

[68] Netmarketshare, "How the most damaging ransomware evades it security," https://netmarketshare.com/operating-system-market-share.aspx, 2018.

[69] M. S. Fernandez, "The platform matters: A comparative study on linux and windows ransomware attacks," https://research.checkpoint.com/2023/the-platform-matters-a-comparative-study-on-linux-and-windows-ransomware-attacks/, 2023.

[70] J. Panetti, "Major operating systems targeted by ransomware," https://ransomware.org/blog/major-operating-systems-targeted-by-ransomware/, 2022.

[71] R. Sobers, "Ransomware statistics, data, trends, and facts," https://www.varonis.com/blog/ransomware-statistics/trends.

[72] B. A. S. Al-rimy, M. A. B. Maarof, Y. A. Prasetyo, S. Z. M. Shaid, and A. F. M. Ariffin, "Zero-day aware decision fusion-based model for crypto-ransomware early detection," *International Journal of Integrated Engineering*, 2018.

[73] M. Medhat, S. Gaber, and N. Abdelbaki, "A new static-based framework for ransomware detection." IEEE Computer Society, aug 2018, pp. 710–715.

[74] K. P. Subedi, D. R. Budhathoki, and D. Dasgupta, "Forensic analysis of ransomware families using static and dynamic analysis," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 180–185.

[75] X. Hu, K. G. Shin, S. Bhatkar, and K. Griffin, "Mutantx-s: Scalable malware clustering based on static features," in *USENIX Annual Technical Conference*, 2013.

[76] M. Brengel and C. Rossow, "Yarix: Scalable yara-based malware intelligence," in *USENIX Security Symposium*, 2021.

[77] A. Moser, C. Krügel, and E. Kirda, "Limits of static analysis for malware detection," *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421–430, 2007.
[78] F. Mbol, J.-M. Robert, and A. Sadighian, "An efficient approach to detect torrentlocker ransomware in computer systems," in *Cryptology and Network Security*, S. Foresti and G. Persiano, Eds. Cham: Springer International Publishing, 2016, pp. 532–541.
[79] M. Held and M. Waldvogel, "Fighting ransomware with guided undo," in *NISK 2018 : Proceedings of the 11th Norwegian Information Security Conference*, ser. NISK Journal, S. F. Mjølsnes and R. Soleng, Eds., no. 11, 2018.
[80] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Crypto-ransomware early detection model using novel incremental bagging with enhanced semi-random subspace selection," *Future Generation Computer Systems*, vol. 101, pp. 476–491, 2019.
[81] Z.-G. Chen, H.-S. Kang, S. nan Yin, and S.-R. Kim, "Automatic ransomware detection and analysis based on dynamic api calls flow graph," *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, 2017.
[82] S. Maniath, A. Ashok, P. Poornachandran, V. G. Sujadevi, A. U. P. Sankar, and S. Jan, "Deep learning lstm based ransomware detection," *2017 Recent Developments in Control, Automation & Power Engineering (RDCAPE)*, pp. 442–446, 2017.
[83] Y. Takeuchi, K. Sakai, and S. Fukumoto, "Detecting ransomware using support vector machines," *Workshop Proceedings of the 47th International Conference on Parallel Processing*, 2018.
[84] B. Reidys, P. Liu, and J. Huang, "Rssd: defend against ransomware with hardware-isolated network-storage codesign and post-attack analysis," *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.
[85] "How the most damaging ransomware evades it security," https://news.sophos.com/en-us/2019/11/14/how-the-most-damagin g-ransomware-evades-it-security/, 2023.

APPENDIX A
BEHAVIOR ANALYSIS AND FEATURE VISUALIZATION.

A. Behaviors of splitting attacks and intermittent attacks

In Section III, we choose one sample from imitating attacks to conduct observations and find the unique repetitiveness characteristic in its I/O behaviors. To prove it, we also choose one sample from splitting attacks and one sample, Lockergoga from intermittent attacks. We display their I/O behaviors in Fig. 8 and Fig. 9. It can be intuitively observed in Fig. 8 that each group exhibits repetitive segments of I/O behaviors, which are delineated by dashed rectangles. This is because the authors of splitting attacks divide encryption operations into four groups: Group a: {DirectoryControl}, Group b: {DirectoryControl}, Group c: {Read}, and Group d: {Read, Write, SetInformation}. Each group executes only specific operations repetitively, leading to the observed repetitiveness in I/O behaviors. Note that the repetitive segments between different groups are not consistent due to the varying types and frequencies of operations within each group. However, this does not impact the overall behavioral repetitiveness.

