User Behavior analysis for Malware detection*

Valentina Dumitrasc[1] and René Serral-Gracià[2][0000−0003−2112−0952]

[1] FSP Consulting Services
tina.dumitrasc@fsp.co
[2] BarcelonaTech
rene.serral@upc.edu

Abstract. The rise in cyber-attacks and cyber-crime is causing more and more organizations and individuals to consider the correct implementation of their security systems. The consequences of a security breach can be devastating, ranging from loss of public confidence to bankruptcy. Traditional techniques for detecting and stopping malware rely on building a database of known signatures using known samples of malware. However, these techniques are not very effective at detecting zero-day exploits because there are no samples in their malware signature databases.
To address this challenge, our work proposes a novel approach to malware detection using machine learning techniques. Our solution provides a two-fold contribution, on the one hand, our training the model does not require any kind of malware, as it creates a user profile using only normal user behavior data, detecting malware by identifying deviations from this profile. On the other hand, as we shall see, our solution is able to dynamically train the model using only six sessions to minimize false positives. As a consequence, our model can quickly and effectively detect zero-day malware and other unknown threats without previous knowledge. The proposed approach is evaluated using real-world datasets, and different machine learning algorithms are compared to evaluate their performance in detecting unknown threats. The results show that the proposed approach is effective in detecting malware, achieving high accuracy and low false positive rates.

Keywords: Machine Learning, Malware detection, User Behavior Analysis, Autoencoder

1 Introduction

The increase in cyber-attacks is a growing concern for individuals, organizations and governments as it leads to several problems, including: security breaches, financial losses, reputational damage, disruption to business operations and threats to critical infrastructure. This highlights the need for organizations to implement robust cybersecurity measures to protect against cyber attacks and minimise the risks associated with them.

Cybercrimes are constantly increasing. For example in 2020 [7], malware attacks increased 358% compared to 2019. From here, cyber attacks globally increased by 125% through 2021, while increasing volumes of such attacks continued to threaten businesses and individuals during 2022.

Apart from this increase in cyber-attacks, it is important to notice that these are normally focused on endpoints, such as computers, laptops, smartphones, etc. The reason behind this is that they are seen as the weakest link in an organization's security as they may have less security

*This work was partially funded by IRIS Artificial Intelligence Threat Reporting and Incident Response System (H2020-101021727)

measures than other parts of the network, while providing a door for expanding to other assets of the victim, such as servers. By compromising an endpoint, attackers can steal sensitive information or spread malware to other parts of the network. Therefore it is important to detect a compromised device as soon as possible. This can be achieved through the deployment of intrusion detection systems, which focus on detecting multiple types of malware. Despite of this, none of them is fully prepared to detect new or unknown threats.

Nowadays, detecting malware relies on analyzing known malware code and behavior to build a database of signatures. But this approach has become limited in identifying new, quickly-evolving polymorphic and metamorphic malware, making it difficult to keep the database updated.

In order to mitigate this, we propose a novel Machine Learning based solution that is able to detect malware threats by modeling user behavior. In more detail, our proposal provides, as a first novelty, a solution which does not need actual malware samples or infected systems, on the contrary, we provide a real implementation of a system able to understand and profile user behavior, to then monitor, detect deviations and flag anomalous system behavior.

As a second contribution of this work, our model is able to converge to proper detection rates with a very short training period, which in most times is below 6 hours, providing, as we will discuss later, very high accuracy in our real world testing scenario.

When dealing with machine learning models, the testing and the obtained results need to be trustworthy, as there is no perfect way to validate its full accuracy. To mitigate this uncertainty we installed our prototype in a couple of machines, which were used by real users in their day-to-day work and leisure time, in such an environment, we started gathering data for training, later in the process, we infected the machines with real malware and validated its proper detection.

The rest of the paper is structured as follows, in Section 2 we discuss about the related work and similar solutions found in the literature, later on, in Section 3 we outline our general solution and its architecture. Then, in Section 4 we dig into our Machine Learning model and evaluate its internal workings. This leads to our evaluation and analysis of the obtained solution in our real world scenario in Section 5. We finally conclude in Section 6 with a summary of our findings and outline of our future work.

2 Related Work

Historically, malware was analyzed by directly getting the malware binary file and analyzing its contents, this was, and still is, done mainly through Static and Dynamic code analysis. This approach, albeit if useful to understand how the malware works and to generate signatures [4] that allow fast and efficient malware detection, has been found unsuitable to detect the newest malware types present nowadays, such malware normally uses polymorphism or other well-known techniques to change its payload, thus rendering the signature matching unusable.

To overcome this limitation, malware analysis experts have devised other mechanisms for malware analysis, one of the most used has been Heuristic-based malware detection [10]. A technique based on identifying the characteristics and behavior of a piece of binary code, it relies on setting up a set of rules and heuristics to classify programs as malicious or non-malicious. The huge advantage of this solution over signature based detection is the fact that it allows the detection of polymorphic malware on top of the detection of previously unknown malware, i.e., 0-days. Despite of the benefits these solutions tend to have a relatively high rate of false positives due to the diversity of malware and legitimate applications.

On a different scope, we have observed other solutions, e.g., [6], that have decided to use a Machine Learning approach to malware analysis and detection, the outcomes of such alternatives is a better accuracy than Heuristic based detection.

