Ransomware Detection using Machine Learning with eBPF

Offensive Technologies Project Report

Max Willers
Tomás Philippart

Abstract

Ransomware attacks continue to pose a significant threat, necessitating advanced detection methods. This paper presents a novel ransomware detection system that combines eBPF and Machine Learning techniques to improve efficiency. By leveraging eBPF real-time system event monitoring, coupled with a machine learning pipeline, our approach achieves excellent sensitivity (recall). Extensive experimentation on a dataset of 1.5 million events and 53 ransomware samples showcases a true positive rate of 100% and a Matthews Correlation Coefficient of 68.48%. We identify critical features, including file operation patterns and delete action frequency, contributing to effective ransomware detection. Our research also provides an open source codebase for easy implementation. Future work includes expanding the dataset, incorporating network-based detection, and exploring advanced machine learning models. By fostering collaboration, we can collectively combat the growing ransomware threat and continuously enhance detection mechanisms.

1 Introduction

Ransomware attacks continue to pose a significant and escalating threat to organizations worldwide, resulting in substantial disruptions and financial losses [19, 32, 28]. Recent reports indicate a worrisome surge in such attacks, with the percentage of organizations directly affected by ransomware rising from 37% in 2020 to a staggering 66% in 2021 [7]. While Windows systems have traditionally been the primary target of ransomware attacks, there has been a concerning uptick in attacks targeting Linux systems [15], which are often utilized for critical business applications and infrastructure.

Ransomware, a form of malware designed to extort victims, has gained notoriety due to its ability to encrypt files on compromised systems, holding them hostage until a ransom is paid [12, 19]. To combat this menace, existing ransomware detection methods primarily rely on two approaches: signature-based and behavioral-based techniques. Signature-based detection relies on predefined patterns or signatures to identify known ransomware samples, while behavioral-based detection focuses on analyzing application behavior, encompassing factors such as API calls and network behavior [13].

However, these conventional approaches have their limitations. Signature-based detection struggles to keep pace with the rapid evolution of ransomware variants, rendering it less effective against new and unknown threats. Behavioral-based detection, although valuable, may overlook subtle

yet crucial indicators of ransomware activity, leading to potential false negatives. Moreover, attackers have become adept at obfuscating malware code, complicating static analysis and evading detection.

In recent years, machine learning techniques have garnered attention for their potential in ransomware detection. Notably, the application of deep learning algorithms specifically for ransomware detection remains an understudied area, despite its potential to extract informative features based on ransomware activities. Such an approach enables the detection of previously unseen ransomware samples, bridging the gap left by traditional detection methods.

In this research, we propose the integration of eBPF [8] into the ransomware detection landscape as a promising solution, by answering the following research question:

- How can eBPF be integrated with a Machine Learning pipeline to accurately detect ransomware during runtime?

By leveraging eBPF's capabilities to monitor and filter system events at a low level, we can attain fine-grained visibility into critical actions and behaviors associated with ransomware. This real-time monitoring empowers proactive identification of ransomware activity as it occurs, minimizing the potential damage inflicted by ongoing attacks.

Furthermore, eBPF's lightweight nature and efficient kernel-level hooks facilitate continuous monitoring without significant impact on system performance. Its flexibility and extensibility allow the development of custom monitoring and filtering logic tailored specifically for ransomware detection, leveraging the unique characteristics and behaviors exhibited by ransomware.

To address the growing threat and overcome the limitations of traditional approaches, our research aims to explore the effectiveness of utilizing eBPF to detect ransomware attacks on Linux systems during runtime. By leveraging eBPF's capabilities and incorporating machine learning techniques, particularly deep learning algorithms, we seek to enhance the detection and classification of ransomware samples based on their observed behaviors.

In summary, the integration of eBPF into ransomware detection holds great promise, providing real-time monitoring, fine-grained visibility, low overhead, flexibility, and the potential for deep learning-based analysis. By overcoming the limitations of existing approaches, this research contributes to strengthening ransomware defenses and bolstering the security posture of organizations against these increasingly prevalent and damaging cyber threats.

2 Background

This section introduces the essential concepts surroundings the topics of ransomware detection, machine learning and eBPF.

2.1 Ransomware

Ransomware attacks typically begin with the initial infiltration of a victim's system through various vectors such as phishing emails, malicious attachments, or exploit kits. Once inside the system, the ransomware encrypts critical files or denies access to the entire system, effectively rendering it unusable. The attackers then demand a ransom payment, usually in the form of cryptocurrencies, in exchange for providing the decryption key or restoring access to the compromised system.

The success and profitability of ransomware attacks has been fueled by several factors. First, the anonymous nature of cryptocurrencies makes it difficult to trace the flow of funds, enabling attackers to receive payments without being easily identified. Second, the increasing sophistication of ransomware techniques, including the use of advanced encryption algorithms and evasion tactics, has made it challenging for traditional security measures to detect and mitigate such attacks effectively. Furthermore, the emergence of ransomware-as-a-service (RaaS) models in underground forums has made ransomware attacks more accessible to a broader range of cybercriminals, further contributing to the proliferation of these threats.

The impact of ransomware attacks extends beyond immediate financial losses. Organizations face significant reputational damage, regulatory scrutiny, and potential legal liabilities if they fail to adequately protect sensitive data or respond effectively to such incidents. Additionally, critical infrastructure sectors, including healthcare, finance, and government, are particularly vulnerable to ransomware attacks due to the potentially life-threatening consequences and the disruption of essential services.

2.2 Malware detection techniques

Malware detection relies on malware analysis, an approach that aims to understand the components and behavior of malicious software. Malware analysis can be broadly categorized into two main types: dynamic analysis and static analysis. Dynamic analysis involves studying the behavior and actions of a process during its execution, while static analysis focuses on examining the contents of binary files [12, 17, 31, 21]. Numerous approaches can be employed to detect ongoing ransomware attacks. Honeypots, machine learning, network traffic analysis, file analysis are a few examples of such approaches [12].

2.3 eBPF

eBPF is a technology with roots in the Linux kernel that enables the execution sandboxed programs in a privileged context such as the operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without modifying the kernel source code or loading kernel modules.

The operating system has consistently been considered a prime location for incorporating observability, security, and networking features, primarily because the kernel possesses privileged capabilities to monitor and manage the entire system. Simultaneously, the evolution of an operating system kernel has proven to be challenging due to its crucial role and the need for stability and security. Consequently, innovation within the operating system has traditionally lagged behind advancements implemented in other areas.