The behavioral repetitiveness is also obvious in intermittent attacks, as is shown in Fig. 9. We can visually observe that attackers repeat the work-sleep patterns during encryption. This indicates the intermittent technique adopted in the evasive ransomware also results in repetitive I/O behaviors.

B. Similarity of Ransomware Families

We conduct experiments on samples randomly chosen from each family to analyze their behavioral characteristics, as samples from the same family typically exhibit nearly identical

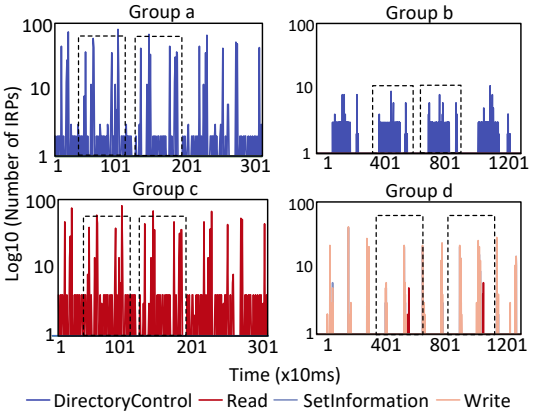


Fig. 8: I/O behaviors of a splitting attack.

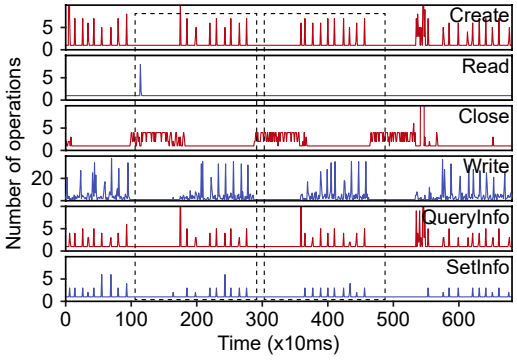


Fig. 9: I/O behaviors of an intermittent attack: Lockergoga.

behavioral patterns during runtime. To prove it, we randomly choose four samples from an Imitating Attack family ANIMAGUS, one sample from an Intermittent Attack family Lockergoga and one sample from the Splitting Attack family Conti. We evaluate the similarity of their runtime I/O behaviors using the Ratcliff-Obershelp Algorithm. As Fig. 10 shows, the similarity between four imitating samples ranges from 0.76 to 0.85. In contrast, it ranges from 0.18 to 0.43 between samples from different families. This indicates samples from a family behave identically at runtime, and samples from different family exhibit significant behavioral differences.

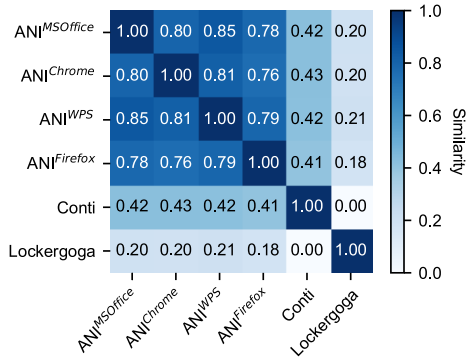


Fig. 10: Runtime behavior similarity of ransomware samples from different families.

C. Visualization of Feature Differences

We apply t-Distributed Stochastic Neighbor Embedding (t-SNE) to quantify the differences between various features in identifying evasive ransomware. We choose the features of ShieldFS and the Correlation Weight of ERW-Radar as the metrics. we choose two samples from the Imitating Attack family ANIMAGUS, two benign programs, Microsoft Office and WPS, and two samples from traditional ransomware family, Cerber and Diavol. t-SNE is used to transform the features into a two-dimensional space for visualization. In Fig. 11 (a), the difference between evasive samples and imitated benign programs is much smaller than that between traditional ransomware samples and benign programs. Thus, detection tools cannot make a correct decision based on the features of ShieldFS. In contrast, in Fig. 11 (b), the difference between evasion samples and benign programs is much longer. Therefore, ERW-Radar can detect evasion ransomware accurately based on the unique behavioral repetitiveness.