All the discussed solutions so far are what is called malware centric, where the detection is based on the analysis of malware signatures, or in some cases on its behavior. Opposed to this, our approach, takes malware detection to a different level. Instead of focusing on a malware-centric approach, our contribution focuses on a user-centric approach in which user behavior is modeled into a profile, we leverage information such as system, network or file activities. Then, our model detects deviations from a baseline that could indicate malware. An important benefit of this approach is that our user-centric solution does not require any prior knowledge of a specific malware's behavior or signature. We solely rely on the user's normal behavior as a baseline.

The idea behind this technique is that normally when system's vulnerabilities are exploited, an abnormal use of the system is observed. As a result, abnormal patterns of system usage could detect security threats that would not be detected with other malware detection techniques.

Even if not common, this approach to the problem has already been studied in the literature, for example [8] the authors follow a similar approach, but the main difference in this case is the fact that the authors base their work by inspecting only the network traces, which greatly limits the type of malware they are able to detect. In [11] or [5] the authors follow a similar approach. The shortcomings we observe on such work is that they provide limited information about the features used and their results. Apart from that, the majority of these studies are based on statistical models, thresholds, and averages of a single metric and very few of them incorporate machine learning techniques into the profile creation. One exception to that is [9], where the used system information is very limited and only includes UNIX shell commands with their arguments, focusing only on detecting system intrusion.

3 Architecture

Given the heterogeneity of data sources, data gathering and normalization is a very well-known problem. In this section we describe the approach we propose to balance such complexity with the minimal data set necessary by our user behavior analysis system. This is specially important given that our system will be running on desktops, laptops or other appliances where resource consumption is important for the users.

In Figure 1 we provide the different building blocks of our system. In a nutshell our proposal requires three different data sources:

- *System stats*: the gathered information, as we shall discuss in detail later on, refers to system metrics such as CPU usage, list of processes, ...
- *Network stats*: networking information is critical in nowadays systems. Despite of this, in this first iteration of the proposal we limit our analysis to amount of sent and received data and packets.
- *Audit Logs*: we leverage on auditing tools to receive insights about system status.

3.1 Enduser host

Endpoint agent: is the agent used to obtain system information from the host. It is in charge of aggregating and buffering the different local events that will be sent to the Smart Behavior Analysis

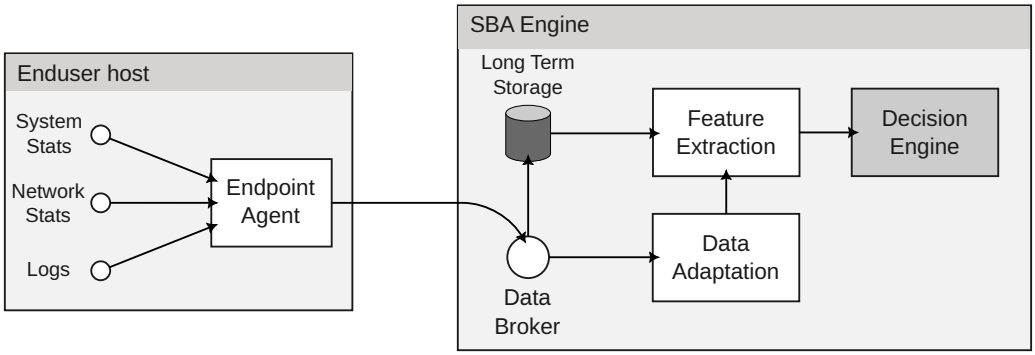


Fig. 1: Overall Architecture building blocks

module. The recollected metrics, as already stated, are System, Network stats and audit logs. We defer the discussion about the different gathered stats to section 3.3.

3.2 Smart Behavior Analysis (SBA)

This module is in charge of gathering all the different stats, to process them and to determine if the current state represents a potential anomaly due to malware activity.

Data Broker: When the information is extracted from the Endpoint Agent, it is reconciled into the Data Broker which has a three-fold goal, first coordinates the reception of data from different Endpoint Agents, second, it stores the raw information into persistent storage for later recovery in case of need, and third feeds the data to the data normalization module.

Data Adaptation: this module unifies the received data, scaling or transforming it when needed. This is necessary for some Machine Learning models improve the obtained results.

Feature Extraction: this module allows us to combine the historical information of this system and the current obtained data to extract the necessary processed metrics for the evaluation. We further discuss this logic in 3.3.

Decision Engine: By applying the ML algorithm, as described later in Section 4, the system is able to determine whether there is malware in action or not. In case of positive outcome, we leave as an important part of our future work to determine the mitigatory actions to perform.

3.3 Metrics

This section focuses on the system metrics used as features to train our model. The metrics are derived from the above-mentioned data sources, namely, system metrics and processes, network metrics, and audit logs. These metrics are periodically collected and assist in the detection of

security risks when their values deviate from the normal behavior for an specific user. This deviation is detected by using machine learning models on these metrics, we can create a model that can automatically identify any potential problems, more specifically, malware. Now we will examine each metric category, exploring their individual components and how they can be effectively used as features in machine learning models. It is worth noticing that our model considers all these metrics as a whole when profiling the user.

System Metrics and processes: this group of metrics are related to CPU, memory usage and the existing processes in the system.

By monitoring these metrics, we can gain insights about the usual system performance depending on the user actions.

