Towards Runtime Classification of Ransomware

by

Madhumitha Balaji

**A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Computer and Information Science)
in the University of Michigan-Dearborn
2021**

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The availability of computer systems is constantly being challenged by cybercriminals who seek to disrupt access to this indispensable technology and the data they contain as a means for making profit. This trend has given rise to a new form of malware that is known as ransomware, an invasive type of malware that is designed to appropriate compute resources in return for a ransom. According to the U.S. Department of Homeland Security, ransomware represents the fastest growing malware threat to individuals and organizations. With the cost of ransomware attacks projected to exceed $20 billion in the year 2021, it is imperative to explore solutions that can defend against such malware.

In this study, we evaluate the effectiveness of machine learning algorithms and their suitability for detecting ransomware on x86 platforms. We show that dynamically extracting instruction op-codes from execution traces can be harnessed for training machine learning models that can used to perform runtime detection of ransomware. We evaluate different machine learning models and demonstrate that tracking a limited number of instruction opcodes commonly used crypto-graphic are sufficient for reliably detecting ransomware with high accuracy. We show that our method can achieve high detection rates above 99% while evaluating our solution against real ransomware available in a state-of-the-art dataset from VirusTotal.

CHAPTER 1

INTRODUCTION

The digital world around us is growing exponentially, resulting in tectonic shifts to the way we approach everyday computing. The ubiquity of computer systems across all industries has transformed this technology into a vital fabric that governs all facets of our society, including healthcare, manufacturing, and retail. The mobility of today's devices, fused with advances in cloud computing, continue to drive new synergies in the way essential services are delivered to consumers, producing vast amounts of data that both individuals and organizations depend on for making their daily decisions.

Unfortunately, the availability of these systems is constantly being challenged by cybercriminals who seek to disrupt access to this indispensable technology and the data they contain as a means for making profit. This trend has given rise to a new form of malware that is known as ransomware, an invasive type of malware that encrypts the victim's data or locks their system. The malware then extorts the user to pay a ransom in return for decrypting their data or restoring access to their system [1]. According to the U.S. Department of Homeland Security, ransomware represents the fastest growing malware threat to individuals and organizations [2]. More than 112,000 unique mobile ransomware samples were found in 2018 [3], posing a threat to mobile users by denying them access to their personal data, in addition to locking them out of their devices. Moreover, a wide range of business segments incurred significant damages as a result of ransomware, costing the pharmaceutical, shipping services, and chip manufacturing industries over $850 million, $400 million, and $250 million, respectively [4; 5]. Similarly, local governments have fallen victim to such attacks, including the City of Baltimore that has spent over $18 million to date in order to recover from ransomware that crippled various municipal operations [6]. Although the cost of ransomware attacks in 2018 has been estimated to exceed $8 billion, future damages are projected to reach $20 billion by the year 2021 [7]. This trend makes it imperative to explore solutions that can both detect and recover from such attacks.

In this study, we evaluate the effectiveness of machine learning algorithms and their suitability for detecting ransomware on x86 platforms. We show that extracting instruction opcodes from execution traces can be harnessed for training machine learning models that can used to perform runtime detection of ransomware. We examine the performance of different machine learning models and demonstrate that opcodes from the x86 instruction set are sufficient for reliably detecting ransomware deployed on Windows platforms. We show that this method can achieve high detection rates above 99%. We evaluate the robustness of our work against real ransomware from a comprehensive dataset from VirusTotal [8].

Overall, this work makes the following contributions:

- Characterizes the behavior of an emerging type of malware at the instruction level.
- Makes the observation that using the instruction opcodes of executed programs serve as reliable features for detecting ransomware at runtime.
- Evaluates multiple machine learning algorithms and their suitability to detecting ransomware with high accuracy while maintaining a low false positive rate.
- Demonstrates that a limited number of commonly used cryptographic instructions could be harnessed as features for distinguishing between ransomware and benign applications, a step towards autonomous ransomware detection through light-weight hardware-malware-detectors (HMD).

The rest of this thesis is organized as follows: Section 2 provides background information.

Section 3 details related work. Section 4 discusses the methodology for evaluating the approach. Section 5 presents the results of our evaluation; and Section 6 concludes.