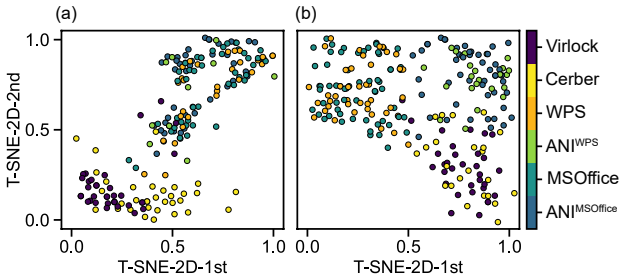


Fig. 11: Visualization of behavior features of different programs. The features used here are in line with ShieldFS and ERW-Radar. The dimension reduction is performed by t-SNE.

APPENDIX B
SUPPLEMENTARY FOR THE MONITORED IRPS.

In this section, we provide the description of IRPs which are parsed by the I/O Monitor. We analyze all IRPs generated during the encryption and filter them out following the steps in §V-B. After that, we obtain 8 types of IRPs. They are:

•IRP_MJ_CREATE: when a user attempts to open or create a file or directory.

•IRP_MJ_READ: when a user attempts to read a file.

•IRP_MJ_WRITE: when a user attempts to write to a file.

•IRP_MJ_CLEANUP: when a user attempts to close the last file handle.

•IRP_MJ_QUERY_INFORMATION: when a user endeavors to retrieve information about a file or directory.

•IRP_MJ_SET_INFORMATION: when a user attempts to modify the security attributes of a file or directory.

•IRP_MJ_QUERY_VOLUME_INFORMATION: when a user attempts to retrieve information about a volume.

•IRP_MJ_DIRECTORY_CONTROL: when a user performs directory operations.

APPENDIX C
DETAILS OF COMPRESSOR

The decoder of traditional Transformer architecture is typically used to create a context that helps to predict the next element in the sequence by combining the input sequence (from the encoder) and the previously generated output elements. This process is facilitated by mechanisms like attention, which weighs the relevance of different parts of the input and output sequences to generate accurate predictions. Therefore, the cross-attention mechanism in the decoder can be utilized to compress a sequence. It focuses on the most relevant parts of the input sequence, filters out redundant or less important information and effectively summarizes the input into a more compact representation. It is useful in cases where essential features need to be captured from a large amount of data.

As Fig. 3 shows, there are two attention layers of a decoder unit. The lower layer uses multi-head attention to calculate the self-attention of the input Y, which provides a compressed vector representation with dimensions $n * c$. The calculation process is as follows:

$$\begin{aligned} &Q = (Y+P)W^Q, \quad K = (Y+P)W^K, \quad V = (Y+P)W^V \\ &Y_+ = \text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \end{aligned} \tag{1}$$

where W^Q, W^K, W^V are weight matrices for the queries, keys and values, respectively. P is the position vector. $\sqrt{d_k}$ represents the scaling factor. Next, the upper cross-attention layer integrates the encoder's output ($\bar{X}$) with the output from the lower multi-head attention layer (Y_+) as follows:

$$\begin{aligned} &Q' = Y_+W^{Q'}, \quad K' = (\bar{X}+P_L)W^{K'}, \quad V' = (\bar{X}+P_L)W^{V'} \\ &Y' = \text{CrossAttention}(Q',K',V') = \text{softmax}\left(\frac{Q'K'^T}{\sqrt{d_k}}\right)V' \end{aligned} \tag{2}$$

where $W^{Q'}, W^{K'}, W^{V'}$ are new weight matrices. P_L is the position vector. Then, the output of the upper cross-attention layer, Y', is calculated as follows:

$$\begin{aligned} Y'' &= FFN(W_2\text{ReLU}(W_1Y' + b_1) + b_2) \\ Y'' &= \text{LayerNorm}(Y'' + Y_+) \\ Y'' &= \text{LayerNorm}(Y'' + Y') \\ Z &= \text{LayerNorm}(Y'' + FFN(Y'')) \end{aligned} \tag{3}$$

where W_1 and W_2 are weight matrices, b_1 and b_2 are bias terms, and ReLU is the activation function. Note that since Y remains constant during compression and some historical results can be reused, we can reduce redundant computation at each time step by pre-computing Y in advance and caching the historical compressed data.