We consider the following metrics groups:

- **CPU usage**: This group of metrics provides information about CPU usage, which can help detect potential malware. The types of CPU usage we identify map to the classical metrics provided by nowadays systems. Hence, we analyze the CPU percentages of the following type:
  - **User**: Time running un-niced user processes, which are processes that have not been assigned a specific priority level or "niceness" value.
  - **System**: Time running kernel processes.
  - **Idle**: Time spent in the kernel idle handler.
  - **I/O Wait**: Time waiting for I/O completion.
  - **Nice**: Time running niced user processes.
  - **Software Interrupt**: Time spent servicing software interrupts.
  - **Harwdare Interrupt**: Time spent servicing hardware interrupts.
  - **Steal**: Time stolen from this VM by the hypervisor.
- **Memory usage**: This group of metrics provides information about memory usage, which can help identify potential malware. In this case we consider the following aspects of memory usage:
  - **Total memory**: Amount of physical memory in the system
  - **Used memory**: Amount of physical memory used in the system
  - **Free memory**: Amount of free physical memory in the system
  - **Buffer/Cache memory**: Amount of memory that is currently being used caching
  - **Total swap**: Amount of swap space in the system
  - **Used swap**: Amount of swap space used in the system
  - **Free swap**: Amount of free swap space in the system
- **Processes**: This section focuses on a set of metrics related to processes. In this case our goal is to analyze high level process statistics such as the amount of processes present on a system, but in the future we plan to use the process list names as it may prove a very good indicator of malware.

  Many types of malware may increase the number of running processes in the system, for example, File infecting viruses. These types of malware infect executable files on a victim's system and create multiple instances of themselves in memory, which can cause a significant increase in the number of running processes. Apart from that, any type of malware could increase the number of processes when performing their respective malicious activities.

  We identify the following metrics in this group:
  - **Total Processes**: Total number of running processes at a specific time.
  - **Userland Processes**: Total number of non kernel processes that are running at a specific time.
  - **Kernel Processes**: Total number of kernel processes that are running at a specific time.

Network Metrics: Nowadays almost all attacks use the network. As a consequence, this type of metrics are critical for our system. At a high level, network metrics such as the number of packets/bytes sent and received and the geographical location of new connections provide valuable information about the user. These metrics can be used to help detect various security threats, including DDoS.

Then, related with the network we identify the following metrics:

- **Amount of information**: This group of metrics is related to the amount of transmitted data over the network, specifically the number of bytes sent and received. These metrics can be used to detect unusual traffic patterns or abnormal levels of data transfer, which may indicate the presence of malicious activity.
  We distinguish in this group "amount of transmitted data" and "amount of packets per time unit". While both packet and byte metrics are useful in identifying potential security threats, they provide information that can be used in different ways. For example, the byte metrics may be more useful in identifying unusually large amounts of data being transferred, which could indicate data exfiltration. On the other hand, the packet metrics may be more useful in identifying unusual network traffic patterns, such as a sudden increase in the number of packets being sent from a particular source, which could indicate DDoS or command-and-control attacks.
- **Endpoint country**: This group of metrics provides information about the number of new network connections originating from each continent, which are well-known major sources of cyber-attacks such as China and Russia.
  The metrics included in this group are important in detecting potential security threats, as a sudden surge in new network connections from a particular region can indicate the presence of malicious activity.

Audit Logs: The audit logs provide information not present on other sources, in particular we consider:

- **Files**: This group of metrics provides information about the number of files that have been deleted, changed, or created during a period of time.
  The metrics in this group are the following, all provide values since last metric report:
  - **Files Write**: Number of files that have been written
  - **Files Read**: Number of files that have been read
  - **Files Created:** Number of files that have been created
  - **Files Deleted:** Number of files that have been deleted

  Many types of malware create, delete or modify a big number of files, for example:
  - **Viruses**: Viruses are a type of malware that infect executable files on a target's system. As the virus infects more files, it can create a large number of infected files on the system.
  - **Worms**: Worms are a type of malware that spread through computer networks, often by exploiting vulnerabilities in software. As worms spread to other systems, they may create copies of themselves or install additional malware on the victim's system. This can result in a large number of files being created, modified, or deleted on the victim's system. For example, the Conficker worm, was known to create a large number of files on infected systems. Once it infected a system, it would create a large number of randomly named files in various directories to make it more difficult for antivirus software to detect and remove the worm.

  - **Ransomware**: As just mentioned, ransomware attacks encrypt the victim's files. Therefore, a large number of files are modified.
  - **Wiper Malware [12]:** Wiper malware is a type of malware that is designed to destroy or overwrite the victim's data. The malware may delete or modify a large number of files.
- **Commands**: In this case, there is only one metric related to commands: **Number of Commands**. This metric represents the number of commands that have been executed since the last set of metrics was obtained. As machine learning models are being used to detect malware behavior, metrics have to be static. Therefore, dynamic metrics can't be used in this case, which could have been used to create metrics related to the specific commands being executed. Nevertheless, the number of commands being executed is an indicative metric that can be used to detect different types of malware.

It's important to note that a single metric may not necessarily provide enough information to detect malware on its own, as certain metrics may have legitimate reasons for fluctuating or deviating from *normal* behavior. However, by analyzing a combination of metrics at a specific time, we can gain a more comprehensive view of the system and identify patterns that may indicate the presence of malware.

4 Machine Learning Model

In this section, we will discuss the performance of the different Machine Learning algorithms tested in this work, namely Autoencoders [16] and Kernel Density Estimation [15].

