

RABIS: Ransomware Analysis Based on Information Theory and System Calls

Pranitha Sanda

`pranitha.sanda@gmail.com`

National Forensic Sciences University

Harineesha Nutakki

C. R. Rao Advanced Institute of Mathematics

Valli Sai Meghana Attili

C. R. Rao Advanced Institute of Mathematics

Thapasvi Pooguri

C. R. Rao Advanced Institute of Mathematics

Amrutha Varshini Astakala

C. R. Rao Advanced Institute of Mathematics

Research Article

Keywords: Ransomware, System calls, Entropy, n-gram, Patterns, Behavioral analysis

Posted Date: October 15th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10370663/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

RABIS: Ransomware Analysis Based on Information Theory and System Calls

Pranitha Sanda^{1,*}
pranitha.sanda@gmail.com

Harineesha Nutakki²
harineeshanut@gmail.com

Valli Sai Meghana Attili²
meghana.attili@gmail.com

Thapasvi Pooguri²
poogurithapasvi@gmail.com

Amrutha Varshini Astakala²
amruthavarshiniastakala@gmail.com

¹ National Forensic Sciences University, Goa, India

² C. R. Rao Advanced Institute of Mathematics, Statistics and Computer Science, Hyderabad, India

* Corresponding author: pranitha.sanda@gmail.com

Abstract—Ransomware is one of the most lucrative and safe malware for cybercriminals. It makes data inaccessible by locking, encrypting, or exfiltrating it until a ransom is paid. With the emergence of Ransomware-as-a-Service (RaaS), ransomware attacks have become more sophisticated and aggressive, causing organizations huge financial damage in the millions of dollars. Thus, it becomes imperative to have ransomware detection techniques that are robust and accurate with minimal performance overhead.

In this paper, we propose a lightweight statistical approach to analyze the nature of ransomware through pattern-based behavioral analysis of system calls. To distinguish benign and ransomware processes we experimented with 150 benign process samples and 4 distinct ransomware binaries. Each ransomware binary was executed in a controlled environment with access to approximately 1,30,000 files. Our proposed approach detects ransomware processes with 98.05% accuracy.

Index Terms—Ransomware, System calls, Entropy, n-gram, Patterns, Behavioral analysis

I. INTRODUCTION

Ransomware remains one of the most disruptive and damaging forms of cybercrime, capable of paralyzing critical operations and causing irreversible harm. Ransomware attacks commonly result in multimillion-dollar losses as well as catastrophic effects like irreversible data loss, service downtime, and reputational damage. As ransomware attacks become more focused and financially destructive, many organisations continue to pay substantial amounts of money to cybercriminals. According to Microsoft's 2024 Digital Defense Report [1], from July 2022 to the end of June 2024, there was a 2.75-fold rise in ransomware-related encounters. Remarkably, unmanaged devices serving as the initial entry point for remote encryption in more than 90% of attempts that reached the ransom stage. According to the CERT-In 2024 report [2], a growing number of sophisticated groups that distribute ransomware are targeting virtualisation environments and taking advantage of cloud infrastructure misconfigurations. Despite adhering to established security protocols, numerous Indian organisations have seen ransomware attacks on regular basis. Thus, it is high time for organizations and governments to

invest in solutions that withstand ransomware attacks rather than pay huge amounts to cyber-criminals.

Considering the threats it poses, it is necessary to understand the behaviour of ransomware. It can be classified into 3 types based on its nature and severity: locker ransomware, scareware, and crypto-ransomware. *Locker ransomware* makes a few services on the system inaccessible, e.g., desktop, applications, input devices, etc., leaving the user files and operating system unaffected. *Scareware* is another type of ransomware that attempts to mislead users into believing that their system is infected, urging them to purchase fake software or services to fix the problem. Whereas, *crypto-ransomware* is a type of ransomware that encrypts users' files. There is an extensive increase in crypto-ransomware compared to locker and scareware ransomware due to its ability to cause significant threats to users' data. Thus, we are considering proposing a detection solution for crypto-ransomware.

Crypto-ransomware typically targets specific file types such as .zip, .doc, .pdf, and others that are likely to contain important user data. To carry out the attack, it systematically scans the environment to identify suitable files and proceeds to encrypt them sequentially. This process triggers a sequence of operations that particularly involve repeated file operations (e.g., read, write and rename) executed frequently, as the ransomware reads original data and writes encrypted content. These bursts of high-frequency input and output activity create a unique execution pattern, particularly across different file types and folders, unlike conventional software. Ransomware programs typically have minimal levels of unpredictability in their system call patterns and follow a repeating pattern of activity during their runtime. Benign processes, on the other hand, typically demonstrate complex operational logic and dynamic user involvement, exhibiting irregular and varied system call utilization. This clear behavioural contrast helps in identifying malicious activity indicative of a ransomware attack.

Our proposed approach, *RABIS: Ransomware Analysis Based on Information Theory and System Calls* captures and analyzes the behaviour of ransomware around events, such as

mass file operations, to facilitate the detection of malicious activity. To achieve this, we use System call patterns. System calls provide detailed information about process behaviour as they serve as a low-level interface between the operating system and applications.

Most of the early research in this area focus heavily on graph-based detection techniques and machine-learning models. While these approaches can be proved to be effective, they often demand significant computational resources. In contrast, the proposed method offers a simpler alternative by utilizing statistical techniques to analyze system call patterns and extract meaningful behavioural insights, avoiding the overhead associated with complex learning models.

Our paper is organized as follows: Section II details the existing work relevant to our study. Section III discusses the proposed approach in depth. Section IV presents the experimental setup. Section V analyzes and assesses the obtained results, providing key insights. Finally, Section VI summarizes our findings and outlines potential directions for future research.

II. RELATED WORK

An extensive research exists on ransomware detection that covers a wide range of platforms like PC, enterprise workstations, Mobile devices and Internet of Things (IoT). Harun et al. [3] delivers an exhaustive survey of the current existing solutions that address ransomware threats. Ransomware detection primarily comprises of two approaches: Static and Dynamic. Static approach, such as signature-based detection, highly relies on digital fingerprints database like code strings, hash values and file structures to identify the known ransomware. However, these methods can be bypassed by changing the ransomware's signature. Dynamic approach, on the other hand, concentrates on detecting the ransomware in real - time by analysing it's behaviour. Kok et al. [4] analyses the ransomware threats and their detection techniques. They emphasize on the rise of Ransomware-as-aService (RaaS) and explore the effectiveness of machine learning, honeypots and statistical methods. It emphasizes the need of hybrid approaches that combines the static and dynamic data. This review lacked experimental validation and doesn't address the real-time detection.

A. Ransomware Detection Methods using Machine Learning

Bae et al. [5] Presented the ransomware detection technique by dynamically extracting the Windows API sequences that categories files as Benign, Malicious and Ransomware. N-gram sequences were created using Intel PIN tool to track the API calls. CF-NCF weighted system was implemented to provide class-specific attributes priority. The method relies on dynamic analysis within virtual machines, automated with Python and `vmrun`, and uses six machine learning classifiers for detection.

Khammas et al. [6] proposed a static analysis method for ransomware detection that extracts features from executable byte streams using 4-gram sliding windows. Features undergo

pattern mining, normalization, and dimensionality reduction with Gain Ratio, followed by classification with a Random Forest classifier. This approach relies on static signatures derived from raw data.