The execution of an eBPF program inside the kernel is always event-driven. Pre-defined hooks include system calls, function entry/exit, kernel tracepoints, network events, and several others [24]. If a predefined hook is not already available, it is possible to create a kernel probe (kprobe) or user probe (uprobe) to attach eBPF programs to almost any location in the kernel or user applications.

Once the desired hook is identified, the eBPF program can be loaded into the Linux kernel using the `bpf()` system call, which is done typically using one of the available libraries. The library used during the course of this research project is BCC [26].

2.3.1 Types of programs

eBPF programs are executed when triggered by specific hooks. There are different types of hooks and eBPF programs. In this paper we will work with tracepoints, kprobes and uprobes.

Tracepoints in the Linux kernel allow attaching "probes" to selected kernel functions. Probes are executed every time a designated kernel function is invoked. During the development process, tracepoints are strategically positioned within the kernel. To enable tracing of a particular function, it must be identified with a tracepoint statement, indicating it as a monitored location. This approach restricts the overall quantity of tracepoints that can be utilized [23].

In addition to tracepoints, kprobes and uprobes are also tracing technologies. The key distinction between the two lies in their respective tracing capabilities: uprobes facilitate tracing within user space, whereas kprobes enable tracing within kernel space. Unlike tracepoints, kprobes have the ability to trap nearly any kernel function, eliminating the need for explicit marking of functions for probing purposes, but at the expense of being subject to change across Linux releases. However, it is important to note the existence of a blacklist comprising functions that are not intended to be probed [23].

2.3.2 BCC Framework

BCC is a software framework enabling users to write Python user-space programs with eBPF kernel-space programs written in C embedded inside them. The framework is primarily targeted for use cases which involve application and system profiling/tracing where an eBPF program is used to collect statistics or generate events and a counterpart in user space collects the data and displays it in a human readable form. Running the Python program will generate the eBPF bytecode and load it into the kernel.

2.4 Supervised Machine Learning

Supervised Machine Learning is a fundamental approach for acquiring input-output relationship information in various domains. In supervised learning, a system learns from a given set of paired input-output training samples, where the output serves as the label or supervision for the input data. This labeled training data enables the system to learn the mapping between the input and output, facilitating the prediction of output values for new inputs. If the output consists of discrete values representing class labels, supervised learning leads to classification tasks. On the other hand, if the output comprises continuous values, it pertains to regression tasks. The input-output relationship information is typically represented by learning-model parameters, which are estimated through a learning process when direct access to these parameters is not available. In contrast to unsupervised learning, which relies on unlabeled input data, supervised learning relies on labeled or supervised information to train the learning system effectively. [16]

Supervised Machine Learning encompasses a range of algorithms, including k-Nearest Neighbors (kNN) and Support Vector Machines (SVM) [25, 16].

The k-Nearest Neighbors (kNN) classifier is a non-parametric algorithm that classifies new data points based on their proximity to labeled training samples. It calculates the distance between a new data point and all the labeled training samples in the feature space. The kNN algorithm assigns the majority class label among the k nearest neighbors to the new data point.

Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. SVM aims to find an optimal hyperplane that separates data points

of different classes with the largest margin. It transforms the data into a high-dimensional feature space, where it constructs a decision boundary that maximizes the margin between classes. SVM can handle both linearly separable and non-linearly separable data through the use of different kernel functions.

In this research project we have mainly tested SVM with Linear and Radial Basis Function (RBF) kernels.

3 Related Works

This section presents an overview of prior research and approaches in the field of ransomware detection. Various studies have explored network-based detection systems, dynamic analysis solutions, and machine learning-based approaches. In this section, we summarize the key contributions from these works, highlighting their methodologies and findings. By examining the existing literature, we identify the gaps and opportunities for further exploration. Additionally, we emphasize the unexplored potential of integrating eBPF into the realm of ransomware detection.

The authors in [10] present a network detection system for the Locky ransomware. A testbed is used to run multiple samples of this malware, and then behavioral and non-behavioral traffic features are analyzed, e.g., HTTP-POSTS, MDN, and DNS (IPv4, IPv6). A multi-classifier intrusion detection system (IDS) is then developed using these features to detect packet and flow-level behaviors using a range of Machine Learning algorithms.

The paper in [22] presents a dynamic analysis solution for ransomware detection called UNVEIL. This scheme uses the Cuckoo Sandbox [1] to monitor system and file actions, e.g., persistent desktop messages (API display message calls), selective encryption/deletion of files based on attributes such as size, date, access time, and extension. UNVEIL was able to identify and detect a previously unreported type of ransomware.

In [31], a survey of some key contributions in ransomware detection and classification is presented. The survey categorizes ransomware detection and classification systems into network-based and host-based. It also presents an open challenge regarding ML-based ransomware detection and classification systems: "even though many solutions have been proposed, their practicality in real world settings has not been fully considered (e.g., many users may be unwilling to share their detailed log files (traces) for external analysis)."

The paper in [12] investigated the latest developments in preventing and detecting ransomware, performing a literature study. The authors defined different ransomware detection approaches: analyzing system information, ransom note analysis, file analysis, finite state machines, honeypots, network traffic analysis and Machine Learning. A custom-made experimental ransomware was also created, which was successful in avoiding detection from eight widely-used antivirus programs.

For the past two decades, the security community has been fighting malicious programs for Windows-based operating systems. However, the recent surge in adoption of embedded devices and the IoT revolution are rapidly changing the malware landscape. The authors in [14] performed a large-scale measurement study on over ten thousand malware samples for Linux-based systems in order to characterize, analyze, and understand Linux malware.

To detect ransomware, numerous operations can be tracked. The authors in [19] developed a predictive model of ransomware in an attempt to characterize all variants within each ransomware family into a unified model. They used the Cuckoo sandbox and analyzed API function calls, network traffic, file I/O operations and registration keys.

These existing works demonstrate various approaches and techniques for ransomware detection, ranging from network-based and host-based solutions to dynamic analysis and machine learning-based approaches. However, the integration of eBPF into the ransomware detection landscape remains an unexplored avenue, which we address in this paper.