Following this, we will discuss the significance of features in detecting specific attacks, such as ransomware and denial of service. This preliminary result has the goal of understanding which of the models have the necessary features to be chosen. Subsequently, we will compare all the machine learning algorithms that were tested and determine which one works best for this project. Finally, we will examine how false positives evolve after collecting data over a few days.

It is important to note that the test data used was the same for all the algorithms. The models were trained using 481 minutes of normal behavior data and each test was performed with data ranging from 14 minutes to 1 hour.

We leave out of this study algorithms such as One-class SVM and Local Outlier Factor models due to its poor performance.

4.1 Autoencoder model

In the autoencoder model, we also present the he combination of parameter values that works best. In this case the layers are defined as follows:

- **Input layer:** Defined with the shape of the training data.
- **Encoding layer:** defined with 49 neurons, ReLU activation function, and L1 regularization with a coefficient of 0.00.
- **Decoding layer:** Defined with 49 neurons and ReLU activation function.
- **Output layer:** Defined with the same number of features as the input layer and sigmoid activation function.

Activation functions are used to introduce non-linearity into machine learning models. ReLU [3] is one of the most commonly used activation functions in deep learning models. It is a simple function that returns the input if it is positive and zero otherwise. ReLU activates/ignores certain neurons in the model, which can help learn complex patterns in the data. On the other hand, the sigmoid [14] activation function maps any input value to a value between 0 and 1, which is useful when the output needs to be either 0 or 1. In this case, we need to classify the output as normal behavior or security threat. Therefore, sigmoid function can be used to map the output reconstruction error to a value between 0 and 1, where values closer to 1 indicate a higher likelihood of a security threat.

L1 regularization [17] is a technique used to prevent overfitting in machine learning models. What it does it to add a penalty term to the loss function that forces the model to set many of the weights to zero, which will remove some features from the model. In this case, we don't want to remove any of the features as all of them were determined to have importance. Therefore, the coefficient is set to 0.00 as it is important to not remove features.

The model is compiled with the Adam optimizer [2] and mean squared error loss function. The optimizer is the algorithm used to update the weights of the neural network during training. The Adam optimizer is a commonly used optimizer that adapts the learning rate of the network during training. The loss function is a measure of how well the model is able to reconstruct the input data. Mean squared error is a commonly used loss function for autoencoders, which calculates the average squared difference between the input and the output of the model.

The *epochs* are set to 50, which is the number of times to iterate over the entire training dataset. The *batch_size* is set to 32, which is the number of samples to be processed at once during each epoch. And the *validation_split* is 0.2, which is the proportion of the data to use for validation during training (in this case, 20%).

Finally, the threshold for anomaly detection is set at 85% of the reconstruction errors. This means that any data point with a reconstruction error greater than the $85th$ percentile of all reconstruction errors is considered an anomaly. We performed different tests and decided that a lower threshold started to get false positives.

The model performed very well in general. However, it had trouble detecting ransomware attacks, so we extracted the important metrics for the model (normalized to 100 for readability) as shown in Figure 2.

In this model, feature importance cannot be directly extracted. However, to understand which features are the most important we can examine the weights of the layers. This approach can be subjective, as the weights do not directly represent feature importance, and interpreting them can be challenging.

All features seem to have a similar importance, which may indicate that the features are not well differentiated and that they are not providing much information to the model. However, the performance of the model is good despite of this, which could mean that the dataset has a relatively simple structure and that the autoencoder is able to capture the important features with a similar level of importance. However, identifying and focusing on important features can improve the performance of the model. Not doing so, resulted in missclassifying the ransomware data in some cases that some of the metrics were similar to the normal behavior ones.

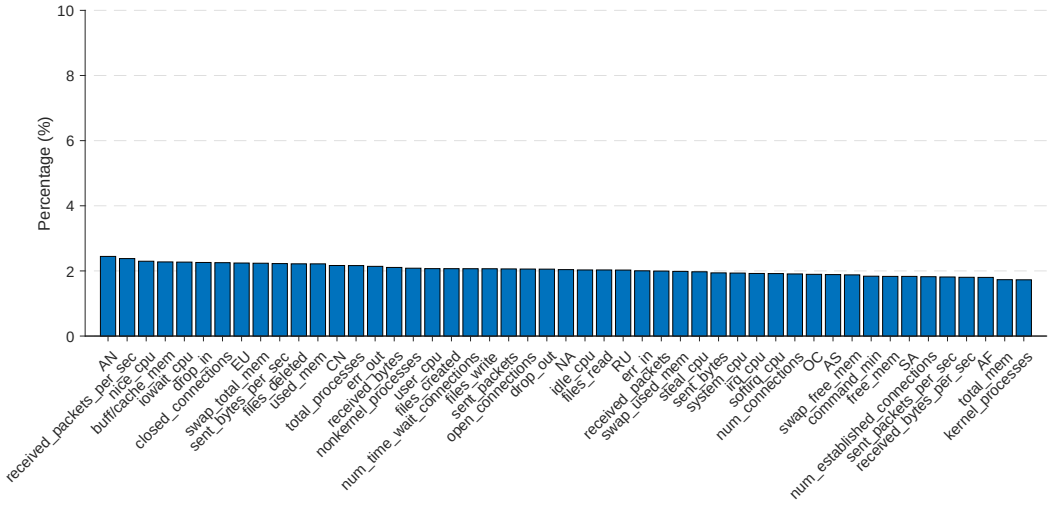


Fig. 2: Autoencoder feature importance

4.2 Kernel Density Estimation model

This model has many different parameters that can be set to different values. In this particular case, several combinations of parameters have been tried, The combination of parameter values that works best is the one that is presented in this section.