Suthar et al. [7] proposed RANSEC, a hybrid ensemble model for ransomware detection in cyber-physical systems that combines two strategies: weighted voting (SVM + Random Forest with accuracy-proportional weights) and stacking (SVM + XGBoost as base classifiers with logistic regression as meta-classifier). Feature extraction covers process activities, file modifications, and network behaviour. Using scikit-learn and TensorFlow, dynamic weighting adjusts classifier contributions based on real-time performance metrics, while the meta-learning layer corrects base classifier errors to improve final prediction accuracy.

Singh et al. [17] proposed a hybrid ensemble integrating DT, ANN, and XGBoost using static (entropy, opcode sequences) and dynamic (API calls, registry changes) features from a Kaggle dataset. With PCA for dimensionality reduction, the model achieved 99.87% accuracy, outperforming CNN, SVM, Logistic Regression, and Naïve Bayes, and employed SHAP for feature interpretability.

B. Ransomware Detection Methods using Graph Theory

El Khairi et al. [8] introduce an anomaly-based intrusion detection system for containerized environments that uses context-aware system call analysis to identify deviations from normal behaviour by modelling syscall sequences and analyzing their contextual dependencies. Each sequence is encoded into an anomaly vector and fed into an autoencoder neural network to detect abnormal patterns.

Nikolopoulos [9] proposed a graph-based model for malware detection and classification that converts system call dependency graphs into group relation graphs, using Δ -similarity for malware detection and SaMe-NP metrics to classify malware into families.

C. System Call-Based Ransomware Detection Techniques

Chew et al. [10] propose a real-time ransomware detection method for Android crypto ransomware using a two-layer FSM. The first layer detects individual malicious behaviours, while the second identifies sequences to confirm ransomware activity. The model uses a lightweight, tokenized system call streaming, avoiding machine learning and sandboxing.

Ahmed et al. [11] proposed a ransomware detection approach using a two-step feature reduction technique with system call refinement and an Enhanced Maximum Relevance Minimum Redundancy (EmRmR) method. The system call refinement eliminates irrelevant API calls, while EmRmR reduces computational overhead. Five classifiers, including SVM and Decision Tree, were tested, and the system was validated with a real testbed of benign apps.

D. Forensic Analysis of Ransomware Attacks

Neweva et al. [12] proposed a forensic framework for detecting and mitigating live ransomware attacks on Linux-based systems, combining memory analysis, network traffic

TABLE I: Comparison of Ransomware Detection Studies

Publication	Static/Dynamic	Approach	Platform	Limitations
[5]	Dynamic	N-gram on API calls	Windows	Misses lightweight ransomware and complex behaviours due to API log size filtering and limited n-gram size.
[6]	Static	N-gram on file content	N/A	It relies on a narrow dataset and static feature extraction, reducing effectiveness against novel, obfuscated, or real-time ransomware threats.
[7]	Dynamic	Hybrid ensemble (weighted voting and stacking)	Cyber-Physical Systems	High computational complexity, limited cross-dataset validation and stacking incurs higher latency on resource-constrained CPS nodes.
[8]	Dynamic	System call patterns (contextual)	Containers	Misses instant attacks and triggers false positives due to legitimate activity variations.
[9]	Dynamic	Graph-based density on node	Windows	Relies on dynamic analysis which can be time-consuming, and may miss malware that behaves differently in controlled environments.
[10]	Dynamic	FSM for pattern detection	Android	Narrow Android focus, small dataset, potential false positives from benign apps, and reduced accuracy from token-based FSM abstraction.
[11]	Static	System call refinement + feature reduction	Windows	The proposed EmRmR approach enhances the detection accuracy but it heavily relies on Windows XP environments for system call tracing. This approach is outdated and unrepresentative of the modern Windows behavior as a result limiting its real world applicability and generalization of the results.
[12]	Dynamic	Memory + network + process monitoring	Linux	A controlled lab setup reduces real-world applicability, and memory analysis introduces high resource overhead, limiting continuous deployment.
[13]	Static+Dynamic	Static and dynamic forensic signatures	Windows/Linux	Relies heavily on static analysis, lacks real-time dynamic tracking, is vulnerable to evasion techniques, and is tested on a limited dataset.
[14]	Dynamic	Dynamic anomaly matrix	Windows	Training data bias, overfitting risks in high-activity systems, and reduced accuracy over prolonged use.
[15]	Dynamic	Pattern-based signature extraction	Windows/Linux	High computational demands for processing high-dimensional data, potential issues in low-resource systems, and requires optimization in anomaly filtering and pattern indexing.
[16]	Dynamic	Transformer using process resource metrics	Linux (Cloud VM)	Single-dataset evaluation; limited malware families; high Transformer overhead; reduced IoT suitability.
[17]	Static+Dynamic	Hybrid ensemble (DT + ANN + XGBoost) with PCA and SHAP	Windows/Linux	Evaluated solely on a single Kaggle dataset, limiting generalisability; PCA-based dimensionality reduction may discard discriminative features; no real-time deployment validation; limited robustness against unseen or zero-day ransomware variants.

monitoring, and process monitoring. Memory analysis extracts volatile data like encryption keys, network monitoring detects command-and-control traffic, and process monitoring identifies behavioural indicators like rapid file changes and abnormal CPU usage.

Natsos and Symeonidis [16] proposed a Transformer-based malware detection model using process resource utilization metrics. The approach achieved 97.44% accuracy, outperforming an LSTM baseline and demonstrating strong robustness under limited training data. The study also introduced dynamic malware signatures to identify key behavioral indicators and showed that malware can be detected through its indirect impact on other system processes.

E. Anomaly-Based and Signature-Based Detection

Subedi et al. [13] proposed CRSTATIC, a static ransomware detection approach that is significant for its ability to identify ransomware through reverse engineering and cryptography-related DLL analysis, using association rule mining to distinguish malicious binaries from benign ones.

Brownor et al. [14] introduced the Dynamic Anomaly Matrix (DAM), which is important for enabling real-time ransomware detection by identifying deviations in system behaviour without relying on static signatures.

LaRocque et al. [15] proposed APSE, a notable framework that uses unsupervised learning to automatically generate behavioural signatures, allowing effective detection of both known and zero-day ransomware variants.

III. PROPOSED APPROACH

We introduce our proposed approach *RABIS*, a ransomware detection technique that distinguishes malicious processes from the benign processes based on their low-level behavioral signatures. The primary concept lies in the observation that ransomware, due to its very nature, performs periodically occurring operations such as mass file encryption, irregular file renaming, frequent file access events, and so on. These activities participate in low-level interactions with system resources such as file I/O and memory, which can be acknowledged in the form of system calls.

Our approach leverages on this by analysing how frequently various patterns of system calls are invoked together and occur repeatedly during the execution of a single process. Such patterns are especially noticeable in many ransomware variants. For example, to encrypt a single file, ransomware variants perform three basic activities that include: 1) Opening the file, 2) Reading the contents of the file, 3) Encrypting the contents and 4) Renaming file extensions. A similar set of operations are performed across user files in the system, creating highly consistent and recurring system call patterns, thereby causing the same pattern to appear multiple times. Ransomware is hence more likely to exhibit frequent system call patterns. Whereas, a few benign processes including command line tools, text editors, and media players etc communicate with the system in a more diverse manner. Their system calls usually reflect a wide range of operations like handling user inputs, reading configurations, accessing network resources and performing background checks, which leads to patterns that are irregular and have less overall frequency during their time of execution. These variations create a higher level of unpredictability in the system call sequences which reflect natural randomness in legitimate process behavior.