4 Methodology

In this section, techniques for detecting ransomware using eBPF will be detailed, as well as the solution design and implementation, which consists of a unified mechanism of ransomware detection using eBPF integrated into a Machine Learning pipeline. The environment which allowed ransomware experimentation is also presented in detail.

4.1 Experimental setup

The main goal for this experimental setup was to create an isolated environment preventing any malware lateral movement or data exfiltration, using a Zero Trust security model [2]. The experimental setup used in this research project consists of a double-nested isolated virtualized environment, as per Figure 1. All the virtualization is done using the KVM Hypervisor [3]. The API used to communicate with KVM is libvirt [4], an open-source framework which provides access to management of virtual machines on a host.

Virtualized guests connect to networks via a virtual network, usually through a virtual network switch. The virtual network switch can operate in several modes, of which we are interested in solely two of them for this setup:

- **NAT mode**: the default mode in which the libvirt virtual network switch operates, without additional configuration. All guests have direct connectivity to each other and to the virtualization host. The guests can access external networks by network address translation, subject to the host system's firewall rules. This mode is not secure for ransomware experimentation and, therefore, it is only used for the initial setup phase, which includes the installation of software, ransomware sample download, etc..
- **Isolated mode**: the guest virtual machines can communicate with each other and with the virtualization host, but the traffic will not pass outside of the virtualization host, nor can they receive external traffic. This network type is the one used at times where ransomware is being experimented, ensuring that any potential propagation into the public is not possible.

The fact that we have a double-nested virtualized environment allows us to benefit from the snapshot capability by capturing and saving snapshots of the first layer of virtualization at different times of the experiment. This allows us to properly run multiple ransomware (and benign) samples in multiple rounds from a fresh start at the beginning of each run.

To be able to run the samples on this environment, a first layer of virtualization is created, by provisioning a Ubuntu 20.04 Virtual Machine - *Sandbox Host VM*, fully patched and with a firewall only allows SSH and VNC access from the secure LAN. The second layer of virtualization is created nested inside this VM, giving origin to the isolated zone, where ransomware experimentation is done. To run these, two virtual machines are provisioned:

- Ubuntu 20.04 -*Infected VM*: fully patched and running our ransomware detector, as well as the one which will be used to execute different ransomware samples in each run.

- Ubuntu 20.04 - *REMnux VM*: it is a toolkit used to analyze the internet traffic generate from the ransomware. Using services such as INetSim [29] and FakeNet-NG [6], we can emulate common network services and interact with the ransomware.

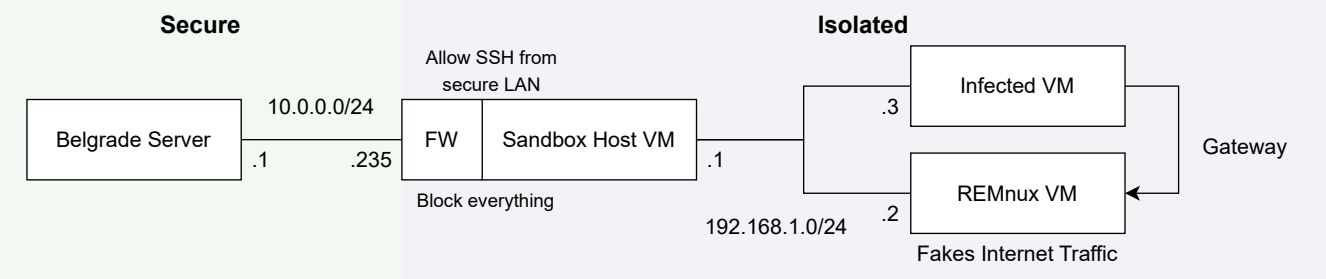

Figure 1: Overview of the secure and isolated lab environment.

4.2 Ransomware detection approaches

Ransomware detection involves analyzing the behavioral traits and patterns exhibited by ransomware. By examining these characteristics, such as file encryption, communication with command and control servers, and suspicious process behavior, it becomes possible to identify potential instances of ransomware attacks. Furthermore, advanced detection methods may leverage machine learning algorithms and anomaly detection techniques to enhance the accuracy and effectiveness of identifying ransomware behaviors [12, 18, 19, 31].

Ransomware detection techniques can be categorized in network-based detection and host-based detection [31]. Network-based detection involves analyzing host traffic to identify potential ransomware activities. Packet data can be collected from infected hosts as well as connected networks. Various types of network traces can serve as indications of ransomware activity, such as DNS queries for C&C server IP addresses and network storage access. Host-based detection entails monitoring the activities occurring within the local system. This type of detection involves examining a wide range of static and dynamic actions, including file and memory operations, API function calls, etc. [10, 12].

In this paper we combine host-based detection (including preliminary analysis and filtering) and machine learning methods. The classifiers presented above play a crucial role in our ransomware detection solution by leveraging their ability to learn from labeled training data and make accurate predictions. By applying these algorithms to the specific characteristics and behaviors of ransomware, we can effectively classify and identify potential threats in real-time. Models can be trained on different features. These can be all kind of processes and operations like: API functions, system calls, network traffic, file I/O operations, log files, etc. [19, 12, 31, 11].

4.2.1 Using eBPF

With its event-driven nature and direct execution within the kernel, eBPF emerges as an optimal technology for runtime ransomware detection. By utilizing kernel-based hooking, eBPF establishes a robust security boundary, significantly enhancing overall system protection. It offers the convenience of triggering eBPF programs through probes in both user and kernel spaces, providing a unified mechanism to intercept and handle all relevant events.

To ensure stability and reliability, eBPF code undergoes thorough verification before insertion into the kernel and operates within a dedicated VM. This approach effectively mitigates the risk of programming errors that could lead to kernel crashes or instability [27]. Moreover, eBPF optimizes system performance and resource utilization by enabling event filtering directly within the kernel, eliminating the need to transmit and process irrelevant events in user space.

In addition to its technical advantages, eBPF facilitates the provision of essential contextual information about events. It captures valuable details such as arguments, timestamps, and other relevant information, enabling comprehensive analysis and empowering ransomware detection systems. Another notable advantage of eBPF is its capability to access and modify data in user space memory. This feature provides flexibility and control during the detection process, allowing for the examination and modification of arguments as needed. The execution of eBPF code within the kernel space, compiled to native code, ensures minimal performance overhead, enabling efficient and swift analysis of events.