The *kernel* is defined as *Gaussian*. A kernel is a weighting function that is used to estimate the probability density function of the underlying data.

The *Bandwidth* is set as 'Scott', which determines the width of the kernel function, and it affects the smoothness of the density estimation. The Scott [13] bandwidth is a method that estimates the bandwidth based on the data, which can help to avoid overfitting or underfitting the data.

Finally, the *threshold* is set to the 7th percentile of the log-densities of the test data. This means that any data points with a log-density below this threshold will be classified as anomalous, and any data points with a log-density above this threshold will be classified as normal. The threshold is chosen based on the assumption that anomalous data points will have a lower density than normal data points. This parameter was decreased until the normal behavior data started to give false positives as it is important to assure the lowest number of false positives.

This model was able to accurately pass half of the tests. However, it had trouble detecting botnet and ransomware data. To better understand where the model is failing, we analyzed the features used in the model.

Kernel density estimation does not directly provide information about the importance of individual features in the data. Therefore, the sensitivity of the features is extracted. To do this, features of the test data are perturbed and the density estimate is observed. The sensitivity is calculated as the difference between the log-density of the original data and the log-density of the perturbed data. Features with higher sensitivities are likely to be more important in the anomaly detection task. The sensitivity of the features can be observed in Figure 3 presented as a percentage over 100.

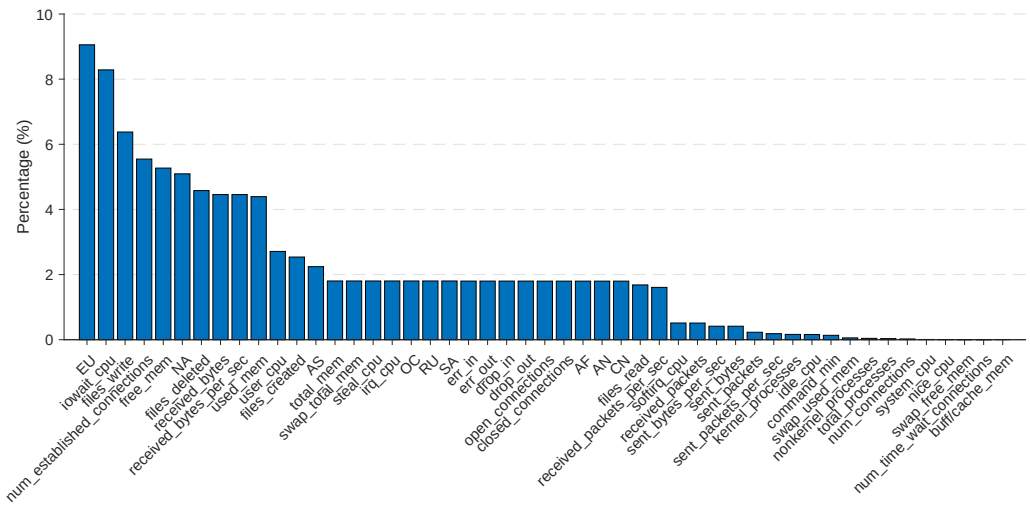


Fig. 3: Kernel Density Estimation feature sensitivities

Before applying the perturbation in this model, the features are first normalized. This is important because the range of values for each feature can be very different, so by normalizing, the same perturbation can be applied to each feature. If the perturbation was applied before normalization, it could result in inaccurate sensitivity calculations.

This model has identified certain features as being highly important, such as "EU", even though they should not have that level of importance. This can lead to incorrect classifications. The model's poor detection of ransomware attacks could be due to this incorrect feature importance. Some of the features that increase in ransomware attacks are the ones related to processes, which are not taken into account in this model. Due to all this we believe that this model is not suitable for our task.

5 Model Accuracy and validation

This section focuses on the data used to train and test the machine learning models. Firstly, we will examine the data that represents the user's normal behavior, which is the training data. Next, we will move on to the data that was collected to test the performance of the different models. Finally we compare the studied models and analyze how the false positives are reduced as the system learns.

5.1 Training data

In this section we take a look at the data that was generated to create the machine learning algorithms. Specifically, the data was collected by performing on a real system day-to-day activities such as:

- **Audiophile:** We profiled a person listening to music while idly performing other tasks.
- **Writer:** While working on this paper we created a writer profile.
- **Researcher:** To assess the behavior of a researcher, while processing the training data we also performed a profile of this workload.

- **Sys. Admin:** During the model debugging sessions using the Linux terminal we created this profile.
- **Journalist:** While searching for information related to this paper we performed long information search sessions.
- **Publisher:** As a backup and as work collaboration platform, publishing this paper to cloud facilities allowed us to create this profile.
- **Idle state:** This is not a profile per se, but in order to model the breaks and the system in idle state we created this profile as well..

5.2 Test data

The test data used to validate our work was collected on different days from the training data to ensure a more accurate evaluation of the algorithms' performance. This was done to assess real-world scenarios and obtain results that better reflect the models' capabilities.