To determine the extent of this randomness, we rely on Shannon entropy, an information-theory metric that depicts the degree of uncertainty or variance in the occurrence of patterns across the sample space. A significantly smaller entropy value indicates more repetitive and structured activity, common in ransomware. A larger entropy value on the other hand, suggests less predictable or more diverse activity, typical in benign operations where the activity flow of the user cannot be predicted (context-driven).

RABIS is comprised of three modules, each of which aims to systematically extract, quantify, and compare the behavioural patterns:

- **Module 1 :** In the first module, we focus on extracting patterns from system call traces. Fixed-length subsequences of system calls (n-grams) are extracted from a process's execution trace. These n-grams capture localized behavioral units of the process. Since ransomware iteratively performs actions over a large number of files, the same n-grams tend to occur more frequently, forming a strong behavioral indicator.
- **Module 2 :** In the second module, we compute the Shannon entropy over the distribution of these n-grams to measure the randomness. Low entropy, which represents repetitive behavior, arises when a small set of system call patterns is invoked frequently, as in ransomware, whereas high entropy reflects greater randomness and variation, typical of benign processes.
- **Module 3 :** The third module classifies the processes based on normalized entropy values and computes the threshold.

Together, these modules form *RABIS*. A visual representation of the overall model is shown Figure 1. The workflow shows the interaction between modules and how they influence

the final classification result.

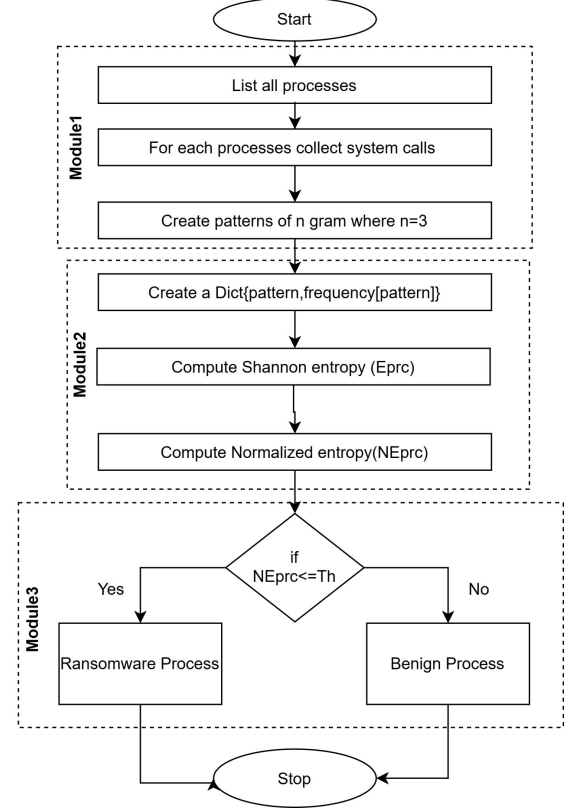


Fig. 1: RABIS Workflow

We further formalized *RABIS* in Algorithm 1 to describe the step-by-step procedure for collecting system calls, generating n-grams, computing entropy, and normalizing the final entropies. Because of this structured workflow, *RABIS* manages to generate results that are both interpretable and computationally efficient.

ALGORITHM 1 *RABIS* Algorithm

```

procedure RABIS(processes[ $P_i$ ],  $n$ )
  for all processes[ $P_i$ ] do
    patterns[]  $\leftarrow$  GenerateNGramPatterns(Sys-
      temCalls_ $P_i$ [],  $n$ )
    Dict{pattern, frequency}  $\leftarrow$ 
      BuildDict(patterns)
     $E_{prc}, S \leftarrow$  ComputeEntropy(Dict)  $\triangleright$  Compute
      the entropy,  $S \in \sum frequency(pattern)$ 
     $NE_{prc} \leftarrow E_{prc} / \log(S)$   $\triangleright$  Normalize the entropy
    if  $NE_{prc} \leq threshold$  then
      Classify as ransomware process
    else
      Classify as benign process
  
```

A. Module 1: System Call Collection and Pattern Generation

1) *System Call Collection* : The way a process interacts with the operating system during its execution is primar-

ily through system calls. For example, `open()`, `write()` and `rename()` are a few file handling system calls. There are similar low-level functions for system operations like memory management, process control, and network communication. Tracing these system calls helps in identify the behaviors by closely observing which calls are being made and in what order by a process. This helps discover the behavioural anomalies by bringing out deviations from typical execution patterns.

During its execution, ransomware shall use these system calls to perform actions such as modifying files, overwriting existing content, renaming files, or deleting user data. Due to the fact that these calls have definitive and unavoidable interactions with the system, tracking them gives us a detailed and transparent picture of what a certain process is doing. For example, using system calls like `write()`, `rename()`, or `unlink()` frequently in brief intervals (all of which are commonly employed to remove files), may indicate malicious, turbulent activity that differs from how legitimate processes operate.

We use `strace`, a Linux utility that records and logs system calls made by a process, to keep an eye on these system calls during controlled testing. Every system call, including the arguments and return values, is shown in detail by `strace`. Because of these characteristics, it is beneficial for examining how ransomware alters the system files. The following command must be run in order to retrieve the system call trace of a suspicious process, `ransomware_binary`:

```
sudo strace -o trace.log ./ransomware_binary
```

This command executes is as follows: Because ransomware samples frequently need elevated permissions to execute or access certain system resources, `sudo` guarantees that enough privileges are granted. The system call tracing is triggered by `strace`, and the `-o trace.log` flag saves the complete trace output to a file called `trace.log` for simpler analysis. Lastly, the ransomware process under observation is carried out via `./ransomware_sample`. After the process has run, system call behaviour and process interactions can be studied by analyzing the `trace.log` file. A portion of the output of `strace` command is displayed in Example 1.

Example 1:

```
open("file.txt", O_WRONLY|O_CREAT|O_TRUNC,
0666) = 3
write(3, "ENCRYPTED DATA", 14) = 14
close(3) = 0
unlink("file.txt") = 0
```

This example presents a sample sequence of system calls upon using `strace` on the REvil ransomware binary. Each of these calls helps to develop an understanding of a process's behaviour. The `open()` call indicates that the process is opening a file named "file.txt". The `write()` call indicates that the process is writing data to the file, which may include encrypting the contents. The `close()` operation shows that the file descriptor was successfully closed, whereas the `unlink()` call indicates that the file was deleted. This is an increasingly prevalent

feature in ransomware operations, which frequently delete the original unencrypted files when the encryption is complete.

To accurately analyze the process behaviour, we first preprocess the obtained `strace` logs by extracting only the system call names, trimming their parameters and return values. The following system calls have been extracted from the trace displayed in Example 1:

```
open
write
close
unlink
```

This abstraction helps eliminate noise and focus on the important execution flow patterns of processes. While parameters such as file names, flags, and modes can provide additional contextual information, in this work we restrict our analysis to system call sequences alone. Incorporating parameters remains a potential direction for future enhancements.

2) *N-Gram Pattern Generation*: The raw sequences of system calls collected using `strace` are processed using an n-gram sliding window technique. Given a process trace $p = \{sc_1, sc_2, \dots, sc_n\}$, where sc denotes an individual system call (eg. `open()`, `write()` etc.). We generate all contiguous n-length subsequences (n-grams) as:

$$\text{pattern}_i = (sc_i, sc_{i+1}, \dots, sc_{i+n-1}) \quad (1)$$

where $i \in \{1, 2, \dots, k\}$, k is the total number of patterns and n is the size of each pattern. Further, we break the long system call trace, into k overlapping sub-sequent patterns each of length n using sliding window mechanism, a fixed-size window of size n is moved one element at a time across the full sequence, capturing short, temporally ordered fragments of system behavior. This ensures the capturing of localized behavioural execution flow patterns. Algorithm 2 outlines n-gram pattern generation using sliding window technique.