It's important to highlight that eBPF is actively maintained as an integral part of the Linux kernel, continuously receiving improvements and the addition of new features. This active development ensures that eBPF remains a powerful and adaptable technology, keeping pace with the ever-evolving landscape of malware detection.

In conclusion, the combination of eBPF's event-driven nature, kernel-level execution, ability to manipulate user space memory, and continuous development as part of the Linux kernel makes it an effective technology for ransomware detection [9].

4.3 Our solution

In this subsection, we present a dynamic analysis detection solution which uses eBPF for tracing system calls and generates events which are sent to a machine learning pipeline which predicts whether or not a certain process is ransomware. The overall software architecture is presented in Section 4.3.1 and the machine learning pipeline is presented in detail in section 4.3.2. All of the source code is publicly accessible in [30].

4.3.1 Architecture Overview

The eBPF detector uses the BCC framework introduced in Section 2.3.2. A simplified diagram of the software architecture is presented in Figure 2 and consists of two parts: kernel and user space programs. In the kernel space, an eBPF C program (`bpf.c`) is attached to various key system calls and generates events. It computes various statistics for each running process, checks threshold crossings and does pattern matching on event sequences. After analysis, these events are made available to the user-space Python program (`bpf.py`) via a BPF Ring Buffer data structure. The user-space program sends event sequence patterns and configuration settings back to the eBPF program to fine-tune and improve the existing detection. These events are used as input into the Machine Learning pipeline (described in Section 4.3.2), which returns patterns and configuration to be sent back to the eBPF program.

The eBPF hooks are:

- **Tracepoints**:
  - `open()`/`openat()` - Detect file open/read/write during encryption.
  - `unlink()`/`unlinkat()` - Detect backup/file spoliation.
- **uprobes**: (for crypto-related shared libraries)

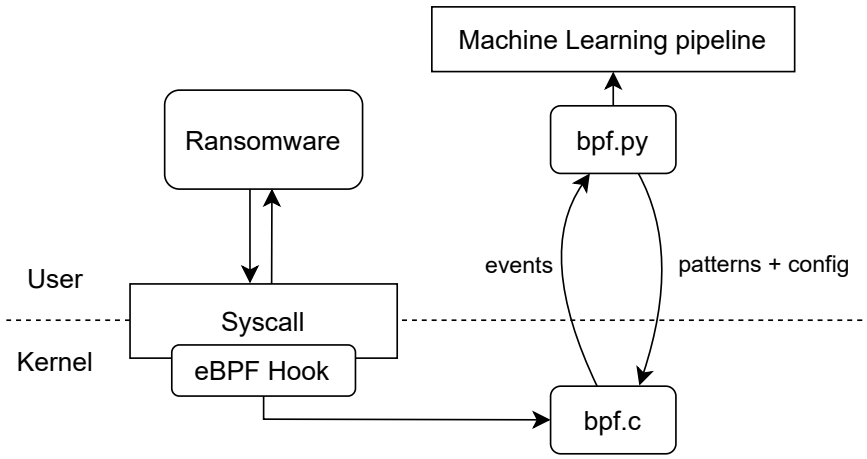

Figure 2: Simplified Software Architecture. Note that this diagram omits the usage of uprobes, which aren't hooked onto system calls, but to user-space libraries.

- `EVP_EncryptInit_ex` - Detect encryption.
- `EVP_CipherInit_ex` - Detect encryption.
- `EVP_SealInit_ex` - Detect encryption (cipher context initialization).

By attaching the hooks to key system calls and user-space libraries, our detector effectively captures and analyzes the behavior of processes, allowing for the establishment of access patterns. These hooks are strategically placed to intercept important operations such as file open, read, write, backup, and spoliation. The events generated by our eBPF hooks serve as valuable indicators of process behavior, providing insights into potential ransomware activities.

On each event, the kernel-space eBPF program computes statistics per process and detects event patterns and threshold counts, which are both provided by the Python frontend program. After this, the eBPF program filters out irrelevant events, thereby only submitting relevant ones to a ringbuffer - a shared data structure which the Python frontend program can access. This process is pictured in Figure 3.

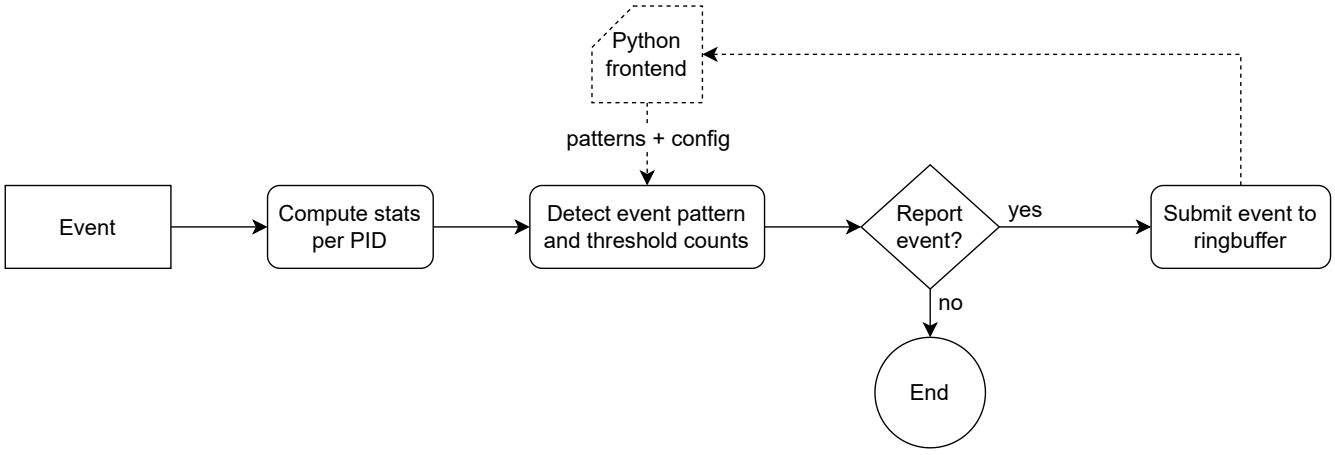

Figure 3: eBPF program event reporting overview.

To process and utilize this information, the generated events are transmitted to the Python frontend program. Here, the events are labeled and curated to form a dataset for training the

Machine Learning model. By incorporating this labeled data, the pipeline can learn and recognize the distinctive patterns and behaviors associated with ransomware.

This seamless collaboration between the eBPF hooks, Python frontend program, and Machine Learning pipeline forms a robust and efficient framework for ransomware detection, enabling real-time monitoring and proactive defense against evolving threats.

4.3.2 Machine Learning pipeline

The machine learning pipeline follows a simple flow presented in Figure 4. In Figure 5, the last three steps are presented in more detail.

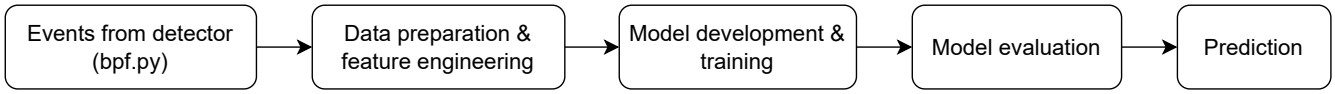

Figure 4: Machine Learning pipeline.

Events are received from the Python detector frontend program, which formats them into a `.csv` consisting of a row per event with columns that are the features used for our classifier to base its predictions on. The dataset input format which results from this step is described in Table 1.

Table 1: Dataset input format.

Name	Description
Timestamp	Event generation timestamp
PID	Process ID for a given process
Type	Type of the event
Severity	According to thresholds crossed and pattern matches
Pattern	ID of any pattern matched by eBPF
EventTypesCross[]	For each type, whether the number of events of each crossed the threshold
Filename	Name of the file or name of the encryption function

Based on the events received, the following features can be computed and normalised. This step corresponds to the Data preparation & feature engineering step, which is performed by the `dataprep.py` program:

- For each type of event:
  - Average number of events per second;
  - Maximum number of events per second (rate);
  - Total number of events.
- For each possible event patterns:
  - Average number of pattern matches per second;
  - Maximum number of patterns matches per second (rate);
  - Total number of pattern matches.

- Type of files accessed:
  - Sensitive Linux files (e.g., `/etc/`, `/var/`, `/usr/`, `/sys/`), especially if they are modified or created.

A sample output can be viewed in Appendix A.

For the next step, we considered simple classifiers like kNN (k-Nearest Neighbors) and SVM (Support Vector Machine) with different kernels [25]. SVM works particularly well for our dataset and purpose, since it is very effective in high-dimensional spaces and relatively memory efficient [20].

The processed data is split into training and testing data, following a 60/40% split. This means that the resulting Machine Learning model is validated using data that wasn't used for training.

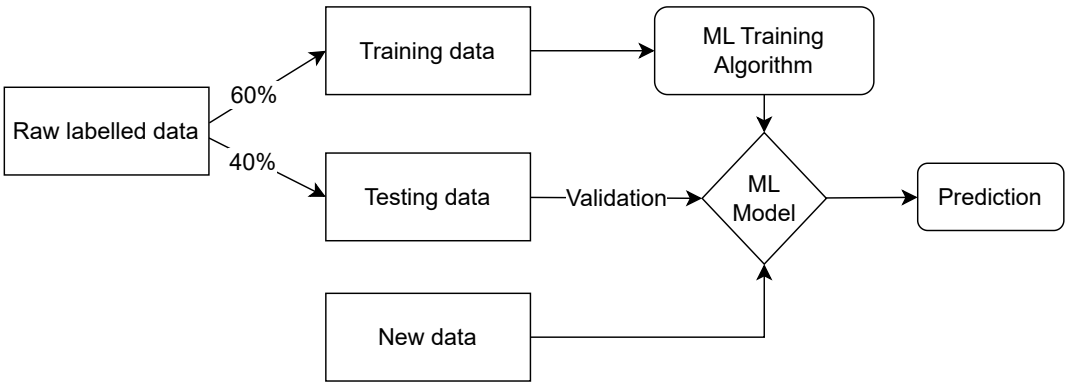

Figure 5: Model development & training, evaluation and prediction steps detailed.

When new data is processed through the model, it will assign a label depending on whether it is classified as ransomware or benign. In the following section, we will look at the results achieved by our solution.

5 Results

In this section, we present the findings and outcomes of our experiments on ransomware detection. We discuss the dataset used, including its size and composition, as well as the importance of different features in distinguishing ransomware behavior. Furthermore, we evaluate the performance of our detection solution using various metrics and analyze the classification results. These results shed light on the effectiveness of our approach and provide insights into the strengths and limitations of our ransomware detection system.

5.1 Dataset & Feature Importance

The dataset used in our experiments consists of around 1.5 million events, containing 2786 different processes, of which 53 were ransomware samples. The dataset was split according to Table 2.

Table 2: Dataset split in training and testing data.

	Training data	Testing data	Total
Events	639,720	833,636	1,473,356
Benign processes	1,416	1,317	2,733
Ransomware	32	21	53

All of the ransomware samples used for experimentation were obtained from MalwareBazaar, a project whose purpose is to collect and share malware samples [5]. The ransomware families successfully tested include the following families:

- IceFire
- MONTI
- REvil
- AvosLocker
- BlackMatter
- HelloKitty

Although many more ransomware families were tested, most of them either detected that the system they were being executed on was a virtual machine, weren't compatible with our operating system, or requirements for running them weren't met (e.g., missing shared libraries).

Figure 6 represents the importance, or weight, of each feature. It assigns each feature a given weight based on its relevance in predicting the output.

In our case, the most important features revolve around specific patterns of file operations, namely the sequences of "Open", "Create" and "Delete" actions.

Among these patterns, the top three influential features are:

1. **Open, Create, Open** (OCO): This pattern represents the sequence of opening a file, creating a new file, and then opening another file. It signifies a specific behavioral pattern associated with ransomware activity.

2. **Create, Open, Create** (COC): This pattern involves creating a file, opening an existing file, and then creating another file. It captures a distinct sequence of file operations commonly exhibited by ransomware samples.

3. **Create, Open, Open** (COO): This pattern signifies the sequence of creating a file, opening an existing file, and then opening another file. It represents another behavioral pattern that helps identify potential ransomware activities.

These patterns play a crucial role in differentiating ransomware behavior from normal system operations. Their high importance suggests that they provide strong discriminatory power for effective ransomware detection.

Additionally, other important features include the number of delete actions (D_sum) and the maximum number of delete actions (D_max). These features capture the frequency and intensity of file deletion operations, which are often indicative of ransomware behavior.