Each type of data was collected during multiple days to improve the results' accuracy. By collecting data over a longer period, the study was able to capture a more comprehensive picture of the malware's impact on the user's behavior and computer usage patterns. Particularly we checked our system using the following malware:

- **Ransomware**: To obtain precise results, an actual malware was utilized, specifically a ransomware sample obtained from MalwareBazaar [1]. The sample was executed simultaneously while the user was engaged in typical activities.
- **DoS**: It was also decided to collect Denial of Service data as this malware detection system can be applied not only to computers used by individual users but also to servers, which are frequently targeted by DoS attacks. In this case we used GoldenEye.
- **Botnet**: In this scenario the user's machine has been compromised and is being used as part of a botnet involved in a Denial of Service (DoS) attack on a remote server. We also leveraged GoldenEye but as generator of the attack.

In all cases the user was working as usual using one of the profiles perviously generated.

In order to improve a little bit our model, we decided to also feed the system with similar load of malware but using legitimate applications. Particularly we used:

- **Backup**: Our model was trained both including and excluding the data collected while performing a backup in order to see the how this impacted the accuracy. The comparison can be found at section 5.4. It is important to add that the backup data used for testing was taken in different days that the one used for training.
- **Software compilation**: We used a kernel compilation as baseline to see how the system detected high CPU usage, as before we trained the model with and without this information.
- **Normal behavior**: To make the tests completed we also checked the accuracy of the system through normal work sessions to see how it was detected by the system.

5.3 Metric Relevancy

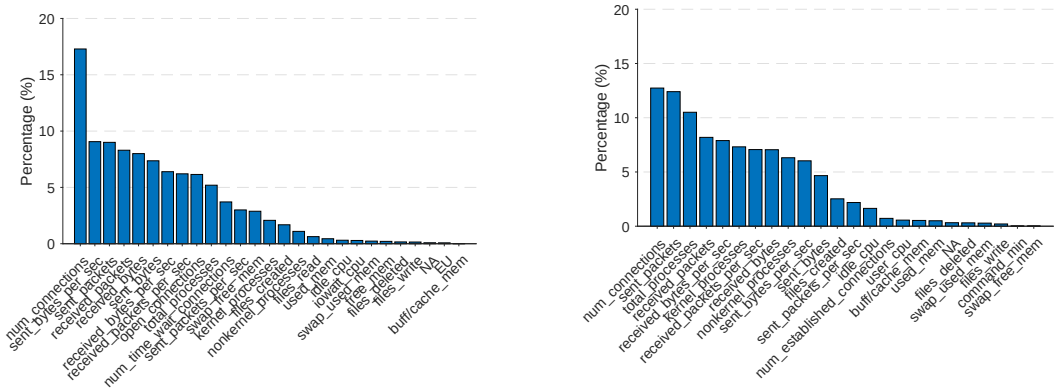
In this part, the Random Forest machine learning algorithm was used to compare normal user behavior data with malware data to observe which metrics are the most significant in detecting the malware. Apart from that, we detected that some machine learning models used had issues

identifying ransomware when high CPU usage was added to normal user behavior, we decided to analyze their difference in metrics.

It is worth noticing that the purpose of this comparison is to provide a general idea of the relevant features when it comes to differentiating normal behavior from security threats.

To make sure that only the key feature differences are extracted, the normal user behavior was the same in both malware and no malware data. Each feature's importance is presented as a percentage over 100 to make it easier to understand. This allows for a clearer visualization of the relative importance of each feature in the model.

DoS: the Random Forest model was trained with 2 labels, the normal behavior data and the malware data. The accuracy of the model was found to be 0.96296, which is very close to perfect accuracy. The metrics shown in Figure 4a are the ones that were the most relevant for the malware classification.



(a) Feature importance graph of a DoS attack

(b) Feature importance graph of a botnet

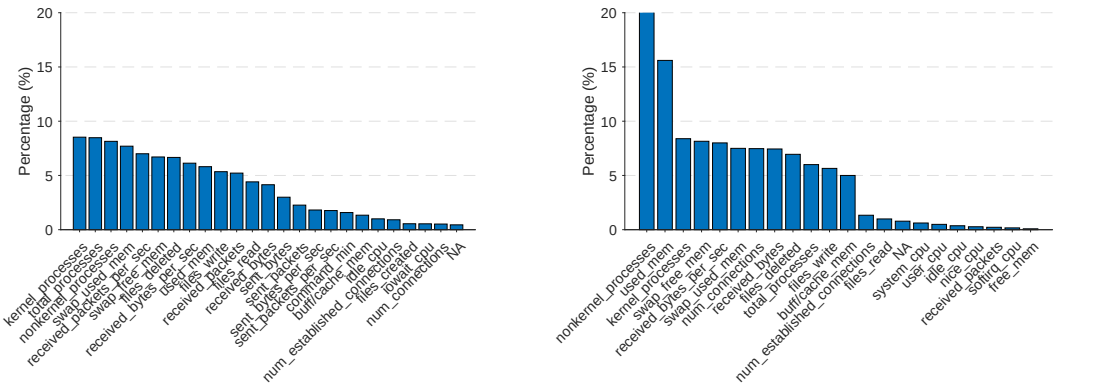
In this scenario, it is easy to differentiate between a Denial of Service (DoS) attack and normal user behavior. The most important metrics are related to the network, such as the number of connections and packets sent, which have higher values during a DoS attack.

Botnet: in this case the accuracy of the model to detect a botnet was 1, indicating perfect accuracy. The metrics shown in Figure 4b are the ones that were the most relevant for the malware classification.