ALGORITHM 2 Generate N-Gram Patterns

```
procedure GENERATENGRAMPATTERNS(SystemCalls_Pi[], n)
    ▷ Generate all possible consecutive patterns of length
    'n' from the System Calls Pi[]
    for all  $i = 0$  to  $\text{length}(\text{SystemCalls\_P}_i) - n$  do
         $\text{patterns.append}(P[i] \text{ to } P[i + n - 1])$ 
    return patterns[]
```

B. Module 2: Behavioral Analysis

In this section, the activity of a process during execution is examined, with an emphasis on its actions and patterns that contribute to the classification of its behaviour.

The behavior of a ransomware process differs significantly from that of a benign process. Let us understand their behavioural difference using examples. Example 2 is used to present benign process behaviour, and Example 3 is used to present ransomware process behaviour.

Example 2: In this example, let us consider 'ls' a benign process and analyze its behaviour. 'ls' is a command line tool used to list all the files, directories, and sub-directories. System call patterns for 'ls' are collected using the `strace` tool. Considering the space constraint, we have presented a few initial system calls for the 'ls' process shown in Figure 2. A sliding window of size 3 is applied across this sequence to extract overlapping patterns, where each window captures 3 consecutive system calls. This process results in 7 unique patterns representing the behavioral signature of the 'ls' process.

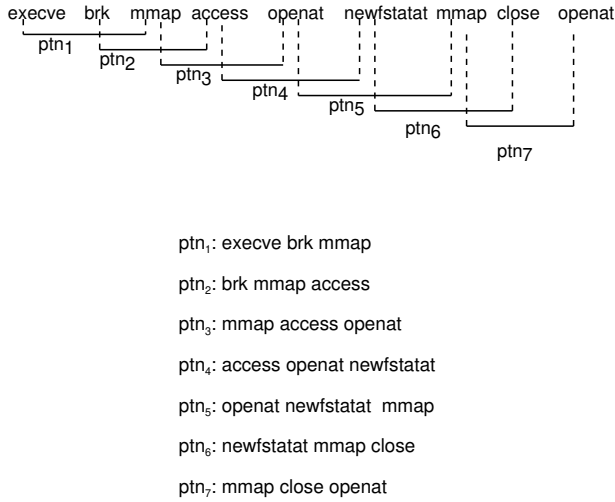


Fig. 2: System call pattern and window visualization for benign process 'ls'

Example 3: To analyze the behaviour of the ransomware process we consider 'REvil', a ransomware binary that encrypts user files. System call data for 'REvil' is gathered using the same procedure applied to the 'ls' process. Figure 3 highlights a specific execution flow patterns of the trace during its encryption phase. As discussed in the previous Example 2 by sliding a window of size 3 over this sequence, overlapping groups of system calls are extracted. This yields 4 unique patterns that reflect the distinct behavioral characteristics of REvil, as illustrated in Figure 3.

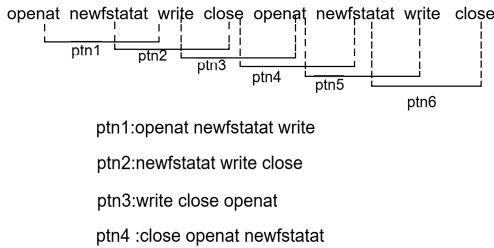


Fig. 3: System call pattern and window visualization for ransomware process 'REvil'

Now let us compare the behavioural differences between benign process 'ls' Example 2 and ransomware process 'REvil' Examples 3. The 'ls' process in Example 2 exhibits a broad range of unique and structured system call patterns as shown in Figure 2, reflecting varied activity (i.e., more random) typical of benign behavior. In contrast, REvil in Example 3 shows repetitive and narrowly focused patterns (i.e., less random), particularly centered on file write and metadata operations, a behavior consistent with deterministic ransomware actions and mass file modifications. Here 'ls' process shows 7 unique patterns while 'REvil' begins repeating patterns after just 4 unique patterns. The diversity in the number of unique patterns generated in 'ls' and 'REvil' reflects their respective degree of randomness and variation. This provides a strong basis for behavioral variation.

Algorithm 3 constructs a dictionary, denoted as $\text{Dict}\{\text{patterns}, \text{frequency}\}$, for each process to record the frequency of observed system call patterns. In this dictionary, each unique pattern acts as a key, and its corresponding value represents how often that pattern occurs. For every new pattern encountered during analysis, the algorithm first checks whether it already exists in the dictionary. If it does, the associated frequency is incremented; if not, the pattern is added with an initial frequency of 1. This pattern-frequency mapping is built individually for all 150 benign processes and 4 ransomware processes. We capture these patterns and store them in the dictionary for further behavioral analysis.

ALGORITHM 3 Create Dictionary for system call patterns

```

procedure BUILDDICT(patterns[])
    Dict{pattern, frequency}
    for all pattern in patterns[] do
        frequency ← Dict.getValue(pattern)
        if frequency = 0 then
            Dict.add(pattern, frequency)
            frequency ← frequency + 1
        else
            Dict{pattern, frequency + 1}
    return Dict

```

To better understand the distribution and behavioral implications of these patterns, we analyze their frequency characteristics across different processes, as shown in Table II, benign processes tend to exhibit low-frequency and diverse system-call patterns. In contrast, Table III demonstrates how ransomware processes repeatedly invoke specific system-call sequences, leading to significantly higher frequencies. Interestingly, Table IV lists benign processes that exhibit ransomware-like behavior due to burst of system-call repetitions, increasing the scope for higher number of false positives. To reduce these false positives, we have identified 2 crucial parameters i.e 1) n-gram size and 2) the volume of data being encrypted. We in the later sections - V discuss how these parameters effect our results.

TABLE II: Benign processes' patterns and their frequency

Process	Pattern	Frequency
google_meet	rt_sigaction , rt_sigaction , read	1
	read , read , read	2
	dup2 , close , newfstatat	1
	dup2 , close , dup2	1
Calculator	prlimit64 , munmap , geteuid	1
	wait4 , chdir , pipe2	2
	wait4 , wait4 , pipe2	2
		2
status	pread64 , newfstatat , pread64	2
	futex , write , openat	1
	read , close , futex	1
	openat , read , fstat	2
	read , fstat , mmap	1
fail2ban	fcntl , mmap , access	1
	newfstatat , openat , newfstatat	2
	access , openat , newfstatat	1
	newfstatat , openat , newfstatat	2

TABLE III: Ransomware processes' patterns and their frequency

Process	Pattern	Frequency
Conti	write , newfstatat , openat	3748
	getpid , getpid , lseek	3706
	getpid , write , lseek	2532
REvil	close , rename , write	1935
	clock_nanosleep , futex , clock_nanosleep	15767
	getdents64 , clock_nanosleep , openat	3025
	openat , newfstatat , getdents64	2578
Monti	clock_nanosleep , openat , newfstatat	5162
	lseek , getpid , write	12500
	getpid , getpid , lseek	12499
	close , rename , stat	10980
Sougoulock	stat , stat , stat	652
	getrandom , getrandom , getrandom	16443824
	write , newfstatat , openat	1329
	getpid , tgkill , sched_yield	2951
	futex , futex , getrandom	1021

TABLE IV: Benign files replicating Ransomware patterns

Process	Pattern	Frequency
clamav	read , read , read	78944
	munmap , munmap , munmap	4798
	read , mmap , read	4047
	mmap , read , read	2877
halt	recvmsg , recvmsg , recvmsg	1571
	mmap , mmap , mmap	18
	openat , newfstatat , openat	10
	mmap , close , openat	10
nmap	socket , fcntl , fcntl	1004
	setsockopt , setsockopt , setsockopt	1003
	setsockopt , connect , socket	963
	read , read , read	164
settings	read , read , read	35104
	recvmsg , sendmsg , poll	1718
	newfstatat , close , openat	1056
	openat , newfstatat , close	1197

C. Module 3: Computing the Entropy and Normalization

Shannon entropy is a widely used measure of randomness or unpredictability in a system. It quantifies how uncertain or diverse a distribution is, with higher entropy indicating more randomness, and lower entropy reflecting predictable or repetitive behavior. Shannon Entropy is driven by the formula shown in Equation 2. Here, $p(x_i)$ represents the probability of occurrence of event x_i .

$$H(X) = - \sum_{i=1}^n p(x_i) \log_2(p(x_i)) \quad (2)$$

We use Shannon entropy particularly for capturing concentrated behavior, such as mass file operations using system call patterns. Entropy reaches its maximum (i.e, the process exhibits highly diverse and unpredictable behaviour) when all outcomes are equally likely to occur. In contrast, entropy becomes lower when certain outcomes occur more frequently. For better understanding, let us consider an extreme case where entropy is 0. If an event occurs every time, i.e., the probability of occurrence of an event is 1 (it occurs with absolute certainty), the term $\log_2(p(x))$ is 0, leading to an overall entropy of 0, indicating complete predictability with no randomness and vice-versa.

1) *Calculation of Shannon Entropy for RABIS:* In context of ransomware detection, we now substitute event x_i in Equation 2 with the system call patterns i.e., $pattern_i$ as shown in Equation 1.

$$x_i = pattern_i \quad (3)$$

To compute entropy, we first calculate the frequency of each unique pattern within a process. Next, to determine the probability of occurrence of each pattern, we need the total sum of the frequencies of all patterns. Therefore, we define S as sum of the frequencies of all patterns in the process, calculated using Equation 4.

$$S = \sum_{i=1}^k frequency(pattern_i) \quad (4)$$

For each pattern, the probability of its occurrence is denoted as $p(pattern_i)$. This probability is calculated when the frequency of each pattern is divided by the total number of occurrences: S . This is formulated as given in Equation 5.

$$p(pattern_i) = \frac{frequency(pattern_i)}{S} \quad (5)$$

Once the probability of each pattern is obtained, the Shannon Entropy for the process is calculated using the Equation 6.

$$E_{prc} = - \sum_{i=1}^k p(patterns_i) \log_2 p(patterns_i) \quad (6)$$

Algorithm 4 offers us a complete methodology of this section.

ALGORITHM 4 Compute entropy of system-call patterns for each process

```

procedure COMPUTEPROCESSENTROPY(Dict)
  for all patterns in Dict do
     $S \leftarrow S + \text{Dict.getPatternFreq}(\text{patterns}) \triangleright S \in \sum \text{frequency}$ 
  for all patterns in Dict do  $\triangleright$  Compute probability for each process
     $p \leftarrow \text{Dict.getPatternFreq}(\text{patterns})/S$ 
     $E_{prc} \leftarrow -(E_{prc} + p * \log(p)) \triangleright$  Compute the entropy
  return  $\{E_{prc}, S\}$ 

```

2) *Normalization of Entropy*: To enable fair and meaningful comparisons of entropy across different processes, normalization is essential. Normalization standardizes the entropy range to the interval [0, 1]. Here, the normalized entropy of each process is represented by NE_{prc} and is calculated using Equation 7.

$$NE_{prc} = \frac{E_{prc}}{\log(S)} \quad (7)$$

Beyond classification, normalization reduces the influence of uneven pattern distributions, allowing behavior-based distinctions to emerge more clearly. As shown in Table V, normalized entropy effectively differentiates ransomware from benign processes. Ransomware processes typically exhibit lower entropy than benign ones due to repeated and predictable patterns. In contrast, benign processes tend to generate more randomized and diverse data, resulting in higher entropy.

Ransomware	Normalization	Benign Files	Normalization
Conti	0.2122	domainname	0.9899
Monti	0.1900	date	0.9787
REvil	0.1785	apparmour	0.9784
SougoLock	0.0352	chown	0.9783

TABLE V: Normalization comparison between Ransomware and Benign processes

D. Classifying the Process

Once entropy values are calculated for the system call patterns and normalized as previously discussed, we assume that these values follow a normal distribution, where most of the entropy values are close to the mean and lesser spread towards the extremes. The values that are nearer to mean emerge naturally for benign processes, which, due to their non-repetitive and diverse nature, generate high entropy values.

We apply statistical reasoning to define a left-tailed confidence interval, which identifies processes with abnormally low entropy values. Specifically, we compute a statistical threshold (denoted as Th) that serves as a boundary; only a small number of benign entropy values are expected to fall below this threshold. Processes with $NE_{prc} > Th$ are considered benign, while the ones with $NE_{prc} < Th$ are flagged as suspicious, typically indicating repetitive or structured behavioral characteristics of ransomware.

To arrive at this threshold, we first compute the mean normalised entropy (μ_{NE}), corresponding to the average entropy value over all benign processes in a batch using Equation 8.

$$\mu_{NE} = \frac{1}{N} \sum_{P=1}^N NE_{prc} \quad (8)$$

Here, the normalised entropy of process prc is denoted by NE_{prc} , where N represents the number of benign processes.

Next, we calculate the standard deviation, σ_{NE} , to evaluate the dispersion or variance of the entropy values using Equation 9.

$$\sigma_{NE} = \sqrt{\frac{1}{N-1} \sum_{P=1}^N (NE_{prc} - \mu_{NE})^2} \quad (9)$$

Using Equation 10, we build a confidence interval (CI) that represents the predicted range of entropy values under the normal distribution.

$$CI = [\mu_{NE} - z \cdot \sigma_{NE}, \mu_{NE} + z \cdot \sigma_{NE}] \quad (10)$$

In this equation, z represents the critical value from the z -distribution that corresponds to the required significance level. At a 95% confidence level (or a 5% left-tail), the z value is around -1.96 .

Since we are only interested in identifying abnormally low entropy values (less random), we consider only the lower bound of the confidence interval as the threshold, as illustrated in Equation 11.

$$Th = \mu_{NE} - z \cdot \sigma_{NE} \quad (11)$$

This formulation uses a left-tailed test, which means we label a process as suspicious if its entropy is much lower than what is expected for benign behaviour.

To make this approach more robust, we divide the benign dataset into numerous equal-sized chunks, calculate the threshold for each batch, and then average them to get the final threshold. This is expressed by Equation 12.

$$Th_{\text{final}} = \frac{1}{B} \sum_{b=1}^B Th_b \quad (12)$$

Here, B is the number of batches, and Th_b is the threshold calculated for each batch b .

The choice of acceptance level influences detection sensitivity. A higher acceptance level results in a wider range and a more conservative threshold, reducing false positives. A lower level tightens the range, improving sensitivity but increasing the risk of false alarms.

By modeling the behavior of the Entropy of benign processes using this statistically grounded, left-tailed approach, we develop a mechanism that is interpretable and effective for detecting ransomware behaviors without requiring supervised learning or large labeled datasets.

IV. EXPERIMENTAL SETUP

In this section we detail the experimental setup for our model *RABIS* that simulates both benign and ransomware processes on Ubuntu based systems. The system setup involved four identical systems running Ubuntu 22.04.5 LTS with 16 GB RAM and 756.2 GB disk space. These systems were utilized to run four variants of ransomware in controlled environments.

The ransomware samples were downloaded from Malware-Bazaar [18] to ensure that the testing was performed using real-time ransomware binaries. The ransomware samples used were:

- **Conti** [19]: Known for encrypting files very quickly by using multiple threads simultaneously, allowing it to lock data across large systems efficiently.
- **REvil (Sodinokibi)** [20]: A powerful ransomware that often infiltrates systems by exploiting security vulnerabilities.
- **Monti** [21]: Functions similar to Conti, but designed specifically to target Linux-based systems, particularly those in cloud environments.
- **SougoLock** [22]: Acts differently from most ransomwares by encrypting files outside its own directory, including those in protected directories, thereby infiltrating deeper into the system.

To assess the behavior of these ransomware strains, we tested them on two datasets of different sizes:

- **Dataset – 1 (Low-Volume)**: Comprising 48,035 files (14.1 GB) of general user documents and application-related documents from our own machines.
- **Dataset – 2 (High-Volume)**: Containing 1,34,557 files including audio files, video files, JPEG images, CSV datasets, and PDF documents, totaling more than 34 GB. These datasets were obtained from public sources such as GitHub repositories [23], [24], Kaggle, [25]–[29], and other open-access websites [30]–[46]. This reflects real-world data distributions in end-user systems.

To test benign process behavior, we ran 150 benign processes that were selected to represent general-purpose applications that are typically used by end-users in day-to-day activities. These included:

- Common Ubuntu command-line tools such as `ls`, `cat`, `cd`, `mkdir`, etc.
- General-purpose programs like text editors, calculators, and other utility applications.
- GUI-based applications such as Evince (PDF viewer) and Firefox (web browser).
- Additionally, we included one antivirus program: ClamAV, a popular open-source antivirus for Unix systems.

V. RESULTS AND DISCUSSION

This section summarizes and discusses the results obtained using the proposed approach, *RABIS*. Normalized entropy is calculated for both benign and malicious processes to assess the effectiveness and reliability of the detection method.

To evaluate *RABIS*, a total of 150 benign processes and four ransomware binaries (Conti, Monti, REvil, and SougoLock) were analyzed (see Section IV). To simulate real-world attack scenarios, the ransomware binaries were executed within an isolated environment with access to a dataset consisting of 34 GB of files. System call traces were collected independently for both benign and ransomware processes using `strace`, followed by trigram construction and entropy-based analysis.

The calculated NE_{prc} values, derived using Equation 7, reveal a clear separation between ransomware and benign processes. The X-axis represents individual processes, while the Y-axis denotes the corresponding normalised entropy values. Ransomware processes are shown in red, and benign processes are shown in blue. A classification threshold (Th), derived as discussed in Section III-D, distinguishes between benign and ransomware behaviour: processes with $NE_{prc} < Th$ are classified as ransomware, whereas those with $NE_{prc} > Th$ are classified as benign. All ransomware processes are correctly classified; however, a small number of benign processes are misclassified as ransomware. The false positives are primarily associated with benign utilities that repeatedly perform file-system operations. Such processes naturally generate highly repetitive system call patterns, producing entropy values similar to those observed during ransomware encryption activities.

To quantitatively evaluate the classification performance of *RABIS*, accuracy is computed using Equation 13.

$$Accuracy(\%) = \frac{TP + TN}{TP + FP + TN + FN} \times 100 \quad (13)$$

In this Equation,

- **True Positive (TP)**: A ransomware process correctly classified as ransomware.
- **False Positive (FP)**: A benign process incorrectly classified as ransomware.
- **True Negative (TN)**: A benign process correctly classified as benign.
- **False Negative (FN)**: A ransomware process incorrectly classified as benign.

Using trigram-based behavioural patterns, *RABIS* achieves an overall classification accuracy of 98.05%, correctly identifying all ransomware samples while misclassifying only a small number of benign processes. The absence of false negatives is particularly important in ransomware detection, where missed attacks can result in significant data loss. To further analyse robustness and explore possible improvements, different n-gram sizes were evaluated where $n \in \{3, 5, 7, 9, 11, 15\}$, considering both behavioural relevance and computational efficiency. Figure 5 presents the variation in accuracy across different n-gram sizes, with trigrams yielding the highest performance.

The effectiveness of trigrams can be attributed to the nature of system call behaviour, where many fundamental operations, particularly file-system interactions, are typically executed through short three-step sequences of consecutive system calls. For example, a basic file operation often follows the sequence

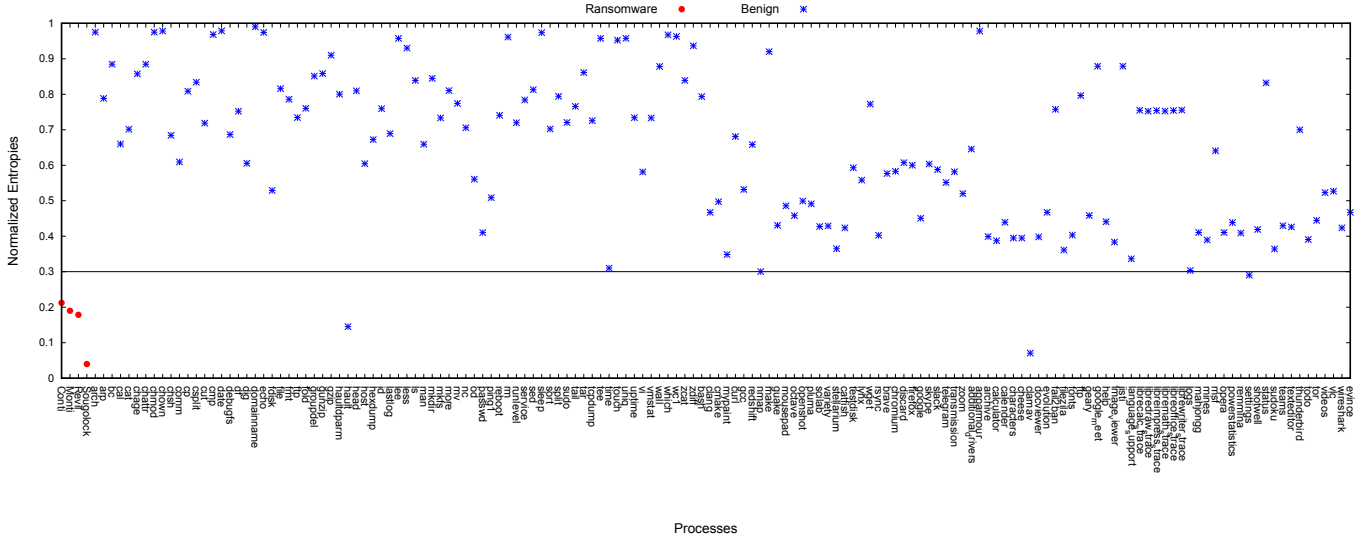


Fig. 4: Normalised entropies of benign and ransomware processes

open $\rightarrow$ write $\rightarrow$ close. Although trigrams capture only a small portion of the execution trace, they retain sufficient contextual information to represent localised behaviour and effectively capture repetition or reordering patterns, making them more resilient to noise than longer n-grams.

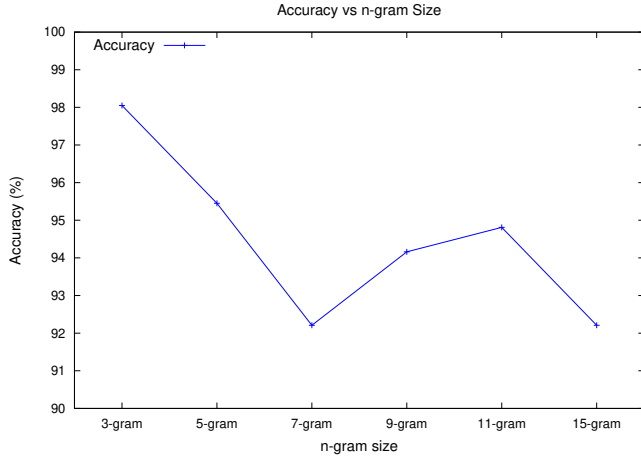


Fig. 5: Accuracy variation with n-gram size

RABIS is further evaluated by varying the volume of encrypted data across two datasets: Dataset-1 (14 GB) and Dataset-2 (34 GB) (refer to Section IV). Each ransomware binary is executed independently on both datasets. As the volume of files increases, system call events become more complex and frequent, producing more pronounced behavioural signatures and higher entropy values. This indicates that ransomware behaviour becomes increasingly distinguishable as the volume of encrypted data grows, resulting in stronger and more separable entropy signatures that benefit detection accuracy in large-scale encryption scenarios.

Notably, the SougoLock ransomware sample exhibits min-

imal entropy change across datasets. Unlike other ransomware binaries that encrypt files within specific directories, SougoLock initiates encryption from the root directory and progressively targets critical system and boot-related files. Consequently, its execution pattern remains largely consistent regardless of dataset size, as shown in Figure 6.

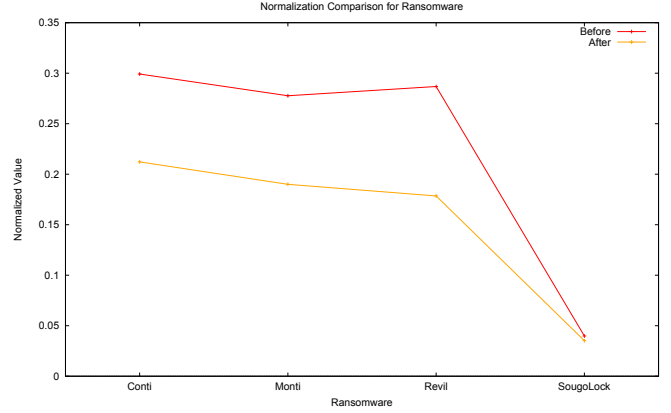


Fig. 6: Comparison between ransomware samples when run on different volumes of files

The experimental results demonstrate that *RABIS* can effectively distinguish ransomware from benign processes using entropy derived from system call patterns. No false negatives were observed during the evaluation, and an overall classification accuracy of 98.05% was achieved. The results further indicate that ransomware behaviour becomes increasingly distinguishable as the volume of encrypted data grows, resulting in stronger behavioural signatures. These findings highlight the effectiveness, efficiency, and practical applicability of the proposed approach for ransomware detection.

VI. CONCLUSION

Our proposed approach, *RABIS*, introduces a lightweight and efficient method for detecting crypto-ransomware through behavioural analysis of system call activity. Unlike existing solutions that rely on complex machine learning pipelines or graph-based models, *RABIS* adopts statistical techniques to differentiate malicious processes from benign ones. The approach focuses on identifying variations in system call patterns that reflect the characteristic operational behaviour of ransomware. As certain benign applications, such as antivirus software or backup utilities may also exhibit low-entropy behaviour. Thus, contextual information is incorporated to improve classification accuracy.

The first module establishes a behavioural baseline by monitoring a process's execution at the system level. System calls are decomposed into trigram sequences to capture behavioural patterns, Shannon entropy is applied to measure the degree of randomness within these sequences.

The second module aggregates and analyzes the frequency of the extracted trigram patterns. Since ransomware typically performs highly repetitive operations, particularly during file encryption, this behaviour results in lower entropy values. Consequently, this stage plays a crucial role in strengthening the distinction between ransomware and benign processes.

In the final module, the computed entropy values are evaluated against a dynamic threshold. As the amount of observed data increases, this threshold gradually decreases which enables earlier and more accurate ransomware detection.

By integrating these three modules, *RABIS* effectively detects ransomware based on runtime behavioural characteristics, even in cases where malicious processes attempt to mimic benign activity. At the same time, the approach remains computationally efficient and avoids the overhead associated with complex detection models. Experimental results demonstrate that *RABIS* achieves a detection accuracy of 98.05%, highlighting its effectiveness and reliability.

In summary, *RABIS* provides a robust and practical framework for classifying processes as benign or ransomware by leveraging system call analysis through information theory metrics.

VII. FUTURE WORK

Future work will focus on automating the proposed framework by integrating data collection, feature extraction, pattern generation, and process classification into a unified detection pipeline. Such automation would improve the practicality of deploying the framework in real-world environments.

Another direction is the development of intelligent process filtering mechanisms. Since tracing system calls from all running processes can introduce significant overhead, lightweight filtering techniques can be used to identify potentially suspicious processes before detailed analysis is performed, thereby improving efficiency.

Finally, we plan to extend *RABIS* into a cross-platform ransomware detection framework. Supporting multiple operating systems would enhance the scalability, deployability,

and applicability of the framework across diverse computing environments.

DECLARATIONS

Funding: No funds, grants, or other support were received for conducting this study.

Conflicts of Interest: The authors declare that they have no known financial or non-financial interests that could have appeared to influence the work reported in this paper.

Ethics Approval: This study did not involve human participants, human data, human tissue, or animals. Therefore, ethics approval was not required.

Consent to Participate: Not applicable.

Consent for Publication: Not applicable.

Data Availability: The datasets generated and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Code Availability: The source code developed during the current study is available from the corresponding author upon reasonable request.

Authors' Contributions: The first author led the conception of the study and the overall research process. All authors contributed to the development and refinement of the work, participated in the preparation and revision of the manuscript, and approved the final version.

