Ransomware Detection through Temporal Correlation between Encryption and I/O Behavior

LIHUA GUO, Tsinghua University, China
YIWEI HOU, Tsinghua University, China
CHIJIN ZHOU*, Tsinghua University, China
QUAN ZHANG, Tsinghua University, China
YU JIANG*, Tsinghua University, China

In recent years, the increase in ransomware attacks has significantly impacted individuals and organizations. Many strategies have been proposed to detect ransomware's file disruption operation. However, they rely on certain assumptions that gradually fail in the face of evolving ransomware attacks, which use more stealthy encryption methods or benign-imitation-based I/O orchestration.

To mitigate this, we propose an approach to detect ransomware attacks through temporal correlation between encryption and I/O behaviors. Its key idea is that there is a strong temporal correlation inherent in ransomware's encryption and I/O behaviors. To disrupt files, ransomware must first read the file data from the disk into memory, encrypt it, and then write the encrypted data back to the disk. This process creates a pronounced temporal correlation between the computation load of the encryption operations and the size of the files being encrypted. Based on this invariant, we implement a prototype called RansomRadar and evaluate its effectiveness against 411 latest ransomware samples from 89 families. Experimental results show that it achieves a detection rate of 100.00% with 2.68% false alarms. Its F1-score is 96.03, higher than the existing detectors for 31.82 on average.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**.

Additional Key Words and Phrases: Ransomware Detection

ACM Reference Format:
Lihua Guo, Yiwei Hou, Chijin Zhou, Quan Zhang, and Yu Jiang. 2025. Ransomware Detection through Temporal Correlation between Encryption and I/O Behavior. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE010 (July 2025), 22 pages. https://doi.org/10.1145/3715725

1 Introduction

In recent years, a significant increase in the variety and frequency of ransomware attacks has occurred, which affects a wide range of organizations. In 2023, 66% of organizations were reported being compromised by ransomware attacks [1], with an average cost of recovery reaching $1.82 million [2]. These substantial economic losses and impacts on business operations have raised significant concerns in both academia and industry. The majority of ransomware attacks involve data encryption, which is used to hold victims' data hostage and demand ransom payments [3].

*Corresponding authors

Authors' Contact Information: Lihua Guo, Tsinghua University, BNRist, Beijing, China, glh22@mails.tsinghua.edu.cn; Yiwei Hou, Tsinghua University, BNRist, Beijing, China, houyw22@mails.tsinghua.edu.cn; Chijin Zhou, Tsinghua University, BNRist, Beijing, China, tlock.chijin@gmail.com; Quan Zhang, Tsinghua University, BNRist, Beijing, China, zhangq20@mails.tsinghua.edu.cn; Yu Jiang, Tsinghua University, BNRist, Beijing, China, jiangyu198964@126.com.

ACM 2994-970X/2025/7-ARTFSE010
https://doi.org/10.1145/3715725

As technical capabilities of cyberattacks have advanced, ransomware's disruption operations have become more covert and difficult to detect. New types of ransomware use customized encryption algorithms or mimic the I/O sequences of benign software [4] to hide their malicious activities and evade detection. As a result, traditional detection methods that monitor encryption behavior or specific I/O patterns are increasingly vulnerable to being bypassed. To address this issue, it is essential to identify an *invariant* in ransomware attacks. Regardless of how ransomware conceals its actions, this invariant remains unavoidable and can serve as a robust indicator for detection.

Insight. This paper introduces RansomRadar, a novel approach for detecting ransomware attacks by analyzing the temporal correlation between encryption and I/O behaviors. The key idea of RansomRadar is that there is a strong temporal correlation inherent in ransomware's encryption and I/O behaviors. This correlation stems from the essential file disruption operations employed by ransomware during its attack. To disrupt files, ransomware must first read the victim's file data from the disk into memory, encrypt it, and then write the encrypted data back to the disk. This process creates a pronounced temporal correlation between the computation load of the encryption operations and the size of the files being encrypted. Regardless of the specific encryption methods used or the sequence of I/O operations re-arranged by the ransomware, it cannot escape this behavioral pattern. Therefore, this pattern can be used as an invariant for detecting ransomware.

Challenges. To apply the idea in practice, the primary challenge lies in constructing the temporal correlation between encryption and I/O behaviors, which emerges from two main aspects.

First, *robust indicators* for encryption and I/O behaviors are lacking. When it comes to encryption behaviors, commonly used indicators such as system encryption APIs and entropy of written buffers fall short in the face of advanced ransomware. Empirical research [5] shows that only 6.22% of ransomware utilize encryption APIs, indicating that the majority of ransomware can successfully evade detection based on these APIs. Moreover, advanced ransomware [4] can reduce entropy by blending normal data with encrypted data, further complicating detection. In terms of I/O behaviors, commonly used indicators, such as the frequency of different operations, also prove to be insufficient. Existing research [4, 6] has shown that ransomware can mimic the I/O sequences of regular programs, rendering these characteristics indistinguishable from those of legitimate software.

Secondly, accurately measuring the *temporal correlation* between encryption and I/O behaviors is challenging. It is insufficient to simply confirm that these behaviors occur concurrently; it is imperative to verify that the encryption activities specifically target I/O file objects. An existing approach [7] attempts to detect ransomware based on sequences of encryption API invocations and I/O data. While this approach achieves a high detection rate, it yields a non-negligible false positive rate of over 5.80%. This issue frequently originates from legitimate applications such as browsers, which execute encryption and I/O operations concurrently, with the encryption focused on network data as opposed to file data. As a result, a more sophisticated model is necessary to construct and validate the temporal correlation between encryption and I/O behaviors effectively.

Solution. To address these challenges and apply our idea to practice, we make the following progress. Firstly, RansomRadar employs more robust indicators for encryption and I/O behaviors. Regarding encryption behavior, RansomRadar employs a hardware-based approach that utilizes distinct patterns exhibited in hardware performance counters (HPCs). These patterns stem from the intrinsic characteristics of encryption algorithms, which typically involve extensive loops and memory manipulations. This occurs regardless of the encryption implementation, whether through system APIs or customized code, making it a reliable indicator for encryption behavior. As a result, RansomRadar is capable of detecting various encryption operations employed by ransomware attacks. As for I/O behavior, RansomRadar quantifies it based on the size of files being processed following the "read-write" pattern and the number of basic I/O operations. Since the "read-encrypt-write" pattern is fundamental to ransomware attacks, reading files and then writing files form the invariant pattern in ransomware's I/O operations. This allows RansomRadar to quantify *the amount of potentially disrupted data*, regardless of ransomware's specific I/O orchestration.

Secondly, RansomRadar measures the temporal correlation between encryption and I/O operations using proposed indicators to improve detection accuracy. It first identifies the period during which ransomware involves encryption and then quantifies the computational load of encryption and I/O throughput at 100ms intervals, creating time-series data for these activities. RansomRadar then employs an LSTM model, trained on data from normal programs and ransomware, to assess the presence of temporal correlation. If such a correlation is detected, RansomRadar identifies it as a potential ransomware attack, particularly because the encryption targets I/O files. This comprehensive analysis of both encryption and I/O operations allows RansomRadar to distinguish between ransomware and legitimate programs that may also perform encryption or engage in intensive I/O activities, thereby improving detection rates and reducing false alarms.

Evaluation. We implemented RansomRadar [1] and evaluated its effectiveness against the latest ransomware variants. Our evaluation dataset was formed by collecting 411 samples from 89 latest ransomware families, and from benign program data comprising 1,266 processes. Benign data were obtained by monitoring users' daily usage under real-world scenarios. Compared to the evaluation of prior detection techniques [8, 9], which cover roughly 12 families, our dataset includes a more diverse set of ransomware samples. Experimental results show that it achieves a detection rate of 100.00% with 2.68% false alarms. Its F1-score reaches 96.03, higher for 1.73-84.97 compared to the state-of-the-art detection tools Unveil [9], Redemption [8], ShieldFS [10], PayBreak [11], and DeepWare [12]. In addition, we perform an ablation study to evaluate the importance of our temporal correlation analysis. The results show that it can reduce 83.65% false alarms. Finally, our experiment demonstrates that RansomRadar yields an overhead of 11.18%.

In summary, this paper makes the following contributions:

- We propose a novel approach to detect ransomware attacks through temporal correlation between encryption behaviors and I/O behaviors, which can address the challenges posed by the evolving ransomware attacks.
- We implement RansomRadar as a practical ransomware detection tool, which comprises four modules: System Monitor, Encryption Detector, IOTracker, and TCAnalyzer.
- We evaluate RansomRadar against the latest ransomware samples. Its F1-score reaches 96.03, higher than state-of-the-art detectors for 1.73-84.97.

2 Background

2.1 Ransomware Attacks

Ransomware attacks have become more and more frequent and of greater danger in the past years, causing great attention in both academia and industry. Existing ransomware can be divided into two categories: cryptographic ransomware and device lockers [13]. Cryptographic ransomware is designed to encrypt valuable files on a victim's computer, rendering them inaccessible without a decryption key, which is typically offered in exchange for a ransom [14–17]. This type of cyber extortion has evolved rapidly, leveraging sophisticated techniques to subvert data security. In this paper, for simplicity, we use the term "ransomware" to refer to cryptographic ransomware.

The encryption process in ransomware attacks commonly follows a systematic approach [18]. Initially, the ransomware reads the target files on the victim's system. Once read, these files are encrypted using robust cryptographic algorithms like AES, RSA, Chacha20 etc. [19] These may

[1] https://github.com/RansomRadar/RansomRadar.

be used by directly calling system APIs or implemented from scratch. Then, the encrypted data are written back to the system, often replacing the original file. This process ensures the original data is overwritten and inaccessible without the decryption key. In many cases, ransomware adds a unique file extension to the encrypted files, signaling the corruption of the file. The sophistication of modern encryption techniques poses significant challenges for ransomware defense [20–23].

2.2 Hardware Performance Counters

HPCs represent a critical component in modern computing systems, offering deep insights into the performance and behavior of hardware components. They are specialized registers within the CPU to store counts of various hardware-related activities. They can monitor specific hardware events, such as cache hits and misses, instruction execution, branch predictions, and many others, to help with system monitoring and performance analysis.

HPC data can be collected using various tools and APIs provided by the operating system or the processor manufacturers. For example, tools like perf [24] and PAPI [25] are commonly used on Linux systems. As for Windows systems, the collection of HPC data primarily employs built-in tools such as the Windows Performance Monitor and specialized tools like the Windows Performance Recorder (WPR) [26]. It is a component of the Windows Performance Toolkit that offers a robust and versatile solution for gathering and recording HPC data within the Windows environment. The fundamental steps in utilizing WPR involve initiating a recording session, collecting data, and saving data, where the collected information is stored in trace files for subsequent analysis. In addition, HPC values can reveal side-channel information [12]. For instance, LLCReference can be analyzed to infer data access patterns [27], potentially exposing vulnerabilities in cryptographic algorithms. Similarly, branch mispredicts can be used to deduce execution paths in the code [28].

2.3 I/O Request Packet

The I/O Request Packet (IRP) is pivotal in interacting with the operating system and the hardware devices. An IRP is a data structure used by operating systems, particularly in the Windows environment, to encapsulate information about an I/O operation. It is a crucial mechanism through which the system's I/O subsystem communicates with device drivers. The IRP contains details such as the type of I/O operation (read, write, control), pointers to data buffers, information about the file or device being accessed, and the completion status of the I/O operation. In the layered architecture of device drivers, the IRP operates at the lowest layer, directly interacting with the disk driver. This layer is critical because it translates high-level I/O requests from the system into specific commands that the disk hardware can understand and execute.

2.4 Threat Model

The threat model of this paper is identical to that of existing ransomware detection tools [8, 9, 29]. We assume that ransomware can launch attacks similar to other types of malware. It can employ various techniques to infect victims' systems. For example, a ransomware instance could be directly started by the user, delivered by a drive-by download attack, or installed via a malicious email attachment. It also can communicate with its C&C server to exchange encryption keys, identify valuable resources, and encrypt file data. We do not assume the techniques used by ransomware to encrypt files. Ransomware can utilize any encryption methods, including using system encryption APIs, static or dynamic linked encryption libraries, or customized encryption codes.

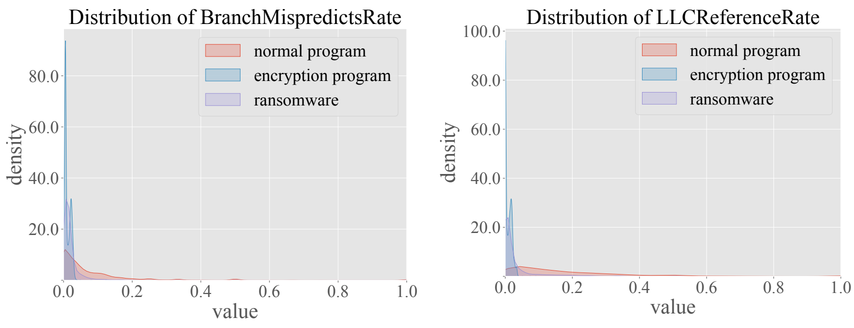
Furthermore, we assume that the OS kernel, underlying software, and hardware stack are secure and can be trusted. Our detection tool RansomRadar runs in a trusted environment, which is not affected by ransomware attacks. Attackers cannot execute malicious code with superuser privileges. This is a reasonable assumption because ransomware attacks typically occur in user mode.

3 Motivation

To apply our proposed idea into practice, we face two primary challenges: the absence of robust indicators for encryption behavior and the difficulty in constructing the temporal correlation between encryption and I/O behaviors. Through thorough analysis, we have identified two key observations that can address these challenges, which are using HPC as an encryption indicator and constructing a temporal correlation between encryption and I/O behaviors through HPC value and I/O throughput. These approaches leverage the unique capabilities of HPCs and I/O data to detect subtle yet consistent patterns associated with ransomware operations.

3.1 HPC as Encryption Indicator

We observe that encryption operations display distinct patterns in HPCs. Figure 1 shows the distributions of Branch Misprediction and Last-Level Cache (LLC) Reference rates across ransomware, normal programs, and encryption programs. The distributions for ransomware and encryption programs are similar but differ greatly from normal programs, indicating that ransomware behaves similarly to encryption programs, especially in data encryption. This phenomenon is due to the inherent structure of encryption algorithms, which involve loops and repeated memory access, leaving traces in branch and memory-related instructions. These traces can indicate encryption operations and allow us to quantify computational volume by analyzing instruction counts.



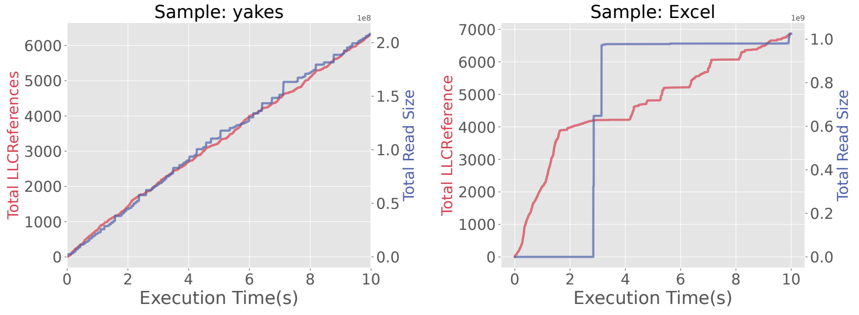
(a) *BranchMispredictsRate* (b) *LLCReferenceRate*

Fig. 1. Special pattern of HPCs in encryption operations.

3.2 Encryption-IO Temporal Correlation

During ransomware execution, we observe a strong temporal correlation between HPC values and I/O throughput, indicating the link between encryption and I/O behavior and helping identify processes targeting file data with encryption. I/O throughput is quantified by the size of files processed in a 'read-write' pattern, reflecting the encrypted data volume. Figure 2a and Figure 2b show this correlation between LLC Reference and I/O throughput for ransomware samples *yakes* and *Excel*, respectively. This high correlation, not commonly seen in normal programs, highlights ransomware's focus on file data. RansomRadar uses this correlation to detect ransomware attacks.

Since ransomware is characterized by intensive encryption targeting file data, RansomRadar detects it by identifying the temporal correlation between encryption and I/O behaviors. Specifically, RansomRadar first identifies all encryption operations in processes and flags them as suspicious based on their HPC features. It then quantifies these encryption and I/O behaviors and analyzes the temporal correlation between them, focusing on the 'read-encrypt-write' pattern indicative of data disruption typical in ransomware attacks. If this temporal correlation is detected, RansomRadar will classify the suspicious processes as ransomware.



(a) Ransomware Sample Yakes (b) Normal Program Excel

Fig. 2. Temporal correlation between HPC values and I/O behaviors in ransomware and normal programs.

4 Design

The overall workflow of RansomRadar is illustrated in Figure 3. It includes four modules: (1) *System Monitor* to collect hardware and I/O data during runtime; (2) *Encryption Detector* to detect processes' encryption behaviors based on hardware data; (3) *IOTracker* to calculate the I/O throughput of suspicious processes; (4) *TCAnalyzer* to analyze the temporal correlation between a process's encryption and I/O behaviors. Once ransomware starts encrypting the data on a system enhanced with RansomRadar, all of its behavior will be firstly captured by the *System Monitor*. Then, based on the HPCs values collected by RansomRadar, *Encryption Detector* will identify its encryption behavior, mark it as suspicious, and report its pid to *IOTracker* and *TCAnalyzer*. *IOTracker* will then calculate the I/O throughput of the suspicious process and pass the results to *TCAnalyzer*. Finally, *TCAnalyzer* will detect the ransomware by comprehensively analyzing the hardware data and I/O throughput of this suspicious process and capturing their correlation.

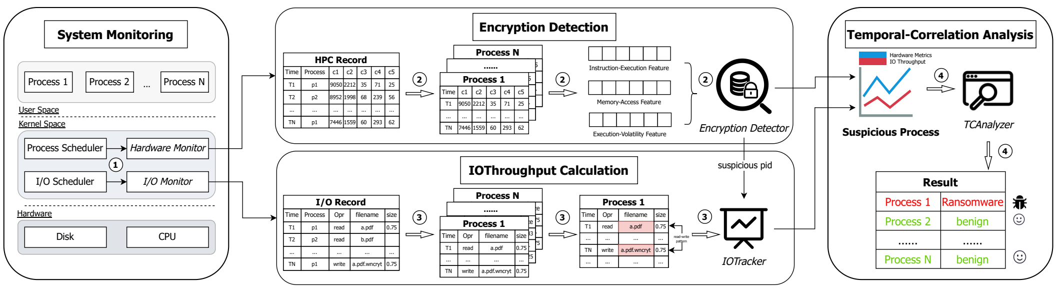


Fig. 3. Overall workflow of RansomRadar. *System Moniter* gathers hardware and I/O data in real-time. *Encryption Detector* detects if a process is doing encryption. *IOTracker* calculates I/O throughput of a detected process performing encryption. *TCAnalyzer* examines the temporal correlation between hardware data and I/O throughput to reach a final verdict. In the HPC Record, c1-c5 denotes five selected PMC data.

4.1 System Monitor

The goal of *System Monitor* is to monitor and collect behavioral data of processes. It comprises two submodules: *Hardware Monitor* to collect hardware data and *I/O Monitor* to collect I/O data.

Hardware Monitor captures critical hardware data during runtime for encryption detection. It continuously collects data without user processes being aware of it. To accomplish this, we leverage the capabilities of Performance Monitoring Units (PMU) and select five representative counters for their relevance to instruction execution and memory access. The counters are shown in Table 1. *InstructionsRetiredFixed*, *BranchInstructionRetired* and *BranchMispredictsRetired* are pertinent to instructions execution, while *LLCReference* and *LLCMisses* are pertinent to memory access. *Hardware Monitor* collects HPC values by sampling mode and aggregates them every 100ms. These data reflect the state of loop execution and memory access, which will exhibit distinctive patterns when running encryption operations. Thus, this serves as a crucial tool for encryption detection.

Table 1. Selected PMC and their meanings.

PMC	**Meaning**
InstructionsRetiredFixed	Executed instructions.
BranchInstructionRetired	Executed branch instructions.
BranchMispredictsRetired	Branch mispredictions.
LLCReference	References to the last level cache.
LLCMisses	Reference misses to the last level cache.

I/O Monitor is to observe processes' I/O operations for further calculation of their I/O throughput. It monitors the filesystem activities at the lowest possible layer to ensure that no actions are overlooked so that malicious programs cannot circumvent detection through any means. To accomplish this, we implement a driver in the kernel that intercepts and sets callbacks for all I/O requests directed to the filesystem. This serves as the foundation for the subsequent calculation of the I/O throughput of any suspicious process.

4.2 Encryption Detector

Encryption Detector serves as the initial stage in the overall ransomware detection process of RansomRadar. It detects all potential encryption behaviors on the system and marks the corresponding process as suspicious. To achieve this, we leverage the distinct pattern exhibited by the properties of encryption algorithms in hardware data. We first introduce three types of features derived from hardware data, which are used to differentiate between encryption programs and normal programs. Then we apply a machine learning model to detect the encryption behaviors.

First, we propose three types of features derived from the gathered hardware data. These include features based on **instruction-execution**, **memory-access**, and **execution-volatility**. The features are calculated every second, and their definitions are as follows. Consider one process's data in time interval $(T_i s, T_i + 1s)$. Since the hardware metrics are aggregated every 100ms, this process has ten records in this time interval.

$$\{(\#Ins_i, \#BI_i, \#BM_i, \#LR_i, \#LM_i)\}, 1 \leq i \leq 10 \tag{1}$$

Here, i denotes the i-th 100ms sub-interval, $\#Ins_i$ denotes total instructions, $\#BI_i$ denotes total branch instructions, $\#BM_i$ denotes total branch mispredicts, $\#LR_i$ denotes total LLCReferences, and $\#LM_i$ denotes total LLCMisses.

For **instruction-execution**, encryption algorithms usually involve extensive loops. Hence, the proportion of branch instructions and mis-predicts may be different from normal programs and can serve as two distinctive features for encryption operations. The definitions of them are as follows:

$$BranchInstructionRate = \frac{1}{n} \sum_i \frac{\#BI_i}{\#Ins_i} \tag{2}$$

$$BranchMispredictsRate = \frac{1}{n} \sum_i \frac{\#BM_i}{\#BI_i} \tag{3}$$

For **memory-access**, we leverage the fact that the encryption algorithm has frequent manipulation of the same memory areas, which significantly influences the cache reference. Therefore, we propose two features that measure the rate of LLCReference and the rate of cache miss while referencing. Their definitions are shown as follows:

$$LLCReferenceRate = \frac{1}{n}\sum_i \frac{\#LR_i}{\#Ins_i} \tag{4}$$

$$LLCMissRate = \frac{1}{n}\sum_i \frac{\#LM_i}{\#LR_i} \tag{5}$$

For **execution-volatility**, we assume that when a process engages in massive encryption operations, it will repeatedly execute identical code pieces. This repetition reduces the volatility of instruction-execution and memory-access features. Therefore, we propose four features to measure the standard deviation of the aforementioned features within 1s. The definitions are as follows:

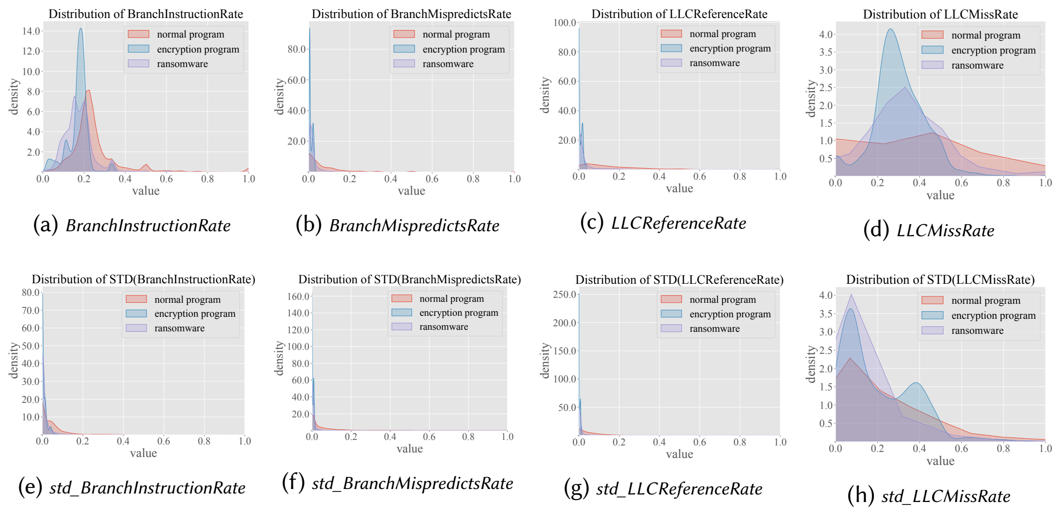
$$std\_BranchInstructionRate = std(\frac{\#BI_i}{\#Ins_i}) \tag{6}$$

$$std\_BranchMispredictsRate = std(\frac{\#BM_i}{\#BI_i}) \tag{7}$$

$$std\_LLCReferenceRate = std(\frac{\#LR_i}{\#Ins_i}) \tag{8}$$

$$std\_LLCMissRate = std(\frac{\#LM_i}{\#LR_i}) \tag{9}$$

We conduct a preliminary analysis of their distributions to demonstrate that these features can be utilized for encryption detection. As shown in Figure 4, we demonstrate the distribution density function of each feature between ransomware, encryption programs, and normal programs and analyze the potential reasons behind these distributions from the perspective of the characteristics of encryption algorithms. For Figure 4a, *BranchInstructionRate* is lower since the processor may



(a) *BranchInstructionRate* (b) *BranchMispredictsRate* (c) *LLCReferenceRate* (d) *LLCMissRate*

(e) *std_BranchInstructionRate* (f) *std_BranchMispredictsRate* (g) *std_LLCReferenceRate* (h) *std_LLCMissRate*

Fig. 4. Proposed features and their distributions between normal programs, encryption programs, and ransomware.

expand the encryption loop body to decrease branch instructions and increase running efficiency; for Figure 4b, *BranchMispredictsRate* is lower since it frequently executes the same pieces of code, which increases the precision of the processor's branch prediction; for Figure 4c, *LLCReferenceRate* is lower due to its frequent access to the same memory areas, which can be done in a higher-level cache; for Figure 4d, *LLCMissRate* is lower due to its frequent access to the same memory areas and the locality of reference causes the low rate in LLCMiss; for Figure 4e, 4f, 4g, 4h, these features are lower since its repeated execution of the similar code pieces. These results show that the distribution of ransomware and encryption programs is closely aligned, and markedly distinct from those of normal programs. These suggest the feasibility of using these features to detect encryption activities. Moreover, this observation allows us to infer that a significant amount of encryption activity occurs during ransomware operations. To conclude, these features enable the identification of potentially suspicious processes performing encryption.

Second, to comprehensively capture these distinct patterns exhibited by encryption algorithms, we employ a machine learning model to distinguish encryption behavior from other activities. We gather famous encryption libraries written in various programming languages, including CryptoAPI [30], OpenSSL [31], Cryptography [32], and others. Subsequently, we utilize these libraries to develop encryption programs, implementing various encryption algorithms frequently used in ransomware, including AES, Chacha20, Xsalsa, etc. Then, we collect hardware data from these encryption and normal programs and utilize this data to train an Encryption Detector. We choose the K-Nearest Neighbors (KNN) model as the classification framework. Before training, we also conducted two pre-processing steps on the training dataset.

(1) Data Balancing.
Since the number of encryption programs is limited, we first balance the data between encryption programs and normal programs. We employ the Synthetic Minority Oversampling Technique (SMOTE) to oversample the samples of encryption programs to achieve balance.
(2) Standard Normalization.
Since the scale of the data varies across different types, we standardize the data before training and judging using Min-Max normalization:

$$feature_i = \frac{feature_i - \min(feature_i)}{\max(feature_i) - \min(feature_i)} \tag{10}$$

We choose $k = 6$ based on the experiment and train the KNN model for encryption detection.

Ultimately, the trained *Encryption Detector* will evaluate each running process in the system and utilize the gathered hardware data to pinpoint those exhibiting encryption behaviors. Processes identified as engaging in encryption behaviors are labeled as suspicious and subsequently subjected to temporal correlation analysis in the next phase.

4.3 IOTracker

For any process identified as suspicious due to its engagement in encryption behavior, *IOTracker* will quantify its I/O behavior with our proposed I/O throughput for further analysis of the temporal correlation between them. This section will explain the definition and calculation of I/O throughput.

In RansomRadar, I/O throughput measures how much data is potentially encrypted and how many core I/O operations have been taken by suspicious processes. Due to the nature of ransomware as a category of encryption-intensive malware targeting files [33], its execution instructions are intricately linked with the volume of data it processes and its I/O operations. Therefore, I/O throughput calculates these values and serves as input to *TCAnalyzer*.

We calculate the size of files being processed following the 'read-write' pattern to explore how much data is potentially encrypted. During ransomware attacks, this metric can represent the amount of encrypted data, which affects the amount of processing required by the encryption process since the 'read-write' pattern is invariant to the file disruption behavior regardless of the I/O pattern used by ransomware. Previous studies [9, 34] have summarized three patterns utilized by ransomware during the disruption of file data, which are *overwrite*, *delete and rewrite* and *smash and rewrite*. In *overwrite* pattern, ransomware first reads the data from the file and then writes the encrypted data back. In *delete and rewrite* pattern, ransomware first reads the data from the file, deletes the original file, and then writes the encrypted data to a new file with a similar name to the original one. In *smash and rewrite* pattern, ransomware first reads the data from the file, overwrites the original file with random data, deletes the original file, and then writes the encrypted data to a new file with a similar name to the original one. Although these patterns vary in detail, we discern a typical operational sequence: a 'read-write' series of actions performed on disrupted files. And, the files receiving encrypted data typically retain the original file name or have the same basename with an added unique extension. This lets us ascertain which files are potentially encrypted by observing whether they are initially read and subsequently written. We can get the volume of data encrypted by ransomware based on the file size at the initial read operation. This serves as the size metric in our proposed I/O throughput. In addition, since different basic I/O operations require specific instructions to be completed, we also consider them: read, write, create, delete, and rename. I/O throughput is calculated every 100ms for each suspicious process.

4.4 TCAnalyzer

Upon completing the above modules, we can identify the processes that undertake encryption operations during specific time intervals and calculate their I/O throughput within these durations. The objective of *TCAnalyzer* is to ascertain the existence of a temporal correlation between hardware data and I/O throughput during these intervals. If the temporal correlation is found, it implies that these encryption operations are predominantly aimed at file data, and *TCAnalyzer* will classify the process as ransomware.

Temporal Correlation Analysis Model. We apply a deep learning method to capture the temporal correlation between hardware data and I/O throughput. We use a long short-term memory network (LSTM), a variant of recurrent neural networks uniquely tailored for discerning data dependencies in sequential datasets. Our analysis is carried out every second. The hardware data fed into the model comprises values from each PMC recorded at 100ms intervals, culminating in a 10×5 matrix of time series of data. As for the I/O throughput data, calculated by the *IOTracker*, it manifests as a 10×7 matrix of time series data.

Training Process. To train the *TCAnalyzer* model, we initially collect the hardware data and I/O data from both normal programs and ransomware. Ransomware data is obtained with a unified structure by executing in a customized virtual environment. This environment features a user's desktop folder embedded with a directory housing 1,000 subfiles to confirm that ransomware has conducted the disruption attack. Then, we preserve only those ransomware processes. Finally, we merge the refined hardware data with the I/O throughput to compose our dataset, which then serves as the dataset for constructing the Temporal Correlation Analysis Model.

5 Implementation

We implemented the prototype of RansomRadar on Win 10 system, which includes four critical modules: *System Monitor*, *Encryption Detector*, *IOTracker* and *TCAnalyzer*. These modules work in tandem to provide comprehensive monitoring, detection, and analysis capabilities for the system.

System Monitor serves as the data monitoring module and operates at the kernel of the operating system. It is composed of two primary components: the *Hardware Monitor* and the *I/O monitor*. The *Hardware Monitor* leverages Event Tracing for Windows (ETW) [35], an advanced Windows-based tracing framework that captures and logs a wide array of hardware performance metrics. It allows RansomRadar to capture and record the selected hardware performance counter values. The *I/O Monitor* is implemented as a minifilter driver embedded within the Windows Kernel. The driver intercepts all I/O requests that transition from user space to the disk driver then decodes the IRP and collects the information of each I/O operation, including the process ID, process name, operation type, filename, buffer size, etc. *Encryption Detector* is developed using a K-Nearest Neighbours (KNN) model, which is implemented using the Python sci-kit-learn library [36] and configured with a parameter of $k = 6$. *IOTracker* is implemented in Python script to dissect the I/O data. Lastly, *TCAnalyzer* employs a Long Short-Term Memory (LSTM) model implemented using Pytorch [37] and configured with $hidden\_size = 50$ and $hidden\_layer = 1$.

6 Evaluation

Our evaluation addresses research questions as follows:

- **RQ1**: What's the effectiveness of RansomRadar in detecting ransomware?
- **RQ2**: What is the runtime overhead of RansomRadar?
- **RQ3**: What's the effectiveness of *TCAnalyzer* and *Encryption Detector*?
- **RQ4**: What's the impact of model selection in RansomRadar?

Dataset. We collect 411 latest ransomware samples from 89 families, including a broad spectrum of well-known families, such as LockBit, REvil, Ryuk, and WannaCry. Sources of these samples include malware analysis platforms [38], cybersecurity forums [39–43], underground hacker forums [44]. Compared to the evaluation of prior detection techniques [8, 9], which covers roughly 12 families, our dataset includes a more diverse and comprehensive set of ransomware samples.

For benign programs' data, we deploy RansomRadar on 3 different machines and collect benign data during users' daily usage under real scenarios. This data contains 1,266 benign processes including browsers, office suites, compilers, etc.

For encryption programs, we collect famous encryption libraries such as CryptoAPI, OpenSSL, and Cryptography as well as 20 encryption codes custom-developed by programmers from GitHub with high stars [45–64]. We utilize them to build encryption programs implementing several encryption algorithms commonly used by ransomware, including AES, Chacha20, Xsalsa, and RC4. These encryption programs were executed for encrypting 250MB of data for hardware data collection.

For *Encryption Detector*, RansomRadar will first utilize HPC data from both benign and encryption programs to train a KNN model. For *TCAnalyzer*, it will transform HPC and I/O data from both ransomware and benign programs into time-series data to train the LSTM model. At each stage, the data is split into a training set and a testing set to evaluate the effectiveness of the method.

Experiment Setup. All the experiments were conducted on a machine equipped with an i7-1165G7 running Windows 10 system. We utilize the Microsoft Hyper-V to construct a virtual environment operating on Windows 10, version 21H2, with 4 CPU cores and 8GB RAM. *Hardware Monitor* and *I/O Monitor* utilize the virtualization mechanism provided by Hyper-V to collect hardware and I/O data. In the virtual environment, a particular folder contains 1,000 random files of various types, including TXT, Word, PowerPoint, Excel, and others. These files are collected from real-world users' computers to simulate a typical user environment but do not contain any sensitive personal data. In addition, to facilitate the successful execution of ransomware samples, anti-virus software within the machine is disabled during execution. Concurrently, unrestricted internet access is provided to the operating system, enabling the samples to establish communication with their respective remote servers as required.

6.1 Effectiveness of Ransomware Detection

This section evaluates the effectiveness of RansomRadar against ransomware attacks. First, we evaluate its overall effectiveness against all the ransomware samples. Then, we evaluate its effectiveness in detecting ransomware samples from unknown families. Finally, we give one case study to show that RansomRadar can solve the challenges brought by new types of ransomware attacks.

Overall Effectiveness. In this experiment, we apply a 10-fold cross-validation method to evaluate the effectiveness. Each time we remove 10% sample data from the training set to train the detection model and evaluate the testing set with the model. We select five state-of-the-art detection tools for comparison: PayBreak [11], Unveil [9], Redemption [8], ShieldFS [10], and DeepWare [12], as summarized in Table 2. PayBreak detects ransomware by monitoring the usage of encryption APIs. Unveil identifies ransomware by monitoring I/O sequences through predefined rules. Redemption and ShieldFS detect ransomware by monitoring I/O operations, extracting features, and applying a machine-learning model for detection. DeepWare identifies ransomware through its distinct characteristics in HPC values.

Table 2. Existing Ransomware Detection Methods.

	T	E	I	TC	API	Entropy	IRP	HPC
PayBreak [11]	R	✔			✔			
Unveil [9]	R	✔	✔			✔	✔	
Redemption [8]	M	✔	✔			✔	✔	
ShieldFS [10]	M	✔	✔			✔	✔	
Deepware [12]	M							✔
RansomRadar	M	✔	✔	✔			✔	✔

T: Type (R: rule-based, M: machine learning based), **E**: involves Encryption detection, **I**: involves I/O detection, **TC**: involves Temporal Correlation analysis between encryption and I/O behaviors.

The detection result of each method is shown in Table 3. RansomRadar achieves a detection rate of 100.00% with 2.68% false alarms. Compared with other tools, RansomRadar achieves an F1-score of 96.03, which is higher than others for 1.73-84.97. Results show that other tools either have a low recall rate or a low precision rate. PayBreak, Redemption, and Unveil have a lower recall rate since they rely on some assumptions on ransomware for detection. They assume that ransomware employs system crypto API to do encryption or behave distinctly differently from normal programs on I/O patterns. However, this gradually fails for the evolving ransomware that uses self-implemented encryption code or imitation-based I/O orchestration, leading to many false negatives. Deepware and ShieldFS achieve a high recall, but their precision is relatively low. Deepware assumes that ransomware will exhibit a distinct pattern on HPCs but it overlooks that normal programs with encryption behavior will trigger a similar pattern, thus triggering false positives. ShieldFS assumes that ransomware's I/O is distinctly different from benign programs, but some I/O-intensive programs may be misclassified since the boundary between them and ransomware is blurred. In our method, we jointly utilize the distinct HPC patterns and the temporal correlation between HPC and I/O data to confirm the encryption behavior toward file data. Therefore, RansomRadar can counter evolving ransomware attacks effectively with lower false alarms.

We also present the detection effectiveness of RansomRadar for each of the two stages. For encryption detection, 100.00% ransomware samples are identified as exhibiting encryption behavior. Meanwhile, 16.43% of processes from normal programs are identified as suspicious due to encryption behavior. However, since encryption is necessary in some normal operations, these instances cannot

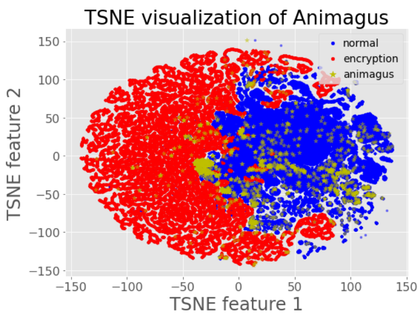
Table 3. Detection Result of All Samples.

Tools	Accuracy(%)	Precision(%)	Recall(%)	F1-score
PayBreak	23.07	50.00	6.22	11.06
Redemption	78.56	95.37	31.21	47.03
ShieldFS	81.25	81.10	100.00	89.57
Unveil	80.57	99.05	65.82	79.09
DeepWare	90.76	90.95	97.90	94.30
RansomRadar	**97.97**	**92.36**	**100.00**	**96.03**

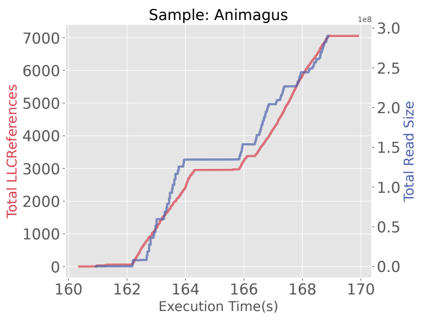
be labeled as ransomware at this stage. These processes are then forwarded to the *TCAnalyzer* for further confirmation. In the temporal correlation analysis phase, *TCAnalyzer* achieves an accuracy of 96.09%, with a recall rate of 100.00%. With this two-stage detection strategy, RansomRadar can detect 100.00% of the malicious processes of ransomware samples with 2.68% false alarms on normal programs.

Detection of Unknown Family. We also evaluate the effectiveness of RansomRadar against ransomware from unknown families. This experiment ensures that there are no similar ransomware samples in the training set. To conduct this, we removed all the samples from one family each time and trained the detection model on the reduced dataset. Then we report the detection result on the test set. RansomRadar still achieves a high detection rate against ransomware samples of unknown families with recall rate of 99.76%, which means that it has the generalization ability towards unknown ransomware families.

Case Study. To demonstrate that RansomRadar can solve the challenges brought by new types of ransomware attacks, we analyze how RansomRadar can detect Animagus [4]. Animagus is a new type of attack evading existing detection methods through two approaches. First, it uses self-implemented encryption code to achieve data encryption and mixes the original and encrypted data in the written back data. Thus, existing API-based and entropy-based methods fail to detect its encryption behavior. Second, it uses an imitation-based method to orchestrate its I/O operations. Thus, it behaves like normal programs and evades existing I/O detection. However, it still left a malicious footprint in the features proposed by RansomRadar. We visualize them in Figure 5.



(a) Hardware features



(b) Temporal correlation

Fig. 5. Visualization of Animagus' behaviors.

Figure 5a demonstrates the distribution of hardware features of Animagus after dimensionality reduction. The result shows that many of the data points of Animagus lie in the area associated with encryption programs. This indicates that Animagus is performing encryption behavior during its execution and is marked as suspicious by *Encryption Detector*. Figure 5b displays the fluctuations in Animagus' hardware and I/O data over time. The figure shows an obvious temporal correlation between hardware and I/O data, indicating that Animagus is possibly performing operations toward file data. These observations led RansomRadar to determine that Animagus uses encryption to target file data and categorizes it as ransomware.

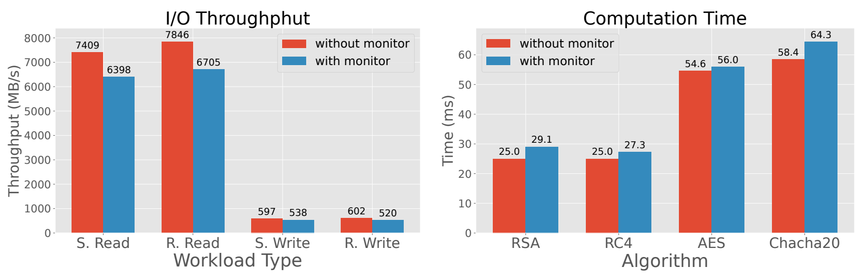
Answer to RQ1: RansomRadar achieves a detection rate of 100.00% against the latest ransomware samples with 2.68% false alarms. Compared to existing works, it achieves an F1-score of 96.02, which is higher than others for 1.73-84.97. This indicates that it has the capability to counter the evolving ransomware attacks.

6.2 Overhead of RansomRadar

Since RansomRadar monitors the system's I/O data and hardware instructions, in this experiment, we separately evaluate its impact on I/O-intensive and computationally-intensive programs during runtime to show that it has a low overhead for normal users.

For I/O-intensive programs, we utilize DISKSPD [65], a storage performance tool from Windows, to measure the impact of RansomRadar on the throughput of I/O operations. We conduct this evaluation across four distinct categories of workloads: Sequential Read, Random Read, Sequential Write, and Random Write. Each workload runs for 100 seconds to ensure a stable measurement of I/O throughput. Results are shown in Figure 6a. By capturing only the basic I/O operations at the system's lowest level, we observe a 12.92% overhead on the I/O throughput.

For computationally intensive programs, we measure the overhead introduced by RansomRadar on encryption tasks. We employed OpenSSL, a widely recognized encryption suite to evaluate the impact of RansomRadar on its encryption speed. The algorithms we tested include RC4, AES, RSA, and Chacha20, which are widely used by ransomware. Among them, RC4, AES, and Chacha20 were tested under the task of encrypting a randomly generated 1MB file, and RSA was tested under the task of singing a randomly generated 256-bit file. We ran each experiment ten times and recorded their average time. The results are shown in Figure 6b. Since we monitor and capture HPCs through ETW, it will not interrupt the user's process during execution and yields an overhead of 9.46%.



(a) Impact on I/O (b) Impact on Compute

Fig. 6. Overhead of RansomRadar.

Answer to RQ2: During runtime, RansomRadar incurs a low overhead of 12.92% for I/O intensive programs and 9.46% for compute-intensive programs.

6.3 Component Evaluation

In this section, we separately evaluate the effectiveness of *TCAnalyzer* and *Encryption Detector*. For *TCAnalyzer*, we evaluate its effectiveness in detecting temporal correlation between encryption and I/O operation and show that it can raise the precision of ransomware detection. For *Encryption Detector*, we evaluate its effectiveness in detecting the self-implemented encryption operations.

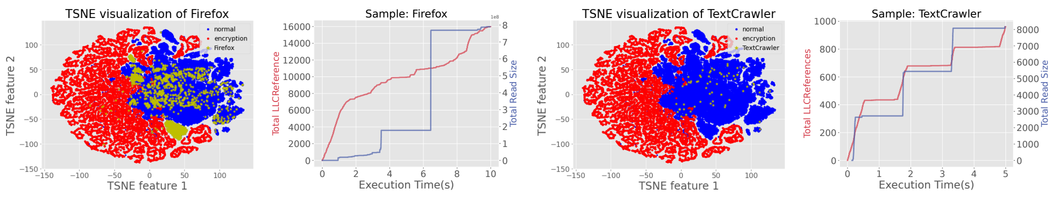
Effectiveness of *TCAnalyzer*. We first conduct an ablation study to evaluate the effectiveness of *TCAnalyzer*. We remove *TCAnalyzer* from RansomRadar to construct RansomRadar $^{TC-}$ and evaluate it against ransomware. In RansomRadar $^{TC-}$, the suspicious processes identified by *Encryption Detector* will be marked as ransomware. The detection result is shown in Table 4.

Table 4. Detection result of RansomRadar $^{TC-}$.

Tools	Accuracy(%)	Precision(%)	Recall(%)	F1-score
RansomRadar	97.97	92.36	100.00	96.03
RansomRadar $^{TC-}$	87.60	66.40	100.00	79.81

This result demonstrates the ability of *TCAnalyzer* in two aspects. First, it can capture the temporal correlation between encryption operations and I/O operations in ransomware attacks. The recall rate of RansomRadar $^{TC-}$ means that 100.00% of the ransomware samples were identified as performing encryption operations. The same recall rate of RansomRadar means that the temporal correlation between encryption and I/O operation was successfully detected in all these samples. Thus, it proves the ability of *TCAnalyzer* to capture the temporal correlation. Second, it can significantly reduce the false positives of RansomRadar. In RansomRadar $^{TC-}$, the precision is only 66.40% since many benign programs perform encryption operations misclassified as malicious. But the precision raises to 92.36% through *TCAnalyzer*, which means that it can filter out the processes that perform encryption behavior not towards file data, thus decreasing the false alarms.

Case Study. Here, we demonstrate an example of how *TCAnalyzer* can remove false positives of Encryption Behavior from Normal Programs. We use the widely used web browser Firefox as a case study. Firefox engages in encryption activities as part of its routine operations, primarily during online communications that involve encrypted protocols. We demonstrate its hardware features' distribution and association between hardware data and I/O data in Figure 7. As shown in Figure 7a, during the execution, some data points fall into the area associated with encryption programs. These will be detected as performing encryption behavior and marked as suspicious by *Encryption Detector*. However, in the temporal correlation analysis, as shown in Figure 7b, a negligible temporal correlation is observed between the hardware and I/O data. This indicates that the detected encryption operations are not towards file data, so Firefox is not marked as ransomware by *TCAnalyzer*. This illustrates that *TCAnalyzer* can eliminate the false positives of normal programs with encryption behaviors and improve the precision of ransomware detection.



(a) Hardware features (b) Temporal correlation (c) Hardware features (d) Temporal correlation

Fig. 7. Visualization of Firefox' and TextCrawler's behaviors.

Effectiveness of *Encryption Detector.* *Encryption Detector* serves as the first detection component of RansomRadar. Its primary function is to identify the encryption behaviors in processes and mark them as suspicious. The effectiveness of this detector is critical for overall performance for two reasons. First, if *Encryption Detector* has high accuracy in encryption detection, it can filter out normal processes not engaging in encryption. Second, the ability of *Encryption Detector* to detect encryption behavior in unknown implementation will influence the capability to identify the malicious processes since it decides which processes will be forwarded to the *TCAnalyzer*. Therefore, we evaluate the effectiveness of *Encryption Detector* from these two perspectives.

Overall Detection Effectiveness. We first evaluate the overall effectiveness of *Encryption Detector*. We label the data from encryption programs as positive and the data from normal programs as negative. In the experiments, we employ 10-fold cross-validation to calculate the metrics for binary classification. Experimental results are shown in Table 5, demonstrating that the *Encryption Detector* performs well in classification between encryption and normal programs.

Table 5. Overall Effectiveness of Encryption Detector.

Metric	**Average**(%)	**Min**(%)	**Max**(%)
Accuracy	99.60	99.52	99.66
Precision	99.24	99.08	99.66
Recall	99.96	99.91	99.98
F1-score	99.60	99.52	99.66

We also apply a T-SNE dimension reduction to show the different distributions between encryption and normal programs. As shown in Figure 8, a potential non-linear classification boundary exists between encryption and normal programs, which can explain the effectiveness of the *Encryption Detector*. Moreover, *Encryption Detector* filters out 83.57% of the normal processes that are not performing encryption behavior, significantly decreasing the workload for *TCAnalyzer*.

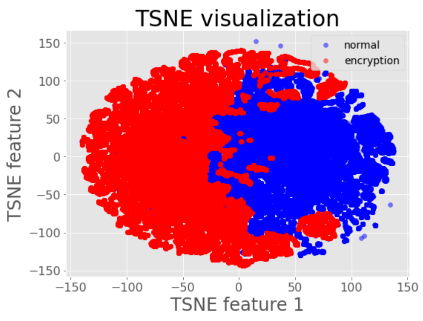


Fig. 8. TSNE visualization of hardware features for encryption programs and normal programs.

Effectiveness of Detecting Unknown Implementations. With the advancement of static and dynamic detection technologies, ransomware using existing encryption APIs is easily captured. Therefore, more ransomware samples have begun incorporating self-implemented encryption codes to carry out more stealthy encryption operations. So, the ability to capture the unique characteristics of encryption algorithms, irrespective of their implementation, is essential in ransomware detection. In this experiment, we evaluate whether *Encryption Detector* can capture the unique characteristics of encryption algorithms by assessing its effectiveness in dealing with unknown implementations of different encryption algorithms. To achieve this, we remove the implementation of one algorithm from the training set and train the classification model. Subsequently, we use the trained model to perform encryption detection. Results are shown in Table 6.

Table 6. Effectiveness in detecting unknown implementations of encryption algorithms.

	C/C++	Rust	Go	Python
Recall	88.85%	98.03%	94.68%	93.33%

We demonstrate the recall rate for various implementations across different programming languages. The average recall rate reaches 93.72%, indicating that the *Encryption Detector* can detect these unseen algorithm implementations during training. This generalized detection capability greatly assists in identifying ransomware that utilizes self-implemented encryption codes.

Case Study. In this case, we demonstrate how *Encryption Detector* filters out normal programs not engaging in encryption operations and reduces the false alarms for normal programs with intensive file operations. In this experiment, we analyze TextCrawler, a text replacement software that scans all files in a specified folder, reads their content, identifies specific strings, and performs replacements. We simulate its usage by replacing the names of header files in all the '.c' files in a project. This results in numerous 'read-write' patterns in its I/O sequence, which is a suspicious pattern that can be exploited by ransomware to achieve data disruption. At the same time, we found that the entropy of each written operation reaches 0.99, which is much higher than other normal programs, with an average of 0.83. Combining the two facts above, it is hard to make correct classification with existing detection methods. In RansomRadar, if we just use *TCAnalyzer*, it is classified as ransomware since it shows a strong temporal correlation between the hardware data and I/O data. However, as shown in Figure 7c, its hardware features all lie in the area associated with normal programs, and *Encryption Detector* doesn't classify it as performing encryption behavior. So, RansomRadar can accurately classify it as benign. This case shows how RansomRadar eliminates these false positives with encryption detection and explains the necessity of *Encryption Detector*.

Answer to RQ3: For *TCAnalyzer*, it can capture the temporal correlation between encryption behavior and I/O behavior during ransomware attacks, and it can significantly decrease the false alarms of the system. For *Encryption Detector*, it can accurately detect encryption behavior in various implementations.

6.4 Model Selection of RansomRadar

For *Encryption Detector* and *TCAnalyzer*, we tested different models and chose the best for RansomRadar according to their effectiveness and efficiency. For *Encryption Detector*, we tested four commonly-used classifiers and evaluated their accuracy: KNN (99.60%), Decision Tree (99.57%), SVM (96.72%), and Logistic Regression (90.22%). We chose KNN for its best effectiveness. For TCAnalyzer, we tested LSTM and Transformer models. Since LSTM achieved a 100.00% detection rate and its inference is 3.1x faster than Transformer's, so we chose LSTM.

Answer to RQ4: We choose KNN for *Encryption Detector* and LSTM for *TCAnalyzer* for their effectiveness and efficiency in detection.

7 Discussion

Ethic Considerations. Ethics is a top priority when we collect samples and conduct experiments. The ransomware samples we used were collected from malware analysis platforms, cybersecurity forums, etc., and they were used for our academic study only. All experiments were conducted in a virtual environment. To comprehensively capture the behaviors of ransomware, we permitted ransomware's network connection to its command and control (C&C) servers. To mitigate the risk of ransomware's lateral movement, we isolated file transfers between the host and guest machines and deployed antivirus software on the host machine to monitor any malicious activities.

Limitations. In this work, we focus exclusively on crypto-ransomware, which exhibits a strong temporal correlation between its encryption and I/O behavior during an attack. The defenses for other types of ransomware, such as Locker, Scareware, or Extortionware, are orthogonal to this work and are not addressed in this study. Furthermore, our focus is solely on dynamic detection, which can only be conducted after ransomware has carried out its encryption activities. Upon detection, backup techniques like FlashGuard [14] and RSSD [16] can be applied to roll back the compromised data, mitigating the impact of the attack.

Potential Evasion Techniques. There are three potential evasion techniques for RansomRadar. First, RansomRadar cannot counter ransomware that does not rely on file encryption. RansomRadar assumes that all I/O operations of ransomware will be completed through the kernel's I/O scheduler, meaning that the Minifilter Driver can capture them. On the contrary, if ransomware attacks operate with the highest permission and directly manipulate the disk, such as directly deleting system shadow copies, which won't generate I/O records, RansomRadar cannot detect it. This requires a more thorough monitoring and analysis of ransomware's behaviors. Second, RansomRadar might fail against ransomware that imitates benign processes at the instruction level by employing code obfuscation techniques to disrupt encryption patterns. However, this approach faces inherent constraints as encryption operations inherently require extensive loops and memory manipulations. Third, ransomware could potentially evade detection by disrupting the temporal correlation between encryption and I/O behaviors through delayed encryption. However, analysis from Animagus [4] demonstrates that such evasion remains challenging, as benign-imitation-based attacks maintain detectable temporal correlations between its encryption and I/O behaviors.

8 Related Work

8.1 Encryption Detection of Ransomware

Encryption is fundamental to cryptographic ransomware attacks, where the primary goal is to encrypt victims' files efficiently and covertly. [66]. Therefore, identifying encryption activity is crucial in detecting ransomware. Extensive research has been conducted on it [67–69].

One effective strategy involves monitoring encryption APIs through dynamic and static analysis. Dynamic analysis examines API sequences in real time while static analysis inspects the software's codebase to identify unusual patterns in function calls and text strings that suggest encryption. PayBreak [11] monitors the usage of symmetric session keys, aiding in the decryption of compromised files. Sheen et al. [70] uses machine learning to detect ransomware based on API usage. UShallNotPass [71] prevents ransomware by blocking access to secure pseudo-random number generators and hindering unauthorized operations. Another practical approach involves monitoring the entropy of I/O operations, which relies on the characteristic that ransomware generates high-entropy data during encryption [4]. ShieldFS [10] involves entropy in the features of their machine learning model. Lee et al. [72] apply entropy estimation to spot encrypted files on cloud servers. Joshi et al. [73] combine Shannon's entropy with fuzzy hashing in a unique, signature-less method to trace ransomware behavior. Rcryptect [74] uses a block-level monitoring system that statistically analyzes entropy to distinguish between normal and encrypted data blocks.

8.2 I/O Detection of Ransomware

I/O monitoring is a critical technique in ransomware detection, focusing on scrutinizing the I/O behaviors to identify threats. Ransomware typically adheres to a 'read-encrypt-write' pattern when disrupting files, which forms an indicator of its activity. Therefore, ransomware's frequent interactions with the victim's file system [75] render I/O data a vital resource in its detection.

There are two primary methodologies in I/O-based ransomware detection. The first is rule-based detection, which uses predefined rules and patterns to identify ransomware. Unveil [9] uses predefined rules to identify ransomware by analyzing I/O request patterns. Redemption [8] and R-Locker [76] use transparent buffers or honey files to detect suspicious activities. RansomSpector [34] and Ranacco [77] extend these methodologies by leveraging system calls to capture security indicators, allowing for effective identification and mitigation of ransomware attacks. The second is machine learning-based I/O detection, which extracts features from the I/O sequence and trains the machine learning model for detection. RWGuard [29] integrates machine learning by analyzing IRP events. DeepGuard [78] uses a deep generative autoencoder to capture typical user file-interaction patterns and apply anomaly detection for ransomware. Mustard [79] employs a random forest classifier trained on various file operation features, enhancing the predictive accuracy of the system.

8.3 Main Differences

Compared with previous work, RansomRadar introduces a novel concept centered on the temporal correlation between encryption and I/O behaviors to detect ransomware. To the best of our knowledge, it is the first solution to use distinct HPC patterns for ransomware encryption detection. Instead of relying on fixed patterns of I/O behaviors, RansomRadar models ransomware's behaviors in a more generalized manner to capture the invariant in ransomware's I/O operations. Additionally, RansomRadar focuses on capturing the temporal relationships between encryption operations and I/O operations, thereby enhancing the effectiveness of ransomware detection. The effectiveness of RansomRadar is further proven by high performance and low overhead, demonstrating its viability as a practical solution in the ongoing battle against ransomware.

9 Conclusion

In this paper, we propose RansomRadar, a novel approach to detect ransomware attacks through the temporal correlation between encryption behavior and I/O behavior. Firstly, it leverages the distinct pattern of hardware performance counter values to detect encryption behavior. Thus, it can achieve precise detection regardless of the encryption implementation employed by ransomware. Secondly, we propose I/O throughput to measure the I/O behavior irrespective of certain I/O patterns. Specifically, it measures the file size being potentially encrypted by ransomware along with the number of other basic I/O operations. Finally, RansomRadar determines whether a process is ransomware by analyzing whether there is a temporal correlation between encryption behavior and I/O behavior. To evaluate its effectiveness, we conduct experiments against the latest ransomware samples. The experimental results show that RansomRadar can effectively detect evolving ransomware attacks with a detection rate of 100.00% and 2.68% false alarms.

10 Data Availability

We follow the FSE Open Science Policy and release our code and data on https://github.com/RansomRadar/RansomRadar.

Acknowledgment

We thank the anonymous reviewers for their constructive feedback and suggestions. This research is sponsored in part by the National Key Research and Development Project (No. 2022YFB3104000), NSFC Program (No. U2441238, 62021002).

References

[1] Sophos. 2023. The State of Ransomware 2023. https://www.sophos.com/en-us/whitepaper/state-of-ransomware. (Online; accessed on Nov 10, 2023).

[2] SoftwareOne. 2023. What to do after a ransomware attack? https://www.softwareone.com/en/blog/articles/2021/07/02/ransomware-attack-enterprise-guide. (Online; accessed on Nov 10, 2023).

[3] ZDNet. 2023. These ransomware victims are paying more to recover data. https://www.zdnet.com/article/these-ransomware-victims-are-paying-more-to-recover-data/. (Online; accessed on Nov 10, 2023).

[4] Chijin Zhou, Lihua Guo, Yiwei Hou, Zhenya Ma, Quan Zhang, Mingzhe Wang, Zhe Liu, and Yu Jiang. 2023. Limits of I/O Based Ransomware Detection: An Imitation Based Attack. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2584–2601.

[5] Yiwei Hou, Lihua Guo, Chijin Zhou, Yiwen Xu, Zijing Yin, Shanshan Li, Chengnian Sun, and Yu Jiang. 2024. An Empirical Study of Data Disruption by Ransomware Attacks. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12. doi:10.1145/3597503.3639090

[6] BlackBerry. 2023. BlackCat Malware (AKA ALPHV). https://www.blackberry.com/us/en/solutions/endpoint-security/ransomware-protection/blackcat. (Online; accessed on Nov 10, 2023).

[7] Abdullah Alqahtani and Frederick T Sheldon. 2023. Temporal Data Correlation Providing Enhanced Dynamic Crypto-Ransomware Pre-Encryption Boundary Delineation. *Sensors* 23, 9 (2023), 4355.

[8] Amin Kharraz and Engin Kirda. 2017. Redemption: Real-time protection against ransomware at end-hosts. In *Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings*. Springer, 98–119.

[9] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. 2016. UNVEIL: A Large-Scale, automated approach to detecting ransomware. In *25th USENIX security symposium*. 757–772.

[10] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi. 2016. Shieldfs: a self-healing, ransomware-aware filesystem. In *Proceedings of the 32nd annual conference on computer security applications*. 336–347.

[11] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. 2017. Paybreak: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 599–611.

[12] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. 2022. Deepware: Imaging performance counters with deep learning to detect ransomware. *IEEE Trans. Comput.* 72, 3 (2022), 600–613.

[13] Craig Beaman, Ashley Barkworth, Toluwalope David Akande, Saqib Hakak, and Muhammad Khurram Khan. 2021. Ransomware: Recent advances, analysis, challenges and future research directions. *Computers & security* 111 (2021), 1–22, 102490.

[14] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K Qureshi. 2017. FlashGuard: Leveraging intrinsic flash properties to defend against encryption ransomware. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 2231–2244.

[15] Danny Yuxing Huang, Maxwell Matthaios Aliapoulios, Vector Guo Li, Luca Invernizzi, Elie Bursztein, Kylie McRoberts, Jonathan Levin, Kirill Levchenko, Alex C Snoeren, and Damon McCoy. 2018. Tracking ransomware end-to-end. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*. IEEE, 618–631.

[16] Benjamin Reidys, Peng Liu, and Jian Huang. 2022. Rssd: Defend against ransomware with hardware-isolated network-storage codesign and post-attack analysis. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 726–739.

[17] Abdulrahman Abu Elkhail, Nada Lachtar, Duha Ibdah, Rustam Aslam, Hamza Khan, Anys Bacha, and Hafiz Malik. 2023. Seamlessly Safeguarding Data Against Ransomware Attacks. *IEEE Transactions on Dependable and Secure Computing* 20, 1 (2023), 1–16.

[18] Joon-Young Paik, Joong-Hyun Choi, Rize Jin, Jianming Wang, and Eun-Sun Cho. 2018. A storage-level detection mechanism against crypto-ransomware. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2258–2260.

[19] Yiwei Hou, Lihua Guo, Chijin Zhou, Yiwen Xu, Zijing Yin, Shanshan Li, Chengnian Sun, and Yu Jiang. 2024. An Empirical Study of Data Disruption by Ransomware Attacks. In *Proceeding of the 46th International Conference on Software Engineering*.

[20] Jisung Park, Youngdon Jung, Jonghoon Won, Minji Kang, Sungjin Lee, and Jihong Kim. 2019. RansomBlocker: A low-overhead ransomware-proof SSD. In *Proceedings of the 56th Annual Design Automation Conference*. 1–6.

[21] Donghyun Min, Yungwoo Ko, Ryan Walker, Junghee Lee, and Youngjae Kim. 2021. A content-based ransomware detection and backup solid-state drive for ransomware defense. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 7 (2021), 2038–2051.

[22] Niusen Chen, Josh Dafoe, and Bo Chen. 2022. Poster: Data Recovery from Ransomware Attacks via File System Forensics and Flash Translation Layer Data Extraction. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3335–3337.

[23] Boyang Ma, Yilin Yang, Jinku Li, Fengwei Zhang, Wenbo Shen, Yajin Zhou, and Jianfeng Ma. 2023. 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*. 341–355.

[24] perf 2023. Perf Wiki. https://perf.wiki.kernel.org/. (Online; accessed on Nov 10, 2023).

[25] University of Tennessee Innovative Computing Laboratory. 2023. PAPI. https://icl.utk.edu/papi/. (Online; accessed on Nov 10, 2023).

[26] MicroSoft. 2023. Windows Performance Recorder. https://learn.microsoft.com/en-us/windows-hardware/test/wpt/windows-performance-recorder. (Online; accessed on Nov 10, 2023).

[27] Yifan Yuan, Mohammad Alian, Yipeng Wang, Ren Wang, Ilia Kurakin, Charlie Tai, and Nam Sung Kim. 2021. Don't forget the I/O when allocating your LLC. In *Proceedings of the 48th ACM/IEEE Annual International Symposium on Computer Architecture*. IEEE, 112–125.

[28] Tanvir Ahmed Khan, Muhammed Ugur, Krishnendra Nathella, Dam Sunwoo, Heiner Litz, Daniel A Jiménez, and Baris Kasikci. 2022. Whisper: Profile-guided branch misprediction elimination for data center applications. In *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture*. IEEE, 19–34.

[29] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. 2018. Rwguard: A real-time detection system against cryptographic ransomware. In *International symposium on research in attacks, intrusions, and defenses*. Springer, 114–136.

[30] MicroSoft. 2023. CryptoAPI System Architecture. https://learn.microsoft.com/en-us/windows/win32/seccrypto/cryptoapi-system-architecture. (Online; accessed on Nov 10, 2023).

[31] OpenSSL Project Authors. 2023. OpenSSL cryptography and SSL/TLS toolkit. https://www.openssl.org/. (Online; accessed on Nov 10, 2023).

[32] Individual Contributors. 2023. Cryptography. https://cryptography.io/en/latest/. (Online; accessed on Nov 10, 2023).

[33] Yiwen Xu, Zijing Yin, Yiwei Hou, Jianzhong Liu, and Yu Jiang. 2022. MIDAS: safeguarding iot devices against malware via real-time behavior auditing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4373–4384.

[34] Fei Tang, Boyang Ma, Jinku Li, Fengwei Zhang, Jipeng Su, and Jianfeng Ma. 2020. RansomSpector: An introspection-based approach to detect crypto ransomware. *Computers & Security* 97 (2020), 101997.

[35] MicroSoft. 2023. Event Tracing for Windows (ETW). https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-. (Online; accessed on Nov 10, 2023).

[36] scikit-learn 2023. scikit-learn. https://scikit-learn.org/stable/. (Online; accessed on Nov 10, 2023).

[37] The Linux Foundation. 2023. PyTorch. https://pytorch.org/. (Online; accessed on Nov 10, 2023).

[38] virustotal 2023. VirusTotal. https://www.virustotal.com/gui/contact-us/premium-services. (Online; accessed on Nov 10, 2023).

[39] VX-Vault 2023. VX Vault. http://vxvault.net/ViriList.php. (Online; accessed on Nov 10, 2023).

[40] 2023. tutorialjinni. https://www.tutorialjinni.com/download-free-malware-samples.htm. (Online; accessed on Nov 10, 2023).

[41] 2023. InTheWild. https://samples.vx-underground.org/samples/Blocks/InTheWild%20Collection/. (Online; accessed on Nov 10, 2023).

[42] 2023. Bazaar. https://bazaar.abuse.ch/browse/. (Online; accessed on Nov 10, 2023).

[43] 2023. theZoo. https://thezoo.morirt.com/. (Online; accessed on Nov 10, 2023).

[44] 2023. vx-underground. https://www.vx-underground.org/malware.html. (Online; accessed on Nov 10, 2023).

[45] dhuertas. 2024. *AES*. https://github.com/dhuertas/AES

[46] dhuertas. 2019. *block-cipher-modes*. https://github.com/dhuertas/block-cipher-modes

[47] waterJuice. 2018. *WjCryptLib*. https://github.com/WaterJuice/WjCryptLib

[48] kokke. 2024. *tiny-AES-c*. https://github.com/kokke/tiny-AES-c

[49] B-con. 2015. *crypto-algorithms*. https://github.com/B-Con/crypto-algorithms

[50] matt wu. 2017. *AES*. https://github.com/matt-wu/AES

[51] SergeBel. 2024. *AES*. https://github.com/SergeyBel/AES

[52] alepacheco. 2017. *FileEncryption*. https://github.com/alepacheco/FileEncryption

[53] wumansgy. 2022. *goEncrypt*. https://github.com/wumansgy/goEncrypt

[54] haomanxing. 2018. *go-aes-ecb*. https://github.com/haowanxing/go-aes-ecb

[55] mervick. 2024. *aes-everywhere*. https://github.com/mervick/aes-everywhere

[56] simplephp. 2021. *encrypt-decrypt*. https://github.com/simplephp/encrypt-decrypt

[57] bozhu. 2015. *AES-python*. https://github.com/bozhu/AES-Python

[58] ricmoo. 2017. *pyaes*. https://github.com/ricmoo/pyaes
[59] the javapocal. 2022. *Python-File-Encryptor*. https://github.com/the-javapocalypse/Python-File-Encryptor
[60] marcobellaccini. 2023. *pyAESCrypt*. https://github.com/marcobellaccini/pyAesCrypt
[61] pcaro90. 2013. *Python-AES*. https://github.com/pcaro90/Python-AES
[62] keepsimple1. 2023. *libaes*. https://github.com/keepsimple1/libaes
[63] adrgs. 2020. *rust-aes*. https://github.com/adrgs/rust-aes
[64] anoadragon453. 2018. *rust-aes*. https://github.com/anoadragon453/rust-aes/
[65] MicroSoft. 2023. DISKSPD. https://github.com/microsoft/diskspd. (Online; accessed on Nov 10, 2023).
[66] Kenan Begovic, Abdulaziz Al-Ali, and Qutaibah Malluhi. 2023. Cryptographic ransomware encryption detection: Survey. *Computers & Security* (2023), 103349.
[67] Felix Gröbert, Carsten Willems, and Thorsten Holz. 2011. Automated identification of cryptographic primitives in binary programs. In *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011*. Springer, 41–60.
[68] Pierre Lestringant, Frédéric Guihéry, and Pierre-Alain Fouque. 2015. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. 203–214.
[69] Carlo Meijer, Veelasha Moonsamy, and Jos Wetzels. 2021. Where's Crypto?: Automated Identification and Classification of Proprietary Cryptographic Primitives in Binary Code. In *30th USENIX Security Symposium*. 555–572.
[70] Shina Sheen and Ashwitha Yadav. 2018. Ransomware detection by mining API call usage. In *2018 International Conference on Advances in Computing, Communications and Informatics*. IEEE, 983–987.
[71] Ziya Alper Genç, Gabriele Lenzini, and Peter YA Ryan. 2018. No random, no ransom: a key to stop cryptographic ransomware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 15th International Conference*. Springer, 234–255.
[72] Kyungroul Lee, Sun-Young Lee, and Kangbin Yim. 2019. Effective ransomware detection using entropy estimation of files for cloud services. In *International Symposium on Pervasive Systems, Algorithms and Networks*. Springer, 133–139.
[73] Yash Shashikant Joshi, Harsh Mahajan, Sumedh Nitin Joshi, Kshitij Pradeep Gupta, and Aarti Amod Agarkar. 2021. Signature-less ransomware detection and mitigation. *Journal of Computer Virology and Hacking Techniques* 17, 4 (2021), 299–306.
[74] Seungkwang Lee, Nam-su Jho, Doyoung Chung, Yousung Kang, and Myungchul Kim. 2022. Rcryptect: Real-time detection of cryptographic function in the user-space filesystem. *Computers & Security* 112 (2022), 102512.
[75] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. 2023. RTrap: Trapping and Containing Ransomware With Machine Learning. *IEEE Transactions on Information Forensics and Security* 18 (2023), 1433–1448.
[76] José Antonio Gómez-Hernández, L Álvarez-González, and Pedro García-Teodoro. 2018. R-Locker: Thwarting ransomware action through a honeyfile-based approach. *Computers & Security* 73 (2018), 389–398.
[77] Timothy McIntosh, ASM Kayes, Yi-Ping Phoebe Chen, Alex Ng, and Paul Watters. 2021. Dynamic user-centric access control for detection of ransomware attacks. *Computers & Security* 111 (2021), 102461.
[78] Gaddisa Olani Ganfure, Chun-Feng Wu, Yuan-Hao Chang, and Wei-Kuan Shih. 2020. DeepGuard: Deep generative user-behavior analytics for ransomware detection. In *2020 IEEE International Conference on Intelligence and Security Informatics*. IEEE, 1–6.
[79] Davide Sanvito, Giuseppe Siracusano, Roberto Gonzalez, and Roberto Bifulco. 2022. MUSTARD-Adaptive Behavioral Analysis for Ransomware Detection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1–3.

Received 2024-09-13; accepted 2025-01-14