In this situation, the analysis is similar to the DoS case in that network features are crucial. However, in this instance, the number of processes is also significant because creating a DoS attack requires a large number of new processes. During the botnet collection data, the number of processes increased by approximately 150, which is noteworthy.

Ransomware: we observed that the accuracy assessing ransomware was 0.96. The metrics shown in Figure 5a are the ones that were the most relevant for the malware classification.

As anticipated, the majority of important features are related to memory and files, for example used memory and files read/created. Additionally, some network features are also important, like



(a) Ransomware attack (b) Backup profile

the number of sent bytes, which shows a significant increase of about 200% during a ransomware attack.

As mentioned before, as some models trouble detecting ransomware attacks when introducing backup data, a comparison was conducted using backup data as the "no malware" group and ransomware data as the "malware" group. The key features that distinguish between the two groups can be visualized in Figure 5b.

In this case, the analysis of the features shows that those related to memory and files are the most important ones in distinguishing between ransomware and backup data. The amount of memory used by ransomware is slightly higher than that of backup data, but this can vary depending on the specific ransomware type. Additionally, the number of files written by ransomware is much higher than that of backup data since the ransomware encrypts the files, while in backup data, the files are not modified. On the other hand, the number of files read in backup data is much higher, reaching up to 1500 files per minute, compared to ransomware.

There are clearly various differences between the two datasets. However, the specific features that are considered important in the models will determine their ability to detect ransomware. For instance, if the important features are related to memory, then it may be difficult for the model to detect ransomware. Apart from that, when it comes to the models that use a threshold for detecting security threats, the value of the threshold can definitely change the result of the prediction. Therefore, an appropriate threshold that has both low false positives and high true positives should be found.

5.4 Models' Comparison

This section will focus on comparing all of the models, specifically their effectiveness in detecting malware and the number of false positives they generate. In addition, we will analyze how well the models perform when additional training data similar to specific types of malware attacks is incorporated. By evaluating these factors, we can determine which model is the most reliable and effective for detecting malware.

Initially, the precision level of each model is reviewed for various types of data. This can be seen in Table 1.

After reviewing the table, it can be observed that the Autoencoder model is the most accurate in detecting both normal behavior and malware activity, achieving good accuracy in all tests. It is

Table 1: Accuracy of different anomaly detection algorithms

	Autoencoder	**KDE**
Normal behavior	1	1
Botnet	1	0.9
DoS	1	1
Ransomware	0.71	0.42

worth noting that models with precision scores different than 1 in detecting normal behavior should not be considered since the goal is to have the least amount of false positives. This is particularly important because false positives can increase when making live predictions, and thus minimizing their occurrence is crucial. Due to space limitations in this work we don't present some algorithms which proved inaccurate, i.e., One-class SVM and Local Outlier Factor.

DoS attacks and botnet behavior is accurately detected by the majority of the models. However, ransomware attacks are not properly classifieds by KDE as their feature importance is not properly defined. In terms of malware detection, low accuracy in detecting malware activity can have serious consequences as it can delay the detection of the threat, which is crucial for containing and mitigating the attack. While the models may eventually detect the threat, the delay in detection can result in a more severe impact on the system. Therefore, apart from the Autoencoder model which has the highest accuracy for detecting both normal behavior and malware activity, the KDE model can also be considered as it has relatively better accuracy than the other discarded models.

To assess the reliability of the models, we will evaluate their performance when additional data related to backup and kernel compiling is included. Backup data, which uses a considerable amount of memory and accesses many files, is similar to ransomware data. Additionally, we will incorporate kernel compiling data, which utilizes significant system resources, to observe any changes in model precision. Table 2 demonstrates the varying performance of each model when this data is added.

Table 2: Accuracy of different anomaly detection algorithms - 2

	Autoencoder	**KDE**
Normal behavior	1	1
Botnet	1	1
DoS	1	1
Ransomware	0.78	0.35

By reviewing the results, it was observed that most models had reduced accuracy in detecting ransomware attacks. However, the Autoencoder model showed an increase in accuracy after adding data that is similar to ransomware attacks. This could be due to a reconfiguration of feature importance. Nonetheless, the accuracy of detecting ransomware attacks is still not perfect, and is something we will study on our future work.

5.5 False positive reduction

Finally, to see how much time would it take the models to have enough training data to have a considerably low rate of false positives, we'll see how these false positives progress in the span 8 days. For this study, 1 hour of normal behavior data was collected every day for the specified time period. Each day (excluding the first day), the new data was passed through the model to observe the false positive rate, then the data was added to the training set and the model was retrained. On

the first day of data collection, the data was solely utilized to create the model, without performing any testing. The Autoencoder was employed for this study, as it demonstrated superior performance in comparison to all other algorithms that were tested. The outcomes of this study are presented in Figure 6.