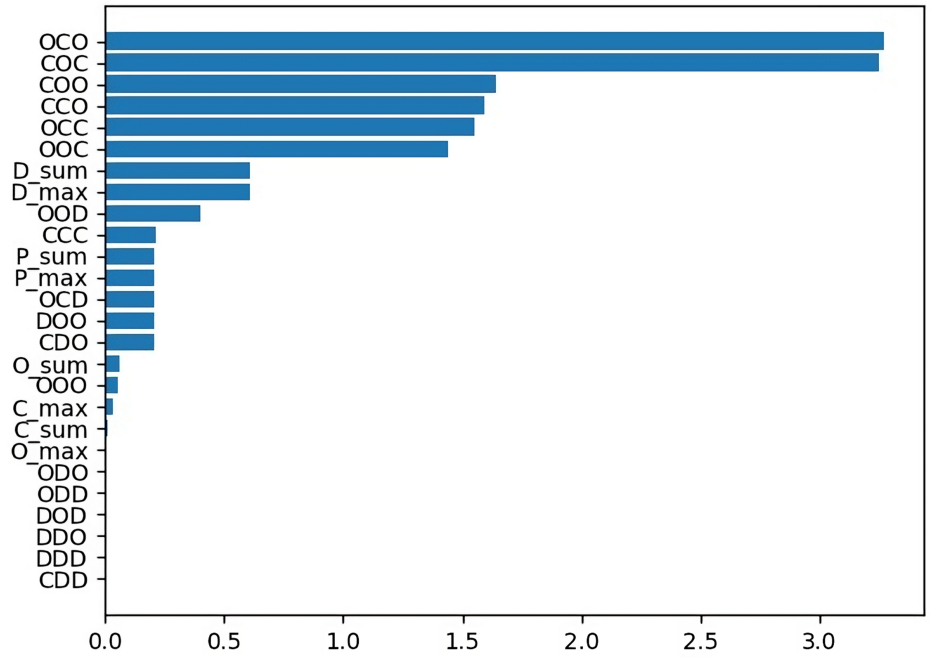

Figure 6: Feature importance chart.

5.2 Evaluation

In this subsection, we will specify the metrics used to evaluate the performance of our ransomware detection solution, showcasing its ability to correctly identify and classify ransomware instances from the dataset. Table 3 corresponds to the confusion matrix, which summarizes the results achieved by our classifier. The classifier was trained to optimize recall rather than precision since the cost of a False Negative (undetected malware) is much more than the cost of a False Positive (false alert). The accuracy of the model can be determined by observing the diagonal values. For these results, the Radial Basis Function (RBF) kernel SVM algorithm was used.

		Predicted	
		Ransomware	Benign
Actual	Ransomware	21	0
	Benign	23	1294

Table 3: Confusion matrix.

According to the confusion matrix, some important metrics can be derived:

- **True Positive Rate (Recall)** $= \frac{TP}{TP+FN} = 100\%$.
- **Matthews' Correlation Coefficient** $= \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} = 68.48\%$.
- **Precision** $= \frac{TP}{TP+FP} = 47.73\%$.

In our case, we observe zero false negatives, indicating that our system did not classify any ransomware processes as benign. This result is highly desirable as it demonstrates the effectiveness of our approach in accurately detecting potential ransomware threats. We observe 23 false positives, indicating that our system is sensitive and on the cautious side. However, this can be also desirable in order to maximize the detection rate.

6 Discussion

In this section, we analyze and discuss the key findings and implications of our ransomware detection system. We delve into the strengths, limitations, and potential areas for improvement, providing valuable insights into the practical application and effectiveness of the proposed approach. Through critical evaluation, we aim to foster a deeper understanding of the system's performance, its impact on real-world ransomware detection scenarios, and avenues for future research and development.

6.1 Comparison with existing solutions

The obtained results of our ransomware detection system demonstrate promising performance and effectiveness. When comparing our solution to existing approaches, it is evident that our system achieves excellent detection rate and recall. In particular, the absence of false negatives indicates the robustness of our approach in correctly identifying and classifying ransomware processes. However, the presence of false positives suggests the need for further improvements to minimize the misclassification of benign activities as ransomware.

However, this system was not yet tested against novel ransomware, meaning that the results might still be heavily skewed due to the dataset incompleteness, as discussed in the following subsection.

6.2 Limitations

While our ransomware detection system shows promising results, it is important to acknowledge certain limitations that should be considered:

1. **Imbalanced Dataset**: The imbalance between ransomware samples and benign samples in the dataset can impact the performance of the detection system. With only a small number of ransomware samples compared to benign samples, the system is be biased towards the majority class, leading to potential challenges in accurately detecting rare or novel ransomware variants.

2. **Generalization to unknown ransomware**: Our system's performance is based on its ability to detect known ransomware samples. However, its effectiveness in identifying unknown or zero-day ransomware variants may be limited. As ransomware continues to evolve rapidly, with new evasion techniques and obfuscation methods, the system may struggle to detect previously unseen or sophisticated ransomware strains.

3. **Limited to Linux**: Our ransomware detection system is specifically designed for Linux-based systems. As a result, its applicability to other operating systems, such as Windows or macOS, is limited. Adapting the system to different platforms would require additional development and customization to account for platform-specific behaviors and characteristics.

It is important to address these limitations and continue research and development efforts to improve the system's performance, robustness, and adaptability. Mitigating these limitations will contribute to the overall effectiveness of the ransomware detection system and ensure its reliability in real-world scenarios.

In addition to the integration of eBPF with our machine learning pipeline, we have designed and developed an open-source codebase for our ransomware detection system [30]. This codebase serves as a valuable resource for the security community, allowing researchers and practitioners to utilize and further enhance our detection capabilities. By making our code open source, we promote transparency, collaboration, and knowledge sharing in the field of ransomware detection. The availability of this codebase enables others to reproduce our experiments, validate our findings, and build upon our work to develop more robust and effective ransomware detection solutions.

7 Conclusion

By providing the capability to hook into various points within the kernel and userspace for tracing and processing diverse events, eBPF emerges as an suitable choice for ransomware detection. To address this need, we have developed our own dynamic analysis detection solution, nicknamed *ebpfangel* [30]. Within this solution, we have leveraged eBPF by strategically attaching our program to the key system calls in the kernel via *tracepoints*, enabling us to identify important operations such as open, read, write, and detect file tampering. Moreover, we have also utilized *uprobes* to hook our program into the OpenSSL library functions in userspace, facilitating the detection of encryption activities.

By passing the relevant events patterns and threshold count to the python frontend program where after they are labeled to form a training dataset for the Machine Learning model. This dataset contains carefully selected and relevant features which play a crucial role in the classification process. Based on these events, we computed and normalized different features, This step allowed us to extract valuable insights from the data.

Our study utilized and created a dataset which consists of approximately 1.5 million events. Among these events, the sequences of "Open", "Create" and "Delete" actions emerged as the most vital features. The distinctive patterns exhibited by these file operations played a crucial role in discerning ransomware behavior from normal system operations. Notably, no instances of encryption activities were detected during our analysis. An encouraging finding is that our solution achieved zero false negatives, indicating that ransomware was not misclassified as benign. However, we did observe 23 false positives, suggesting that some benign processes were incorrectly identified as ransomware. The MCC metric of 68.48% also addresses this issue and minimizing false positives will be a focus for future improvements, as it is crucial to strike a balance between accurate detection and avoiding misclassifications.

7.1 Contribution and Practical Relevance

Our research makes several significant contributions to the field of ransomware detection. Firstly, while our primary objective was not to develop a new state-of-the-art ransomware detection solution, we aimed to showcase the potential of eBPF in automating the detection and potential termination of ransomware processes. By conducting experiments in a controlled and isolated environment, prioritizing the safety and well-being of all stakeholders involved, we have established a robust methodology that can serve as a valuable resource for future studies in ransomware research. Additionally, the integration of eBPF with machine learning techniques has emerged as a true contribution to the field of ransomware detection. This combination enhances the precision and efficiency of our detection system, enabling it to accurately classify ransomware instances with minimal false positives. This integration represents a practical and effective solution, leveraging

the capabilities of eBPF and machine learning in a synergistic manner, thus advancing the state of the art in ransomware detection.

While our solution is a byproduct of the performed study. It is important to emphasize that neither our study nor the developed solution is intended as a final product for deployment in any production environment. We must acknowledge that our solution has certain critical limitations, preventing us from establishing a definitive verdict regarding the practical relevance. However, it is worth highlighting that our study does hold value as supportive and relevant research in the domain of creating dynamic ransomware solutions utilizing eBPF and machine learning. Our finding contribute to the growing body of knowledge in this field, offering insights which may guide further advancements in ransomware detection and mitigation techniques.

8 Future work

Future work in the field of ransomware detection combined with eBPF and machine learning holds significant potential for further advancements. While our research has made notable contributions, there are several areas that warrant exploration and improvement.

One important aspect is the expansion of the dataset used for experimentation. A larger and more diverse dataset, incorporating a broader range of features and including a greater number of ransomware and benign samples, would enhance the effectiveness and accuracy of the detection system. By incorporating more varied and representative data, future studies can provide more robust and reliable results.

Additionally, incorporating additional detection features and techniques can further enhance the detection capabilities. While our study focused on host-based detection, it is crucial to also consider network-based detection. Analyzing network traffic, particularly interactions with command and control servers, can provide valuable insights into ransomware behavior and improve detection accuracy. Exploring advanced anomaly detection algorithms and integrating them into the machine learning pipeline can also contribute to more effective detection mechanisms.

When considering real-world deployment, it is important to address the feasibility and challenges associated with integrating the ransomware detection system into practical settings. Scalability, system integration, and practical implementation strategies need to be carefully considered to ensure the solution can be effectively deployed and maintained in diverse environments. Furthermore, given that ransomware attacks commonly target Windows systems, it would be worthwhile to investigate the applicability of eBPF solutions in such scenarios, particularly considering the recent availability of eBPF for Windows systems.

Considering the evolving landscape of ransomware attacks and the emergence of new threat vectors and ransomware variants, it is crucial to continuously adapt and update the detection system. Regular updates and enhancements will ensure the system remains effective in mitigating the latest threats and can effectively identify and counter new attack techniques.

To foster progress in the field of ransomware detection, collaboration and knowledge sharing within the research community are essential. Encouraging open collaboration, sharing of datasets, methodologies, and findings will enable researchers to collectively combat the growing threat of ransomware attacks. By fostering collaboration, we can pool resources and expertise to develop more advanced and robust ransomware detection techniques, ultimately enhancing the overall security posture against this persistent threat.

References

[1] Cuckoo sandbox - automated malware analysis. https://cuckoosandbox.org/.

[2] Evolving zero trust – microsoft position paper. URL: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWJJdT.

[3] Kvm. URL: https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792.

[4] libvirt: The virtualization api. URL: https://libvirt.org/.

[5] Malwarebazaar — malware sample exchange. https://bazaar.abuse.ch/.

[6] mandiant/flare-fakenet-ng: Fakenet-ng - next generation dynamic network analysis tool. URL: https://github.com/mandiant/flare-fakenet-ng.

[7] The state of ransomware 2022. URL: https://www.sophos.com/en-us/content/state-of-ransomware.

[8] What is eBPF? an introduction and deep dive into the eBPF technology. URL: https://www.ebpf.io/what-is-ebpf/.

[9] Yaniv Agman and Danny Hendler. BPFroid: Robust real time android malware detection framework. URL: http://arxiv.org/abs/2105.14344, arXiv:2105.14344[cs].

[10] Ahmad Almashhadani, Mustafa Kaiiali, Sakir Sezer, and Philip Okane. A multi-classifier network-based crypto ransomware detection system: A case study of locky ransomware. *IEEE Access*, PP:1–1, 03 2019. doi:10.1109/ACCESS.2019.2907485.

[11] Seong Il Bae, Gyu Bin Lee, and Eul Gyu Im. Ransomware detection using machine learning algorithms. 32(18):e5422. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5422. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5422, doi:10.1002/cpe.5422.

[12] Craig Beaman, Ashley Barkworth, Toluwalope David Akande, Saqib Hakak, and Muhammad Khurram Khan. Ransomware: Recent advances, analysis, challenges and future research directions. 111:102490. URL: https://www.sciencedirect.com/science/article/pii/S016740482100314X, doi:10.1016/j.cose.2021.102490.

[13] Enrico Bocchi, Luigi Grimaudo, Marco Mellia, Elena Baralis, Sabyasachi Saha, Stanislav Miskovic, Gaspar Modelo-Howard, and Sung-Ju Lee. Magma network behavior classifier for malware traffic. *Computer Networks*, 109:142–156, 2016. Traffic and Performance in the Big Data Era. URL: https://www.sciencedirect.com/science/article/pii/S1389128616300949, doi:https://doi.org/10.1016/j.comnet.2016.03.021.

[14] Emanuele Cozzi, Mariano Graziano, Yanick Fratantonio, and Davide Balzarotti. Understanding linux malware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 161–175, 2018. doi:10.1109/SP.2018.00054.

[15] Sergiu Gatlan. IceFire ransomware now encrypts both linux and windows systems. URL: https://www.bleepingcomputer.com/news/security/icefire-ransomware-now-encrypts-both-linux-and-windows-systems/.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] M. Gopinath and Sibi Chakkaravarthy Sethuraman. A comprehensive survey on deep learning based malware detection techniques. 47:100529. URL: https://www.sciencedirect.com/science/article/pii/S1574013722000636, doi:10.1016/j.cosrev.2022.100529.

[18] Anand Groenewegen, Majid Alqabandi, Mohanad Elamin, and Pim Paardekooper. *A behavioral analysis of the ransomware strain NEFILIM*. doi:10.13140/RG.2.2.18301.59360.

[19] Gavin Hull, Henna John, and Budi Arief. Ransomware deployment methods and analysis: views from a predictive model and human responses. 8(1):2. doi:10.1186/s40163-019-0097-9.

[20] Padmavathi Janardhanan, L. Heena, and Fathima Sabika. Effectiveness of support vector machines in medical data mining. *Journal of Communications Software and Systems*, 11:25–30, 04 2015. doi:10.24138/jcomss.v11i1.114.

[21] Ilker Kara and Murat Aydos. The rise of ransomware: Forensic analysis for windows based ransomware attacks. 190:116198. URL: https://www.sciencedirect.com/science/article/pii/S0957417421015141, doi:10.1016/j.eswa.2021.116198.

[22] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. Unveil: A large-scale, automated approach to detecting ransomware. 08 2016.

[23] Alonso Arenaza Lucia and Max Willers. Security of ebpf. 02 2023.

[24] Virag Mody. A gentle introduction to eBPF. URL: https://www.infoq.com/articles/gentle-linux-ebpf-introduction/.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] IO Visor Project. BCC. URL: https://liuhangbin.netlify.app/post/ebpf-and-xdp/.

[27] Liz Rice. *What Is eBPF?* URL: https://learning.oreilly.com/library/view/what-is-ebpf/9781492097266/.

[28] Veronika Szücs, Gábor Arányi, and Ákos Dávid. Introduction of the ARDS—anti-ransomware defense system model—based on the systematic review of worldwide ransomware attacks. 11(13):6070. URL: https://www.mdpi.com/2076-3417/11/13/6070, doi:10.3390/app11136070.

[29] Matthias Eckert Thomas Hungenberg. Inetsim: Internet services simulation suite. URL: https://www.inetsim.org/index.html.

[30] Max Willers Tomás Philippart. ebpfangel. https://github.com/TomasPhilippart/ebpfangel, 2023.

[31] Aldin Vehabovic, Nasir Ghani, Elias Bou-Harb, Jorge Crichigno, and Aysegul Yayimli. Ransomware detection and classification strategies. 04 2022.

[32] Lena Yuryna Connolly, David S Wall, Michael Lang, and Bruce Oddson. An empirical study of ransomware attacks on organizations: an assessment of severity and salient factors affecting vulnerability. 6(1):tyaa023. doi:10.1093/cybsec/tyaa023.

Appendix

A Sample output from dataprep.py

```
PID       C_max  C_sum  D_max  D_sum  E_max  E_sum  O_max  O_sum  P_max  P_sum   CDO  COC  COO   DOC  DOO   EEE   OCD
↪  OCO  OOC  OOO
1           0      0      0      0      0      0      4      5      0      0      0    0    0     0    0     0     0
↪  0    0    3
221         0      0      0      0      0      0     20     20      0      0      0    0    0     0    0     0     0
↪  0    0   18
626         0      0      0      0      0      0     16     16      0      0      0    0    0     0    0     0     0
↪  0    0   14
714         0      0      0      0      0      0     12     12      0      0      0    0    0     0    0     0     0
↪  0    0   10
838         0      0      0      0      0      0      3      9      0      0      0    0    0     0    0     0     0
↪  0    0    7
1019        0      0      0      0     36   1501      0      0      0      0      0    0    0     0    0  1499     0
↪  0    0    0
1195        0      0      0      0      0      0     11     11      0      0      0    0    0     0    0     0     0
↪  0    0    9
1196        0      0      0      0      0      0      7      7      0      0      0    0    0     0    0     0     0
↪  0    0    5
1197        0      0      0      0      0      0     10     10      0      0      0    0    0     0    0     0     0
↪  0    0    8
1202        0      0      0      0      0      0      8      8      0      0      0    0    0     0    0     0     0
↪  0    0    6
1230        0      0      0      0      0      0      1     60      0      0      0    0    0     0    0     0     0
↪  0    0   58
1237        0      0      0      0      0      0      1     36      0      0      0    0    0     0    0     0     0
↪  0    0   34
1238        0      0      0      0      0      0      1     27      0      0      0    0    0     0    0     0     0
↪  0    0   25
1239        0      0      0      0      0      0      1     38      0      0      0    0    0     0    0     0     0
↪  0    0   36
1240        0      0      0      0      0      0      1     30      0      0      0    0    0     0    0     0     0
↪  0    0   28
1555        0      0      0      0    207   1978      0      0      0      0      0    0    0     0    0  1976     0
↪  0    0    0
29952       0      0      0      0     63    326      0      0      0      0      0    0    0     0    0   324     0
↪  0    0    0
30490      17   1690     17   1691      0      0     20   1750     17   1690   1689    0    0  1635   55     0  1690
↪  0   55    5
30493       0      0      0      0      0      0     12     19      0      0      0    0    0     0    0     0     0
↪  0    0   17
30495     233   1216     17    983      0      0    278   1458     17    983    983  228    5   949   34     0   983
↪  233   39  201
30496       0      0      0      0      0      0      3      3      0      0      0    0    0     0    0     0     0
↪  0    0    1
```