REFERENCES

- [1] Microsoft Threat Intelligence, "Microsoft Digital Defense Report 2024," 2024, accessed on Apr 30, 2024. Available: <https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/Microsoft%20Digital%20Defense%20Report%202024%20%281%29.pdf>.
- [2] CERT-IN, "India Ransomware Report For H1-2022 By CERT-In," 2024, accessed on Apr 30, 2024. [Online]. Available: https://www.cert-in.org.in/PDF/RANSOMWARE_Report_2024.pdf
- [3] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–37, 2022. [Online]. Available: <https://doi.org/10.1145/3514229>
- [4] S. Kok, A. Abdullah, N. Jhanjhi, and M. Supramaniam, "Ransomware, threat and detection techniques: A review," *Int. J. Comput. Sci. Netw. Secur.*, vol. 19, no. 2, p. 136, 2019. [Online]. Available: https://d1wqtxts1xzle7.cloudfront.net/85021080/20190217-libre.pdf?1651031786=&response-content-disposition=inline%3B+filename%3DRansomware_Threat_and_Detection_Techniqu.pdf
- [5] S. I. Bae, G. B. Lee, and E. G. Im, "Ransomware detection using machine learning algorithms," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 18, p. e5422, 2020. [Online]. Available: <https://doi.org/10.1002/cpe.5422>
- [6] B. M. Khammas, "Ransomware detection using random forest technique," *ICT Express*, vol. 6, no. 4, pp. 325–331, 2020. [Online]. Available: <https://doi.org/10.1016/j.icte.2020.11.001>
- [7] O. P. Suthar, M. W. Bhatt, M. S. Mia, G. Kaur, Y. Prajapati, R. Bhandari, and E. Asenso, "Ransec: Hybrid ensemble learning-based secure approach for ransomware detection in cyber-physical defence systems," *Journal of Applied Science and Technology Trends*, vol. 7, no. 1, pp. 26–40, 2026. [Online]. Available: <https://jastt.org/index.php/jasttpath/article/view/555>

- [8] A. El Khairi, M. Caselli, C. Knierim, A. Peter, and A. Continella, "Contextualizing system calls in containers for anomaly-based intrusion detection," in *Proceedings of the 2022 on Cloud Computing Security Workshop*, 2022, pp. 9–21. [Online]. Available: <https://doi.org/10.1145/3560810.3564266>
- [9] S. D. Nikolopoulos and I. Polenakis, "A graph-based model for malware detection and classification using system-call groups," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 29–46, 2017. [Online]. Available: <https://doi.org/10.1007/s11416-016-0267-1>
- [10] C. J. W. Chew, V. Kumar, P. Patros, and R. Malik, "Real-time system call-based ransomware detection," *International Journal of Information Security*, vol. 23, no. 3, pp. 1839–1858, 2024. [Online]. Available: <https://doi.org/10.1007/s10207-024-00819-x>
- [11] Y. A. Ahmed, B. Koçer, S. Huda, B. A. S. Al-Rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *Journal of Network and Computer Applications*, vol. 167, p. 102753, 2020. [Online]. Available: <https://doi.org/10.1016/j.jnca.2020.102753>
- [12] W. Neweva, O. Fitzwilliam, and J. Waterbridge, "Forensic analysis of live ransomware attacks on linux-based laptop systems: Techniques and evaluation," 2024. [Online]. Available: <https://doi.org/10.21203/rs.3.rs-4900486/v1>
- [13] 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. [Online]. Available: <https://doi.org/10.1109/SPW.2018.00033>
- [14] C. Brownor, P. Andersen, Z. Fischer, and G. Osterberg, "Ransomware detection using dynamic anomaly matrix for accurate and real-time threat identification," 2024.
- [15] A. LaRocque, G. Gross, F. Lindholm, P. Greco, B. Dupont, and J. Kruger, "Effective ransomware detection using autonomous patternbased signature extraction," 2024. [Online]. Available: <https://doi.org/10.22541/au.173016272.26231350/v1>
- [16] D. Natsos and A. L. Symeonidis, "Transformer-based malware detection using process resource utilization metrics," *Results in Engineering*, vol. 25, p. 104250, 2025. [Online]. Available: <https://doi.org/10.1016/j.rineng.2025.104250>
- [17] S. Singh, T. Khanna, and D. K. Verma, "A hybrid ensemble model for ransomware detection using feature engineering and deep learning," *International Journal of Information Technology*, vol. 17, no. 8, pp. 5095–5104, 2025.
- [18] abuse.ch, "MalwareBazaar – A project to share malware samples for research and analysis," 2025. [Online]. Available: <https://bazaar.abuse.ch/>
- [19] signalblur, "Conti Sample," 2025. [Online]. Available: <https://bazaar.abuse.ch/download/8b57e96e90cd95fc2ba421204b482005fe41c28f506730b6148bcef8316a3201/>
- [20] JAMESWT_WT, "REvil Sample," 2025. [Online]. Available: <https://bazaar.abuse.ch/download/f864922f947a6bb7d894245b53795b54b9378c0f7633c521240488e86f60c2c5/>
- [21] ankit_anubhav, "Monti Sample," 2025. [Online]. Available: <https://bazaar.abuse.ch/download/edfe81babf50c2506853fd8375f1be0b7bebbefb2e5e9a33eff95ec23e867de1/>
- [22] abuse_ch, "Sougolock Sample," 2025. [Online]. Available: <https://bazaar.abuse.ch/download/7ea20f9e9f3f4bce48db94f6c2354eae53e21bc130338606d45f51eba6960034/>
- [23] "HAM10000 Dataset," 2025. [Online]. Available: https://github.com/ptschandi/HAM10000_dataset
- [24] "CIFAR Sample," 2025. [Online]. Available: <https://github.com/EN10/CIFAR>
- [25] "IMDB 50K Movie Reviews," 2025. [Online]. Available: <https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews>
- [26] "Customer Data (50k)," 2025. [Online]. Available: <https://www.kaggle.com/datasets/arifmia/customer-data-1>
- [27] "Spotify Songs (50k AI Generated)," 2025. [Online]. Available: <https://www.kaggle.com/datasets/refiaozturk/spotify-songs-dataset>
- [28] "Songs Dataset (2000–2020)," 2025. [Online]. Available: <https://www.kaggle.com/datasets/waqi786/songs-dataset-2000-2020-50k-records>
- [29] "NYUv2 50K Dataset," 2025. [Online]. Available: <https://www.kaggle.com/datasets/awsaf49/nyuv2-50k-dataset>
- [30] "ImageNet Sample Images," 2025. [Online]. Available: <https://github.com/EliSchwartz/imagenet-sample-images>
- [31] "Free Test XML Files," 2025. [Online]. Available: <https://freetestdata.com/xml-files/>
- [32] "Free Test Documents," 2025. [Online]. Available: <https://freetestdata.com/document-files/>
- [33] "Free Test Audio Files," 2025. [Online]. Available: <https://freetestdata.com/audio-files/>
- [34] "Free Test Image Files," 2025. [Online]. Available: <https://freetestdata.com/image-files/>
- [35] "Other Sample Files," 2025. [Online]. Available: <https://freetestdata.com/other-files/>
- [36] "Sample PDFs," 2025. [Online]. Available: <https://file-examples.com/index.php/sample-documents-download/sample-pdf-download/>
- [37] "Audio Sample (M4A)," 2025. [Online]. Available: <https://files.testfile.org/AUDIO/C/M4A/sample1.m4a>
- [38] "Test PDFs," 2025. [Online]. Available: <https://testfile.org/pdf/>
- [39] "All File Sizes," 2025. [Online]. Available: <https://testfile.org/all-sizes/>
- [40] "Sample PDFs (2)," 2025. [Online]. Available: <https://sample-files.com/documents/pdf/>
- [41] "Sample DOCX Files," 2025. [Online]. Available: <https://sample-files.com/documents/docx/>
- [42] "Sample XML Files," 2025. [Online]. Available: <https://sample-files.com/data/xml/>
- [43] "TextInMotion Samples," 2025. [Online]. Available: <https://textinmotion.com/help/samples>
- [44] "Sample Lorem Ipsum," 2025. [Online]. Available: <https://samplelib.com/sample-lipsum.html>
- [45] "Sample JPEG," 2025. [Online]. Available: <https://samplelib.com/sample-jpeg.html>
- [46] "Sample MP4," 2025. [Online]. Available: <https://samplelib.com/sample-mp4.html>