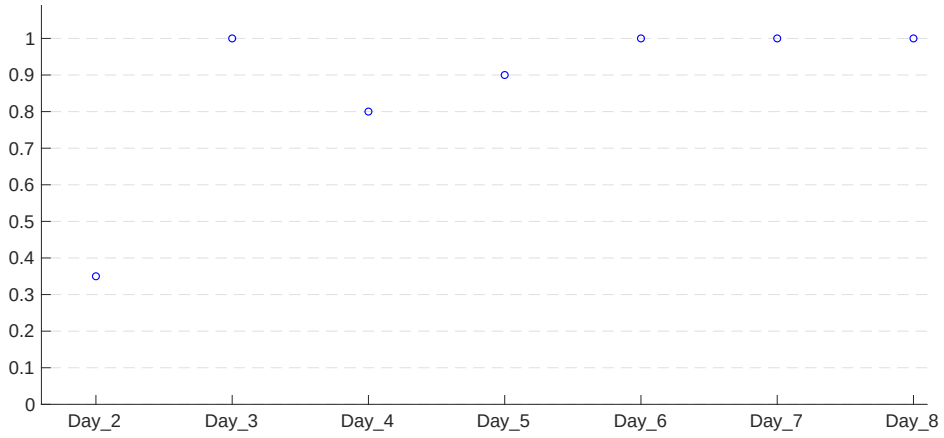


Fig. 6: False positive reduction

On the second day, which is the first day of predicting the data, the precision was quite low at 35%, but starting from the third day, it increased rapidly and stabilized at 100% on day 6. Therefore, it took a total of 6 hours to achieve perfect precision in this particular study.

However, it should be noted that these results are specific to this study, and achieving perfect precision may vary in other environments depending on the diversity of normal behavior data for each user. Nonetheless, this study demonstrates that it is possible to achieve perfect precision.

This statement suggests that in a work environment where employees follow a routine and exhibit consistent web browsing behavior, the model would require less time to reach a stable and accurate prediction. This is because the model would have a more predictable and consistent set of data to learn from, leading to a faster convergence toward accurate predictions.

This also suggests that implementing the detection system on a server with highly automated and repetitive tasks would be beneficial. Not only would it achieve high accuracy with normal behavior quickly, but as the actions are automated, it would effectively detect any type of security threat.

6 Conclusion

In this paper we have shown that using user behavior analysis is a feasible mechanism to detect malware, in our case of all the tested models we found that autoencoders seem to be the most accurate of the tested models, as it can quickly adapt to changing user behavior, while effectively ruling out actual malware samples. We have also shown that our models show high accuracy in predictions in a matter of seconds, given the frequent collection of metrics, any possible security threats can be detected rapidly.

A very big advantage of these light-weighted models is the fact that they can adapt to the specific behavior of the user, which avoids the need of having a database with malware samples and could detect zero-day threats just by noticing a deviation of the user's normal behavior.

As previously mentioned, we believe that it would be wise to deploy our solution on a running server since its behavior is often very repetitive and predictable, and the models tested in the study performed well in such a context. To be more precise, utilizing the Autoencoder model would be an effective way to identify virtually any potential security threat on a system.

We leave as part of our future work several aspects that may be of interest, first we do believe that there is still room for improvement by performing more real tests of the system to be able to better tune the detection under a set of changing user behaviors. We also think that trying other machine learning models such as Isolation Forests may prove a valuable addition to the solution.

References

1. abuse.ch: Sha256 edfe81babf50c2506853fd8375f1be0b7bebbefb2e5e9a33eff95ec23e867de1, https://bazaar.abuse.ch/sample/edfe81babf50c2506853fd8375f1be0b7bebbefb2e5e9a33eff95ec23e867de1/
2. Brownlee, J.: Gentle introduction to the adam optimization algorithm for deep learning, https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
3. Brownlee, J.: A gentle introduction to the rectified linear unit (relu), https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
4. Cyberwire, T.: signature-based detection, https://thecyberwire.com/glossary/signature-based-detection
5. Denning, D.: An intrusion-detection model, https://ieeexplore.ieee.org/abstract/document/1702202
6. Gavriluţ, D., Cimpoeşu, M., Anton, D., Ciortuz, L.: Malware detection using machine learning. In: 2009 International Multiconference on Computer Science and Information Technology. pp. 735–741 (2009). https://doi.org/10.1109/IMCSIT.2009.5352759
7. Griffiths, C.: The latest 2023 cyber crime statistics (updated february 2023), https://aag-it.com/the-latest-cyber-crime-statistics/#
8. Hindy, H., Atkinson, R., Tachtatzis, C., Colin, J.N., Bayne, E., Bellekens, X.: Utilising deep learning techniques for effective zero-day attack detection. In: Electronics. vol. 9, p. 1684 (2020). https://doi.org/10.3390/electronics9101684
9. Lane, T., Brodley, C.E.: An application of machine learning to anomaly detection, http://ftp.cerias.purdue.edu/pub/papers/terran-lane/brodley-lane-nissc97_paper.pdf
10. Miao, Y.: Understanding heuristic-based scanning vs. sandboxing, https://www.opswat.com/blog/understanding-heuristic-based-scanning-vs-sandboxing
11. Muhammad Ejaz Ahmed, Surya Nepal, H.K.: Medusa: Malware detection using statistical analysis of system's behavior, https://ieeexplore.ieee.org/abstract/document/8537842
12. packetlabs: What is wiper malware and how does it work?, https://www.packetlabs.net/posts/how-does-wiper-malware-work/
13. rdrr: bw.scott: Scott's rule for bandwidth selection for kernel density, https://rdrr.io/cran/spatstat.core/man/bw.scott.html
14. Saeed, M.: A gentle introduction to sigmoid function, https://machinelearningmastery.com/a-gentle-introduction-to-sigmoid-function/
15. sklearn: density, https://scikit-learn.org/stable/modules/density.html
16. tensorflow: Intro to autoencoders, https://www.tensorflow.org/tutorials/generative/autoencoder
17. Tyagi, N.: L2 and l1 regularization in machine learning, https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning