LeARN: Leveraging eBPF and AI for Ransomware Nose Out

Arjun Sekar[†], Sameer G. Kulkarni[†] and Joy Kuri[‡]
[†]Indian Institute of Technology Gandhinagar, [‡]Indian Institute of Science, India

***Abstract*— In this work, we propose a two-phased approach to detect and deter ransomware in real-time. We leverage the capabilities of eBPF (Extended Berkeley Packet Filter) and artificial intelligence (AI) to develop proactive and reactive methods. In the first phase, we utilize signature-based detection, where we employ custom eBPF programs to trace the execution of new processes and perform hash-based analysis against a known ransomware dataset. In the second, we employ a behavior-based technique that focuses on monitoring the process activities using a custom eBPF program and the creation of ransom notes — a prominent indicator of ransomware activity through the use of Natural Language Processing (NLP). By leveraging eBPF's low-level tracing capabilities and integrating NLP based machine learning algorithms, our solution achieves an impressive 99.79% accuracy in identifying ransomware incidents within a few seconds on the onset of zero-day attacks.**

***Index Terms*—Cybersecurity, Ransomware, eBPF, AI, NLP**

I. Introduction

Over the last few years, ransomware has emerged as the preferred weapon for cyber criminals [1]. Ransomware is a type of malware that holds victim's sensitive data as hostage until the attacker receives a ransom payment. As per the Sonicwall Cybersecurity report [2], in 2023 alone, businesses faced a staggering escalation in ransomware attacks, with over 318 million incidents reported - roughly an attempt to attack every 0.1 seconds. The threat extends beyond individual finances, reaching hospitals, power grids, and entire industries, posing a severe risk to public safety and national security. The severity of the threat posed by ransomware is highlighted by its reclassification as a top security concern in the latest White House National Cybersecurity Strategy [3].

The most common approach followed by anti-virus solutions and end-point detection and response tools (EDR) to tackle ransomware includes signature-based detection. This approach works well for previously known ransomwares, however they fail to detect when signature evading techniques like binary obfuscation are used, further they also fail to detect zero-day attacks [4]. Extended Berkeley Packet Filter (eBPF) [5] enables to run lightweight programs securely in kernel space without the need to modify the kernel, and facilitates efficient mode of kernel level observability to implement simple security measures with minimal processing overheads when compared to the solutions developed as user-space applications. Hence, we adopt eBPF for kernel level tracing of the application behavior and to implement security measures that enable to detect and deter the ransomware in real-time.

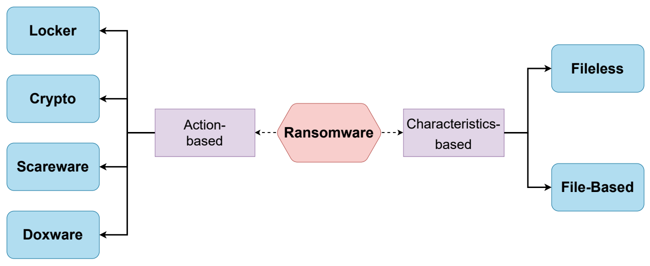

Figure 1. Classification of Ransomware

Natural Language Processing (NLP) [6] — a sub-field of artificial intelligence (AI) enables to interpret and process human language. It encompasses a variety of tasks such as sentiment analysis, spam detection, text summarization, machine translation etc. Given the inherent complexity, ambiguity, and context-dependency of human language, NLP faces numerous challenges. There are several machine learning (ML) algorithms that are commonly used for NLP applications. In our work, we leverage ML algorithms like Support Vector Machines (SVM) , Random Forest *etc.*, to develop a semantic analysis model and detect ransomware by examining the ransom notes. We train the model to look for relevant keywords and patterns in ransom notes and identify suspicious language, demands and threats indicative of ransomware. Accordingly, the key contributions of our work include:

- We perform an exhaustive analysis of existing ransomware to identify their distinguishing features. By tracing the system activities using eBPF, we uncovered specific patterns indicative of ransomware activity. (§III).
- We present a novel two-phased approach composed of static and dynamic modules to provide a robust ransomware detection and deterrence framework. (§IV).
- We develop eBPF modules for static and dynamic analysis (Algo.1). The static analysis relies on hash-based comparison of the binary files against the known set of ransomware samples. In dynamic analysis, we monitor and log system activities to detect any suspicious operations that are indicative of a ransomware. (§IV-A).
- We develop and train a NLP model to detect ransom notes in real-time (§IV-B), and demonstrate its efficacy to detect zero-day attacks with $\sim 99.79\%$ accuracy. (§V).

II. Background

We present a brief background about ransomware classification and eBPF that is relevant to the context of our work.

A. Classification of Ransomware

Ransomware can be classified based on i) the kind of action it performs (action-based) and ii) the characteristics of its operation (operation-based). Figure 1 shows the two kinds of ransomware classifications. The actions of ransomware delineate its immediate impact on the victim's system, ranging from encrypting files and locking down access to outright data exposure threats. Simultaneously, the characteristics of ransomware delve into its underlying attributes, such as the delivery methods, sophistication levels, and overarching strategies employed by cybercriminals.

Classification based on Ransomware Action:

- **Locker Ransomware**: Denies users access to their entire systems by locking them out. Instead of encrypting files, it immobilizes the operating system, demanding a ransom to restore regular access.
- **Crypto-Ransomware**: Encrypts the victim's files, making them inaccessible to the user. Threat actors demand a ransom for the decryption key needed to unlock the files, often causing significant disruptions and data loss.
- **Scareware**: Uses alarming messages or fake security alerts to trick users into believing their computer is infected with malware and prompts users to purchase unnecessary or fraudulent security products or services.
- **Doxware (leakware)**: Threatens to expose sensitive or private information unless the victim pays a ransom. Instead of encrypting files, it focuses on extorting personal data, potentially causing reputational and privacy damage.

Classification based on Ransomware operation:

- **File-less Ransomware**: Operates without leaving traditional executable files on the victim's system. It often exploits system vulnerabilities or uses scripts to execute in memory, making detection and prevention more challenging for traditional antivirus tools.
- **File-Based Ransomware**: Encrypts the files and renders them inaccessible. The attackers then demand a ransom payment, typically in cryptocurrency, to provide the decryption key to restore access to the encrypted files.

B. Extended Berkeley Packet Filter eBPF

eBPF is a powerful in-kernel virtual machine. Although, originally designed for network packet filtering, eBPF has evolved into a universal framework that allows executing custom code snippets in a sandboxed environment within the kernel without the need to recompile kernel or addition of kernel modules [7]. This flexibility enables dynamic and efficient analysis of various aspects of system behavior, leading to its widespread adoption in areas such as system profiling, tracing, security, and performance optimization.

III. Design Rationale

Traditional analysis techniques often overlook critical system interactions and nuances that could provide valuable insights into the complex mechanisms employed by modern ransomware variants. Hence, we first identify the key operational characteristics and run-time behavior of file-based crypto ransomware. We ran an ensemble of 14 ransomware including Kuiper, REvil, *etc.*, on a test virtual machine to unearth the following distinct, yet common features. Table I summarizes our key findings. We discuss below the following observations.

Table I
Primary Syscall Invocations Across Ransomware Variants

Feature	REvil	HelloKitty	WannaCry	Kuiper
Unlink usage	High	Med	High	High
Urandom utilization	High	Low	Low	High
Pkill invocation	High	High	High	Med
Rename syscall	High	High	High	High
Write syscall	High	High	High	High
OpenSSL usage	Low	High	High	Low
CPU utilization	High	High	High	High
Openat usage	High	High	High	Low

eBPF based system call tracing: Here, we specifically instrument the Linux kernel with eBPF to monitor all of the 336 tracepoints available in `sys_enter` system call events. It not only provides an unprecedented level of visibility into the granular activities occurring during a ransomware execution[1], but also offers a more reliable and consistent mechanism than the kprobes (kernel probes) regardless of the variability in kernel versions. We uncovered several distinctive features and behaviors prevalent during ransomware operations that are uncommon in benign system activities. Such key features include i) high CPU usage, ii) frequent invocation of system calls such as *openat*, *unlinkat*, *pkill (process-kill)*, *rename*, *write*, *OpenSSL calls*, and iii) extensive use of *urandom* file.

urandom: Regardless of the specific encryption algorithm or library used, any encryption process requires a source of random numbers for securely generating encryption keys. Our analysis revealed that sophisticated advanced ransomware variants like REvil and Kuiper heavily utilize the *'/dev/urandom'* pseudo-random number generator to aid their encryption processes. This distinctive behavior left a characteristic trace that could not have been identified by merely tracing the usage of OpenSSL or other existing encryption libraries.

Encryption: We observed that ransomware variants like REvil do not rely on standard encryption libraries like OpenSSL or mbedTLS, which can be easily traced by placing uprobes (user-space probes) on `libcrypto.so` and `libmbedtls.so` respectively. Instead, they employ custom-loaded encryption libraries with novel encryption techniques such as the Salsa-20 (used in REvil) and ChaCha-20 (in Kuiper). It is important to note that while sophisticated ransomware like REvil employ custom encryption techniques and avoids standard libraries, some more primitive variants like WannaCry and Hellokitty ransomware directly leverage OpenSSL for encryption. In such cases, placing uprobes on OpenSSL library `libcrypto.so` i.e. `EVP_EncryptInit_ex` and `EVP_CipherInit_ex`, effectively capture their encryption activities.

File Operations: Ransomware variants often create temporary `.lck` or `.tmp` files as intermediary files. After creating the

[1]This exhaustive tracing is possible in our test environment where only the ransomware application is run.

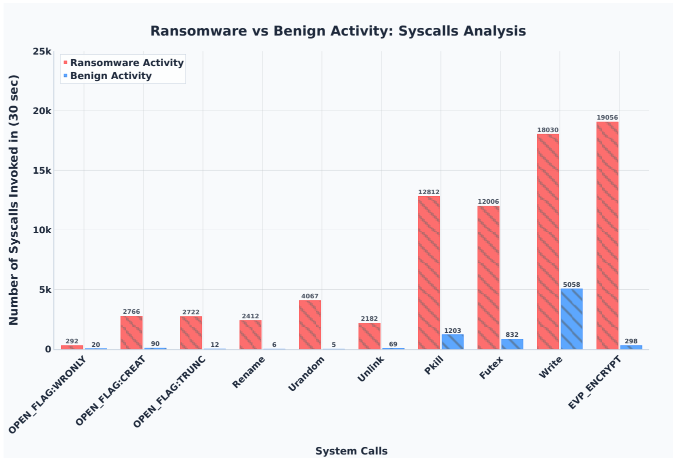

Figure 2. Syscall Invocation Frequency: Ransomware vs. Benign Processes

final encrypted file, the ransomware deletes these temporary files using *unlink* or *unlinkat* system calls, resulting in very high *unlink/unlinkat* system call usage. The high frequency of *pkill* system call invocation is observed because ransomware typically uses pkill to terminate all running processes before encrypting them. The renaming of files to custom encrypted file extensions leads to the frequent invocation of the `sys_enter_rename` system call. This behavior is a characteristic of ransomware operations, as the ransomware renames the encrypted files with specific extensions like '.iFire' and '.vemar' to indicate their encrypted state.

By comprehensively tracing all 336 `sys_enter` system calls, we uncovered previously unseen aspects of the ransomware attack process, revealing the intricate mechanisms and operations employed by these sophisticated malware variants. The data for ransomware processes was collected by running various ransomware samples for approximately 30 seconds where approximately 1.2GB of data got encrypted. The data for benign processes was obtained from normal system usage over a 30-second period. Figure 2 illustrates the comparative frequency of system call invocations observed during ransomware execution (red) versus benign processes (blue). We can observe that ransomware activities result in i) very high frequency of system calls related to file activities like *open*, *write*, *rename*, *unlink*, ii) the usage of *create* with *truncate* flags is common, iii) the usage of *urandom*, *futex* and encrypt system calls is also predominant.

IV. Proposed Solution

We propose a novel two-phased approach that combines static and dynamic behavioral analysis techniques. The key dynamic property we focus on is to monitor the creation of ransom notes and leverage machine learning to classify the ransom notes as malicious or benign. Fig. 3 shows the proposed two-phase ransomware detection pipeline.

Static Analysis: We use eBPF to trace the exec system calls and capture the details of new processes being executed. We then perform hash-based detection on the process by comparing its hash with a dataset obtained from MalwareBazaar [8] consisting of SHA-256 hash for 777,073 malware samples. This static analysis serves as the initial line of defense.

Dynamic Analysis: When the static analysis layer fails to detect any malicious indicators, i.e. zero-day attacks, the dynamic (second) layer of our approach takes over, focusing on the behavior-based detection of ransom note creation, a distinctive and essential step in the ransomware attack lifecycle. Ransomware typically follows a well-defined sequence of actions, including file encryption, followed by the creation and delivery of a ransom note containing instructions for payment and file recovery. While existing behavior-based techniques may monitor file system activities or system call patterns, our technique aims to provide a more targeted and effective defense against ransomware operations by detecting and responding to this critical step in the ransomware kill chain. After analyzing the system calls invoked during ransomware operations, we focused on observing and analyzing the file creation patterns using the openat system call. The tracepoint at `sys_enter_openat` has an argument (flags) of type int, which can be decoded using the definitions in /usr/include/asm-generic/fcntl.h. By understanding these flag values, we can determine the file access modes being used. For instance, the flag `0x00000100` corresponds to `O_CREAT`, indicating a "create-only" file access mode. `O_RDONLY (0x00000000)` represents read-only access, and so on. While observing the file creation patterns and correlating them with the `fcntl.h` definitions, we noticed a crucial pattern: ransom notes were consistently created with the `O_CREAT` flag across various ransomware variants. This observation led to our approach of monitoring the openat system call with the `O_CREAT` file access flag to detect the creation of new files, which the generation of a ransom note would trigger during a ransomware operation. When a new file is created, its content is analyzed using machine learning models trained to distinguish between genuine ransom notes and benign files. This classification is based on various features extracted from the file, such as textual patterns, keywords, formatting, and other relevant characteristics.

If the created file is classified as a potential ransom note, our system can take immediate action, such as terminating the responsible process or quarantining the affected system to prevent further damage. By targeting this critical stage of the ransomware attack chain, our approach aims to disrupt the ransomware operation before file encryption and extortion demands can be completed. However, during our analysis, we observed that some advanced ransomware samples, such as Kuiper, are capable of evading detection by utilizing lower-level kernel functions. These ransomware variants can bypass detection mechanisms that focus solely on static entrypoints like `sys_enter` tracepoints. For instance, tools like strace are limited to monitoring system call entry points and are ineffective at detecting ransomware that need monitor deeper kernel-level functions.

In order to address this challenge, we leveraged eBPF to probe kernel functions such as `vmlinux:vfs_open`, which interact directly with the file system at the kernel

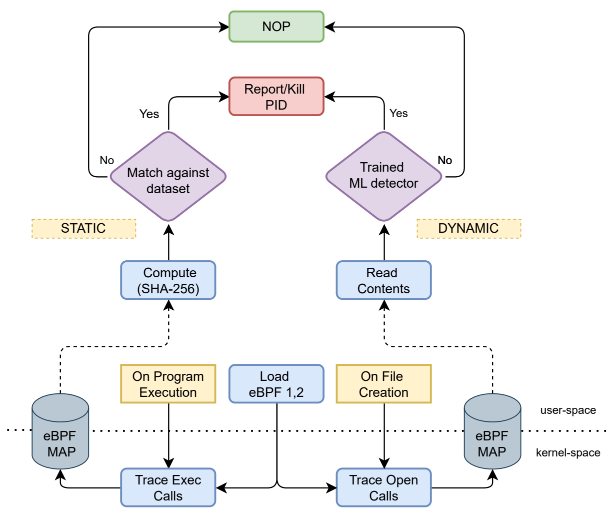

Figure 3. Real-Time Ransomware Detection Pipeline using eBPF and ML

level. This approach proved to be more effective than relying solely on tracepoints like `sys_enter_openat` for detecting ransomware activities. In our analysis, we monitored various tracepoints and kernel functions, revealing that low-level kernel function tracing (e.g., `vmlinux:vfs_open`) provided deeper insights into the behavior of advanced ransomware. These kernel functions are invoked after the system calls, meaning ransomware can evade detection by using system calls that eventually invoke these lower-level functions. Tracing such low-level kernel functions enabled us to detect and profile ransomware like Kuiper that would otherwise evade detection through tracepoints such as `sys_enter_openat`. This comprehensive tracing method allowed us to capture ransom note creation even when system call tracepoints failed.

It is crucial to note that the effectiveness of our technique in protecting files varies depending on the specific ransomware behavior. The majority of crypto-ransomware variants primarily fall into two categories: those that create a ransom note after encrypting each directory and those that create a ransom note before encrypting any files. In the former case, our technique may sacrifice a few files and folders before detecting and disrupting the ransomware operation. However, it can prevent the encryption of other substantial amounts of files and folders. In the latter case, where the ransom note is created before any encryption occurs, our technique can protect all files from being encrypted.

Sophisticated ransomware variants often stay idle for extended periods, slowly encrypting small volumes of files without drawing attention. In such cases, detection techniques relying solely on high CPU usage, *unlink/write* activity, or high I/O operations may not yield impressive results, as these variants maintain a low overhead. Therefore, dynamic analysis focused on detecting ransom note creation after the encryption of a directory becomes an important aspect of identifying these sophisticated ransomware variants.

Algorithm 1 *eBPF* Pseudocode for Dynamic Analysis

```
 1: Function DYNAMIC_ANALYSIS()
 2: #include <asm-generic/fcntl.h>
 3: #define FILE_CREATION_THRESHOLD T
 4: config = {
 5:    stack_mode ← "perf",
 6:    max_strlen ← 256
 7: }
 8: On sys_enter_openat, vfs_open trigger:
 9:    pid ← get_current_process_id()
10:    process_name ← get_current_process_name()
11:    flags ← get_syscall_flags()
12:    if not (pid in process_counter_map) then:
13:       process_counter_map[pid] = {0}
14:    if flags & O_CREAT then:
15:       pid ← get_current_process_id()
16:       filename ← get_syscall_argument(args->filename)
17:       process_counter_map[pid].file_creation_count += 1
18:       if process_counter_map[pid].count ≥ T then:
19:          log_file_creation(pid, filename)
20: End Function
```

A. Real-Time Detection

The real-time ransomware detection system integrates eBPF for dual-layer security by combining static and behavioral analysis. Initially, eBPF tracepoints are placed on the `exec` family syscalls to track program executions. A Python script computes the SHA-256 hash of each executable and checks it against a malware hash dataset, terminating any malicious process immediately. Concurrently, the same eBPF program also monitors file creation events, capturing the file path of newly created files if the number of creation events per process exceeds a threshold value T, with the flag `O_CREAT` attribute and the corresponding Process ID (PID) of the process initiating the creation, i.e. ransomware in the case of ransom note creation (lines 18,19 Algo. 1). A Python script then retrieves these file contents and uses our pre-trained machine-learning model to classify them as potential ransomware ransom notes or benign files. Upon detecting a ransom note, the script terminates the associated process. This swift response stops the ransomware operations within a few (15-50) milliseconds of the ransom note's creation. As illustrated in Fig. 3, this integrated system architecture enables rapid and automated detection and mitigation of ransomware attacks.

B. ML-based Detector

We prepare a diverse dataset, comprising both real-world ransomware ransom notes and a wide variety of benign text files. Using this dataset, we train a machine-learning model to accurately distinguish between malicious ransom notes and legitimate files based on their content and characteristics.

To represent the "ransomware" class, we leveraged the publicly available "ransomware_notes" dataset on GitHub by Threat_labz [9]. From this repository, we collected 210 unique ransom notes from various ransomware variants. Additionally, we executed 7 different Linux ransomware samples in a controlled virtual environment to obtain seven more ransom notes, bringing the total number of unique ransom notes in our dataset to 217. To represent the "benign" class, we utilized the dataset compiled by Ken Lang [10], consisting of text

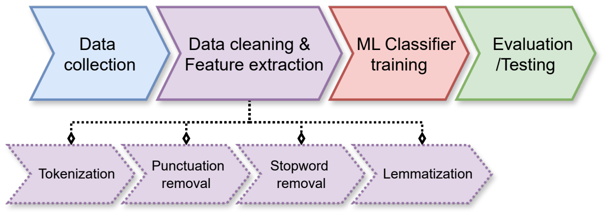

Figure 4. Machine Learning Pipeline

files covering a wide range of topics, including technology, sports, politics, business, entertainment, *etc*. We collected 177 such files and complemented this dataset with 50 README files collected from various public repositories. We also added 27 files consisting of various code scripts (Python, Shell, C, HTML, etc.) and other general notes containing information related to ransomware and malware to enrich the benign data diversity and make it more robust, bringing the total number of benign files data to 254.

We then employ the Natural Language Toolkit (NLTK) library [11], to apply a set of effective pre-processing techniques on the collected text data as depicted in Fig. 4. These techniques include tokenization, punctuation removal, stop word removal and word normalization. At this stage, it is crucial to preserve the keywords that distinguish ransomware notes from benign text for which we use lemmatization. After pre-processing, the lemmatized tokens are joined into a single string, representing the processed content for each sample in the dataset. To extract meaningful features from the pre-processed text, we employ two techniques, namely, Term Frequency-Inverse Document Frequency (TF-IDF) and Chi-Squared Feature Selection. The TF-IDF approach creates a vector representation of the text, where each dimension corresponds to a word in the corpus, and the values represent the importance of that word in the given text based on its frequency and inverse document frequency. Further, the Chi-Squared feature selection method which measures the dependence between the features and the target classes (ransomware or benign) selects the most informative features from TF-IDF.

We find that using the top 400 features yields the best results in terms of accuracy. The extracted features were used to train a Random Forest classifier with 75 estimators, utilizing a 7:3 train-test split and leveraging ensemble learning to capture complex patterns while ensuring strong generalization and minimizing overfitting. The trained model, TF-IDF vectorizer, and feature selector are saved to a pickle file for future use in detecting and classifying potential ransom notes.

V. Preliminary Evaluation

Offline testing on collected data: Table II summarizes the effectiveness of different ML classifiers in detecting the ransomware in terms of Accuracy, Precision, Recall, CV Score. We tested with several ML models, namely, Logistic Regression (LR), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Decision Tree, Naive Bayes, Gradient Boosting, AdaBoost, and Stochastic Gradient Descent (SGD). The RF classifier achieved an accuracy of 99.79% on test data and reliably distinguished the ransom notes from benign files. The recall score indicates the model's ability to identify 99.54% of the actual ransomware ransom notes present. Notably, the average 5-fold cross-validation score of 99.36% demonstrates the model's generalization capability to unseen data. This high score gives us confidence that the model can maintain its performance when deployed in real-world scenarios without overfitting to the training set.

Table II
Classifier Performance Comparison

Classifier	Accuracy	Precision	Recall	CV Score (K=5)
Logistic Regression	0.9936	1.00	0.9862	0.9894
Random Forest	0.9979	1.00	0.9954	0.9936
Support Vector Machine	0.9979	0.9954	1.00	0.9979
K-Nearest Neighbors	0.9108	0.9944	0.8111	0.8366
Decision Tree	0.9936	1.00	0.9816	0.9598
Naive Bayes	0.9936	0.9864	1.00	0.9830
Gradient Boosting	0.9936	1.00	0.9816	0.9809
AdaBoost	0.9936	0.9954	0.9908	0.9851
Stochastic Gradient Descent	0.9979	0.9954	1.00	0.9936

Table III
Real-Time Detection Results

Ransomware	Result	Time to Detect and Kill (secs)	Affected Files
HelloKitty	Success	1.023	0.07%
REvil	Success	2.529	0.39%
Rhysidia	Success	3.120	0.34%
LockBit	Success	4.642	0.53%
Kuiper	Success	4.853	0.58%
Conti	Success	5.183	0.46%
IceFire	Success	5.403	0.56%

Although SVM achieved the same overall accuracy (99.79%), with a slightly higher cross-validation score, we chose the Random Forest classifier as it outperformed SVM and other classifiers in handling the edge cases. Our tests on 10 benign notes containing ransomware-related terminology such as "malware," "ransomware," "locked," and "encrypt", revealed that Random Forest accurately classified 9 out of 10 cases, compared to SVM's performance of 7 out of 10. This distinction highlights RF's superior ability to capture intricate feature interactions through ensemble learning, which is crucial in real-world scenarios where such ambiguities exist.
Real-time testing on unseen ransomware: In order to validate the efficiency and robustness of our proposed solution, we tested with 7 unseen Linux ransomware samples whose ransom notes were not part of the training data. Our detection model was able to successfully detect the ransomware operation and kill the ransomware process within a few seconds for different ransomware as shown in Table III. Notably, these ransomware samples typically take about 90 seconds to completely encrypt the sample files (size of 3.5GB) in the VM. This real-world testing demonstrates the effectiveness and rapid response capabilities of our solution, minimizing the potential damage caused by ransomware attacks.

VI. Related Works

Zhang *et al.* [12] proposed a static analysis framework based on N-gram opcodes with machine learning for ransomware classification. However, this approach fails to handle obfuscated ransomware executable and polymorphic codes. Kharraz *et. al.* [13] proposed monitoring abnormal file system activities as a mechanism to identify ransomware. They further extended their research by developing UNVEIL [14], a dynamic analysis system that detects ransomware by monitoring and analyzing the behavioral patterns of file system activities such as suspicious I/O operations, file modifications, and changes to the desktop environment that are indicative of ransomware activities. Scaife *et al.* [15] developed CryptoDrop, employing techniques such as calculating file entropy using Shannon's formula, measuring file version similarity through hash comparisons, and tracking file deletions to detect ransomware. Similarly, Chen *et al.* [16] proposed monitoring file deletions, creations, renames, and changes for ransomware detection.

Our behavioral model augments these works by detecting the creation of ransom notes. It focuses on kernel-specific countermeasures against ransomware, particularly relevant given the increasing number of ransomware attacks. Zhang *et al.* [17] proposed a defense system using eBPF for machine learning-based ransomware detection. Their system collects data on the volume of file read/write, deletion, and renaming operations, extracting features for machine learning to detect and classify ransomware. They highlight a novel protection feature that addresses the time lag in machine learning detection, providing a mechanism to temporarily copy and store data when files are accessed in specific directories. However, it is computationally intensive and becomes infeasible to copy the content of all the open files in real-time to safeguard them from encryption. Zhuravchak *et al.* [18] proposed a similar method using eBPF alongside network traffic monitoring.

We distinguish our approach by using machine learning for ransomware note detection instead of relying on features obtained from the volume of read, write, rename, and delete operations as performed by the former. Our method offers several advantages, particularly in addressing the limitations of existing techniques when dealing with sophisticated ransomware variants. Modern ransomware often employs evasion tactics, such as remaining dormant for extended periods and encrypting files in small volumes to evade detection based on high CPU usage or excessive I/O operations. These sophisticated variants can maintain a low overhead, potentially bypassing detection methods that rely solely on monitoring file opening, renaming, deleting read/write volumes, or system resource utilization. By concentrating on ransom note detection, our method allows for more reliable detection of ransomware, even when it operates at a low intensity over extended periods.

VII. Conclusion and Future work

We proposed a two-phase approach for timely detection and deterrence of ransomware attacks by developing new eBPF and ML algorithms. Furthermore, we also presented a novel behavior-based ransomware detection using eBPF-based and NLP-based ML models that can detect the creation of ransom notes with very high accuracy (99.79%), and thus able to effectively detect and deter ransomware operations within a few seconds to facilitate real-time mitigation.

Future Work: We are extending the static analysis phase with the creation of pattern matching based on Yara rules to further improve the detection capabilities. We also aim to develop a more robust behavioral model that can build on the sequence distinct process/file operations and correlate the time-series data to aid for robust recurrent neural network based models to detect any zero-day attacks. We will also evaluate the performance of our solution in large-scale environments.

Acknowledgment

This work is supported by grants from NVIDIA corporation, the Govt. of India, Department of Science and Technology, SERB core research grant: CRG/2023/008021 and MeitY grant: 4(3)/2024-ITEA.

References

[1] S. Mansfield-Devine, "Ransomware: the most popular form of attack," *Computer Fraud & Security*, vol. 2017, no. 10, pp. 15–20, 2017.

[2] SonicWall, "Sonicwall cyber threat report," 2024. [Online]. Available: https://www.sonicwall.com/threat-report/

[3] "National cybersecurity strategy," 2023. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2023/03/National-Cybersecurity-Strategy-2023.pdf

[4] M. Alasli and T. Ghaleb, "Review of signature-based techniques in antivirus products," 04 2019, pp. 1–6.

[5] T. Lévai, B. E. Kreith, and G. Rétvári, "Supercharge webrtc: Accelerate turn services with ebpf/xdp," in *Proceedings of the 1st Workshop on EBPF and Kernel Extensions*, ser. eBPF '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 70–76. [Online]. Available: https://doi.org/10.1145/3609021.3609296

[6] S. Bird, E. Klein, and E. Loper, *Natural language processing with python*. Sebastopol, CA: O'Reilly Media, 2009.

[7] D. Calavera and L. Fontana, *Linux Observability with BPF: Advanced Programming for Performance Analysis and Networking*. O'Reilly Media, Incorporated, 2019, accessed: 2024-05-15.

[8] Abuse.ch, "Malwarebazaar dataset," https://bazaar.abuse.ch/export/, 2024, dataset containing SHA-256 hashes of malware samples.

[9] Threatlabz, "An archive of ransomware notes past and present collected by zscaler threatlabz," 2023, accessed: 2024-05-15. [Online]. Available: https://github.com/threatlabz/ransomware_notes

[10] K. Lang, "Newsweeder: Learning to filter netnews," in *Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[11] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[12] H. Zhang, X. Xiao, F. Mercaldo, S. Ni, F. Martinelli, and A. K. Sangaiah, "Classification of ransomware families with machine learning based on n-gram of opcodes," *Future Generation Computer Systems*, vol. 90, pp. 211–221, 2019.

[13] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," 07 2015, pp. 3–24.

[14] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, automated approach to detecting ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 757–772.

[15] N. Scaife, H. Carter, P. Traynor, and K. Butler, "Cryptolock (and drop it): Stopping ransomware attacks on user data," 06 2016, pp. 303–312.

[16] L. Chen, C.-Y. Yang, A. Paul, and R. Sahita, "Towards resilient machine learning for ransomware detection," *ArXiv*, vol. abs/1812.09400, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:56895515

[17] K. Higuchi and R. Kobayashi, "Real-time defense system using ebpf for machine learning-based ransomware detection method," in *2023 Eleventh International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2023, pp. 213–219.

[18] D. Zhuravchak, A. Tolkachova, A. Piskozub, V. Dudykevych, and N. Korshun, "Monitoring ransomware with berkeley packet filter," *Cybersecurity Providing in Information and Telecommunication Systems*, vol. 3550, pp. 95–106, 2023.