CHAPTER 2

BACKGROUND

2.1 Ransomware

Ransomware is considered to be an invasive form of malware that is designed to deny its victims access to valuable digital resources, generally in the form of data present on computer systems. The malware is designed to demand a ransom from its victims in return for restoring access to any affected resources once a system becomes infected. Payment for such attacks are typically carried out through anonymous cryptocurrencies, such as Monero [9] as a way of ensuring anonymity of the cybercriminals' identities behind such malicious campaigns. Ransomware can be broadly classified into one of two categories: cryptographic ransomware and locker ransomware. Although the primary aim of both these types are to garner monetary benefits, each of them differ in their Modus Operandi.

2.1.1 Cryptographic Ransomware

Cryptographic ransomware is a type of malware that is designed to encrypt valuable files present on a victim's system while revoking access until the specified ransom is paid. Such ransomware typically harnesses well established encryption algorithms that are known to be cryptographically strong. This includes the Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA). This allows cybercriminals to re-purpose such algorithms for efficiently preventing victims from accessing their files, obviating the need to explicitly transfer hijacked resources out of the infected system. One of the most notable malicious programs that belongs to this ransomware family is WannaCry [10; 11]. The ransomware became sensational within a short period of time due to the magnitude of its attack. It spread quickly across 150 countries by exploiting the Server Message Block (SMB) vulnerability within the Windows operating system. In addition to WannCry, other types of ransomware have been discovered in the wild including Locky, Odin, TeslaCrypt, Thanos, and Petya.

2.1.2 Locker Ransomware

Unlike cryptographic ransomware, this family of ransomware focuses on locking victims out of their user accounts. Such ransomware operates by overtaking the operating system's module that is responsible for logging users into their systems and preventing further logins until the required ransom is paid. Unlike cryptographic ransomware that require attackers to provide a decryptor that can be used to decrypt encrypted files, locker-based ransomware reverts its effects by simply unlocking the system. Although locker ransomware can be disruptive, the data present on the system remains intact and can be easily recovered by extracting the disk from the affected system and installing it into a clean system that is known good. As a result, our study primarily focuses on evaluating cryptographic ransomware which are known to have far more damaging consequences relative to locker-based ransomware.

2.2 Supervised Learning

In this section, we give an overview of the different machine learning algorithms we employ in the evaluation section. In our study, we primarily focus on using a branch of machine learning algorithms that rely on a supervised learning approach, an approach that associates a set of input data points X(1),...,X(m) to a set of definitive outcomes Y(1),...,Y(m).

2.2.1 Logistic Regression (LR)

Logistic regression, a variant of classification algorithms that is used to assign observations to a set of classes that are discrete. Although known for its simplicity, this model has been employed in a wide range of classification problems including email spam, online fraudulent transactions, and malignant tumor discovery. The algorithm transforms its input to an output using the logistic sigmoid function and returns a probability value, making it a predictive type of algorithm. A set of probability values can be mapped to two or more classes. Logistic regression can be referred to as a linear regression model that uses a complex cost function. However, instead of a linear function, this cost function is defined as the 'Sigmoid function'. The hypothesis of the algorithm tends to limit the cost function between 0 and 1. The linear function passing through the sigmoid results in the equation described in equation (2.1) where the sigmoid function is equal to $[\sigma(z) = 1 \div (1 + e^{-z})]$.

$$w^T x = w_0 + w_1 x_1 + \ldots + w_n x_n \text{ and } P(y = l|x) = \sigma(wTx) \tag{2.1}$$

2.2.2 SVM-Linear (SVM-L)

The Support Vector Machine (SVM) is a linear model that has the ability to classify problems that are both linear and non-linear in nature. The basic idea of the model is that it creates a hyperplane to separate the input dataset into classes. It finds the points that are closest to the hyperplane from the different classes and attempts to maximize the distance between these classes. In other words, the model tries to decide a boundary such that the separation between the classes is as wide as possible. The linear SVM classifier model can be expressed by equation (2.2) where $[x \in R^n]$ is the input vector, y is the class label, w is the weight vector, C is a regularization parameter, and b is the bias.

$$\min_{w,\xi}\left(||w||^2 + C \sum_{i=1}^{n} \xi i \right) \text{ s.t } \forall\ y_i(w^T x_i + b) > 1 - \xi i \text{ and } \xi i > 0 \tag{2.2}$$

2.2.3 SVM-Polynomial (SVM-P)

SVM-Polynomial is considered to be the polynomial version of SVM-Linear. It uses a polynomial kernel function that is used to represent the similarity of vectors in a feature space over polynomials of the original variables. Therefore, giving it the ability to approximate non-linear functions.

A strength of the SVM-polynomial model is that it looks at a combination of features to determine similarities instead of treating each feature independently. Such combinations are known as interaction features. For degree-d polynomials, the polynomial kernel is defined as shown in equation (2.3) where x and y are vectors in the input space and c is a free parameter to trade off the influence of higher-order and lower-order terms of the polynomial.

$$K(x, y) = (x^{\mathrm{T}}y + c)^d \tag{2.3}$$

2.2.4 SVM-RBF (SVM-R)

The Gaussian Radial Basis function (RBF) is another popular kernel method that is used in SVM models. At a high level, the RBF kernel is a function whose value depends on the distance from a given origin that is computed using equation (2.4).

$$||X1 - X2|| = \text{Euclidean distance between X1 and X2} \tag{2.4}$$

Similarities between $X1$ and $X2$ is determined based on the distance in the original space. In addition, the model uses a γ parameter to control overfitting. Whenever γ increases the model tends to overfit and whenever it is decreased, the model tends to underfits.

2.2.5 K-Nearest Neighbors (KNN)

The K-nearest neighbors (KNN) is a classification method that is non-parametric in nature. In this algorithm, the input comprises of the K closest examples in the training data set. This algorithm also relies on distance computations between the points as a way of clustering vectors that share similar characteristics. A peculiarity of the algorithm is its sensitivity to the local structure of the dataset. Unlike other algorithms, KNN doesn't require training. Instead, it learns from the training dataset at the time of providing providing predictions based on a given input. The algorithm has the advantage of being relatively simple to implement since it primarily requires the use of a distance function and the value of K to be used. A downside to this algorithm, however, is that the cost of calculating the distance between the different points can increase drastically as a function of the dataset. As a result, the KNN algorithm can introduce a non-trivial amount of runtime overhead when large datasets are employed.

2.2.6 Random Forest (RF)

Random forest is another learning algorithm that combines a multitude of decision trees during the training phase. Classification in this algorithm is achieved through ensemble learning where the final prediction is based on a majority vote of the different predictions produced bythe individual decision trees within the ransom forest. In addition to its speed, an advantage of this algorithm lies in its robustness against overfitting issues.

2.2.7 Naïve Bayes (NB)

Naïve Bayes is a simple probabilistic model that is based on based on Bayes' theorem. As such, this model is primarily concerned with learning the underlying distribution of the data. Despite the naive design and oversimplified assumptions, this type of classifier has been shown to be effective in a range of complex real-world problems. An advantage of Naïve Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

2.3 Quality Metrics

We evaluate our work against commonly referenced quality metrics. These include metrics such as accuracy, precision, F-score, and the receiver operating characteristics (ROC) curve. As a result, we provide a brief overview of the aforementioned metrics and how they are computed.

The aforementioned metrics rely on quantities such as false positives (FP), and false negatives (FN), true positives (TP), and true negatives (TN). In our models, a false positive corresponds to the case where we misclassify a benign app as ransomware. A false negative is more serious and corresponds to the case where we classify ransomware as a benign application. On the other hand, a true positive refers to the case where we correctly predict ransomware whereas a true negative correlates to the case where we correctly predict benign applications.

2.3.1 Accuracy

This is a metric that provides a high level indication of a model's ability to classify data. It is represented as the ratio of correct predictions relative to all tested data points. This is summarized by equation (2.5).

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.5}$$

Here FP and FN represent the mispredictions within our test dataset and refer to false positives and false negatives, respectively. On the other hand, TP and TN represent the correct predictions and refer to true positives and true negatives respectively.

2.3.2 Precision

This metric reflects the quality of the positive predictions. In other words, it characterizes the ability to detect malicious apps. This is measured through equation (2.6).

$$Precision = \frac{TP}{TP + FP} \tag{2.6}$$

2.3.3 F-score

This represents the harmonic mean of the precision and the true positive rate (TPR) metrics which is summarized by equations (2.7) and (2.8). Unlike accuracy, the F-score provides a more balanced measure of the precision and true positive rate. This is especially useful when assessing class distributions that are not balanced. In addition, define TPR's counterpart, the false positive rate (FPR) as described in equation (2.9).

$$F\text{-}score = 2 \times \frac{TPR \times Precision}{TPR + Precision}. \tag{2.7}$$

$$TPR = \frac{TP}{TP + FN} \tag{2.8}$$

$$FPR = \frac{FP}{TN + FP} \tag{2.9}$$

2.3.4 Receiver Operating Characteristics Curve (ROC)

The ROC represents a measure of the true positive rate (TPR) as a function of the false positive rate (FPR). It illustrates how different decision thresholds affect the TPR and FPR. Therefore, the closer the ROC curve is to the left, the higher the overall accuracy of the model. Another way to capture the ROC information is by computing the area under the ROC curve (AUC) which is what we employ in our study. The closer the AUC value is to 1, the higher the overall accuracy.

CHAPTER 3

RELATED WORK

A significant body of research has explored various forms of ransomware detection solutions. In this section, we discuss the prior work on this topic that spans: ransomware detection, ransomware recovery, as well as solutions for other types of malware.

3.1 Ransomware Detection

Various solutions have been proposed for detecting ransomware [12; 13; 14; 15; 16; 17; 18]. Techniques such as [15] explore the use of a temporary artificial user environment where ransomware can run and interact with the system. In addition, the solution monitors the amount of entropy present in I/O transactions that are destined for the disk drive as a way of detecting encryption activity. Other work by Scaife et al. [12] use a behavioral approach for detecting ransomware that is similar to [15]. Similar to [15], the proposed solution considers anomalies in the sequence of I/O transactions issued to the file system along with entropy of each transaction. Work by Moussaileb et al. [18] examined file system traversal patterns as an early mitigation mechanism against ransomware. Their technique employs decoy files that are located at the root of the file system. The decoy files are treated as canaries for detecting ransomware and preventing the rest of the file system from being encrypted. On the other hand, work by Pascariu et al. [19] take a fundamentally different approach. They propose the use a low-cost honeypot that hosts decoy files through a network file system that has auditing capability. This honeypot is used to serve as a first line defense for users on the same network on the presence of ransomware in the event that the shared decoy files are encrypted.

Additional work considered static analysis techniques in detecting ransomware. Work by Naik et al. [20] applies fuzzy hashing and clustering techniques on a ransomware corpus to demonstrate the effectiveness of their approach. Other work by Vidyarthi et al. [21] considered a hybrid design that combines both static and dynamic analysis techniques to further improve the accuracy of ransomware detection. Their work examines parameters that include the entropy of the file, the presence of packers in the executable, embedded strings that contain suspicious keywords such as ”crypt”, and code that makes changes to the Windows registry.

In addition to desktop-based solutions, other work focused on mitigating the effects of ransomware on mobile systems [22; 1; 23]. Work by Chen et al. [22] proposed a detection mechanism tailored for mobile systems known as RansomProber. The solution operates by correlating finger movements initiated by the user to actively running apps. The solution then classifies any running apps that don’t correlate to any interactions with the user as malicious. Other work by Lachtar et al. [1] focused on statically analyzing executables produced by the Android Runtime system (ART) for malicious content. The solution leverages light-weight machine learning algorithms to detect ransomware code and prevents them from being launched. [23] considers a similar approach for the Android platform while efficiently harnessing convolutional neural networks to classify mobile applications. Other work by Ko et al. [24] explored a design that considers intercepting system calls into the kernel as a way of fingerprinting ransomware activity.

Unlike the majority of the aforementioned work that focuses on monitoring I/O transactions destined to the file system, our study examines ransomware at a more fundamental level. We investigate ransomware at the instruction level as it executes through the processor. As a result, our work has the potential to enable autonomous hardware malware detectors that can detect ransomware much earlier, as it passes through the microprocessor during the execution stage. This approach has the potential to preserve the file system by flagging malicious sequences of instructions well before data is altered on the file system.

3.2 Ransomware Recovery

Paying a ransom for lost damaged data does not always guarantee proper restoration of one's data files. This prompted several researchers to investigate solutions that can seamlessly undo the effects of ransomware through different mechanisms [25; 26; 27; 28; 29]. For instance, Continella et al. [28] proposed ShieldFS. The proposed solution examined the use of an add-on driver that safeguards the file system from cryptographic ransomware. The add-on driver considers various file system parameters that include that entropy of write operations, the frequency of read and write operations, folder-listing operations, and file type usage statistics in order to stop ransomware activity destined to the system. On the other hand, [29] explored the concept of creating safe zones within the file system that can later serve as a backup. With this design, users can mitigate the effects of ransomware by moving important documents to the aforementioned zone. Once files have been moved to the safe zone, file access is only granted upon authorization of the user. Running applications are not granted access to the safe zone in order to prevent malicious apps from encrypting the data. In the event that a system is infected with ransomware, the user can restore any damaged files from the aforementioned safe zone. Unfortunately, such backup solutions introduce a non-trivial amount of overhead during runtime which can impact the usability of the system. Such solutions also require a substantial amount of storage space in order to create backups that can be later used as restoration points.

Work by Kolodenker et al. [30] developed a prototype that employs a key escrow mechanism for saving any encryption keys that are generated on the system. The solution then retrieves the saved encryption keys from the key escrow and uses them to decrypt any affected files in the event that the system is infected by ransomware. The solution in [30] achieves this by monitoring operating system services that call the cryptographic libraries used by the OS. Unfortunately, this solution can be evaded by ransomware that embed cryptographic functions directly within their malware. Other work by Huang et al. proposed a mechanism known as FlashGuard [31]. FlashGuard uses a device-level recovery mechanism that obviates the need for using backups at the filesystem-level. This solution operates by tracking updates made to pages within the memory subsystem and storing the original copies to a solid-state-drive (SSD). Original pages are flushed to the SSD through modifications made to the garbage collection subsystem. Unfortunately, this solution is limited to systems that are equipped with SSDs. In addition, it requires duplicating data on the drive which makes it inefficient given that SSDs typically have limited storage capacities.

3.3 Other Malware Solutions

Previous work has also investigated defenses against general malware that span mobile and computer systems [32; 33; 34; 35; 36; 37; 38; 39; 40]. Work by Jiang et al. [32] characterized various forms of Android malware providing information such malware in terms of method installation, activation, malicious payload, permissions requests. Other work [33; 34; 35] looked at the malware behavior at run time to identify and detect the malicious behavior. DroidDolphin [34] used API calls to identify malware. However, the speed provided by the emulation environment was inefficient. MalDozer [35] embodied a solution that detects malware with high accuracy using raw sequence of API and deep learning techniques. However, this introduces overhead to the overall system, in addition to being vulnerable to different forms of obfuscation attacks.

As mitigation to the high overhead of software-based malware detection the research community explored various hardware-based solutions for defending against malware threats [36; 37; 38; 39; 40; 41; 42]. Such solutions generally rely on the usage of standard high-performance counters (HPCs) that can be used to identify malware by combining readings from the aforementioned performance counters and machine learning algorithms. Work by Khasawneh et al. [43] is an example of such solutions. They also explored making their hardware-based malware detector more resilient against obfuscation techniques that rely on reverse engineering malware detectors. The solution accomplishes this by stochastically switching between different detectors that are embedded within hardware as a way of making it difficult for attackers to reverse engineer the solution.

Other work, such as [44] combined static analysis techniques with machine learning algorithms. The solution considered both the permissions requested by applications, as well as the system calls they use in order to distinguish between malicious and benign applications. Leeds et al. in [45] explored the effectiveness of using convolutional neural networks (CNN) for malware detection. However, the results of this approach were sub-optimal, achieving a detection rate of 90% or less. [46] utilized a feature vector generation approach with multi-modal deep learning. However, similar to the aforementioned solutions, the model used in this solution can be evaded by obfuscation attacks. New ways are being explored to use CNN for malware detection by converting malware to image format. [47] converted permission files into images and fed them to different CNN models. This method resulted higher accuracy than previous ones. Providing promising results for malware dictation using image classification.

3.4 Threat Model

In this section, we discuss the threat model of our system and assumptions made about the attackers. We make the assumption that the ransomware programs are installed on systems through established mechanisms that are supported by the operating system. We also assume that the target system on which the solution is being executed is clean, up to date, and no authentication mechanisms have been compromised. We also assume that an attacker could use social engineering techniques such as phishing as a way of luring victims into executing their malicious programs.

CHAPTER 4

METHODOLOGY

4.1 Experimental Framework

Experiments for this study were conducted on an Ubuntu 16.04 host machine that was equipped with Intel E5-2680 processors and two Nvidia P2000 GPUs (Pascal architecture). Since the majority of available ransomware samples are designed for platforms equipped with the Windows operating system, we created a virtual machine (VM) that ran a Windows 7 image on top of the aforementioned host. The virtual machine (VM) was configured to model a standard PC enabled with sufficient compute resources that could run modern software compiled using the x86 instruction set. To this end, the VM consisted of 8 CPU cores, 10GB of memory, and 30GB of disk storage. To ensure the consistency of the collected results across different ransomware families, a new VM was cloned from a pristine copy after the execution of every ransomware sample. This approach allowed us to eliminate any compounding effects associated with ransomware from previous runs and ensure a clean state of the operating system prior to the initiation of any new runs.

Furthermore, we developed a framework written in Python that automatically executed and dynamically collected execution samples from a list of applications. In addition to executing programs, the framework was responsible for interacting with running programs by issuing key presses and mouse clicks in order to simulate a real user environment. To track the instructions executed by the different applications, we used Intel's Software Development Emulator (SDE)

Ransomware	Lock	Encryption	RAT	Samples
Locky		✓		10
Matsnu.com	✓	✓		10
MalPack		✓		10
Nanocore			✓	10
Odin		✓		10
WannaCry		✓		10
Kryptik		✓		30
Trojan Agent E.Generic			✓	30
GandCrab		✓		180

Table 4.1: Summary of ransomware families used in evaluation.

[48]. On the other hand, our framework handled the necessary parsing of the trace logs generated by the SDE tool and summarizing them into a Comma Separated Vector format (CSV). To ensure the completeness of our dataset, our framework took multiple samples of each executing application/ransomware. This allowed us to capture different execution phases of the running program. In addition to using the framework for recording execution traces, we used libraries from scikit-learn for implementing the machine learning models we evaluated in this study. We built models for each of the following machine learning algorithms previously described in section 2.2: Logistic Regression (LR), Support Vector Machine Linear (SVM-L), Support Vector Machine Polynomial (SVM-P), SVM Radial Basis Function (SVM-RBF), K-Nearest Neighbor (KNN), Random Forest (RF), and Naive Bayes (NB). We characterized the performance of each model against different feature sets in order to understand the relevance of each instruction in detecting ransomware.

4.2 Dataset

Our dataset consisted of 600 samples that were collected from popular user applications and ransomware running on a Windows 7 platform. We used a balanced dataset made up of 300 samples from benign applications and the remaining from real ransomware. We dedicated 70% of our dataset for training, 15% for validation, and the remaining 15% for testing. Our benign samples within the dataset were based on execution traces collected from popular user applications such as Microsoft Word, Power Point, Excel, Outlook, Visual Studio, Snipping Tool, WhatsApp, Zoom, VirtualBox, and Google Chrome. This allowed us to profile the nature of instructions executed on a clean system that is free of ransomware that we used for training our detection algorithms.

We used real ransomware from the Virus Total [8] repository for our malicious portion of the dataset. Although our study focuses on cryptographic ransomware, we also considered ransomware with locking capability, such as Matsnu, as well as ransomware in the form of Remote Access Trojans (RAT), such as Nanocore that moves files off the victim's machine to a a cybercriminal-owned server. A summary of the different ransomware families along with their capabilities are listed in Table 4.1.

CHAPTER 5

EVALUATION

Our study investigates the effectiveness of combining low level instructions with machine learning algorithms for detecting cryptographic malware. We analyze the performance of each machine learning model against different feature set configurations that include the use of the frequency of all recorded instruction opcodes, the frequency of a subset of recorded instructions determined via the principle component analysis (PCA) algorithm, and the frequency of a reduced set of instructions that are commonly employed in standard cryptographic algorithms to perform data encryption. In all cases, we consider instruction opcodes that belong to the x86 instruction set architecture (ISA). We first begin by discussing the performance of different machine learning when using the full feature set.

5.1 Instructions as Features for Ransomware Detection

After characterizing several ransomware and commonly used applications, we observe that dynamically tracking the instructions effective on the system is effective in detecting ransomware. Figure 5.1 summarizes the detection rate of our design across different machine learning models when using the full list of observed instructions as a feature set. On average, we observe a detection rate of 70.8% and a false positive rate of 2.8% while using our test dataset. More specifically, we find that with the exception of SVM-P and SVM-R, most models are able to detect ransomware at rate of 93.6% and above. We observe similar results for the F1-score metric which averaged 70.7% across the different models. Overall, our results show that when

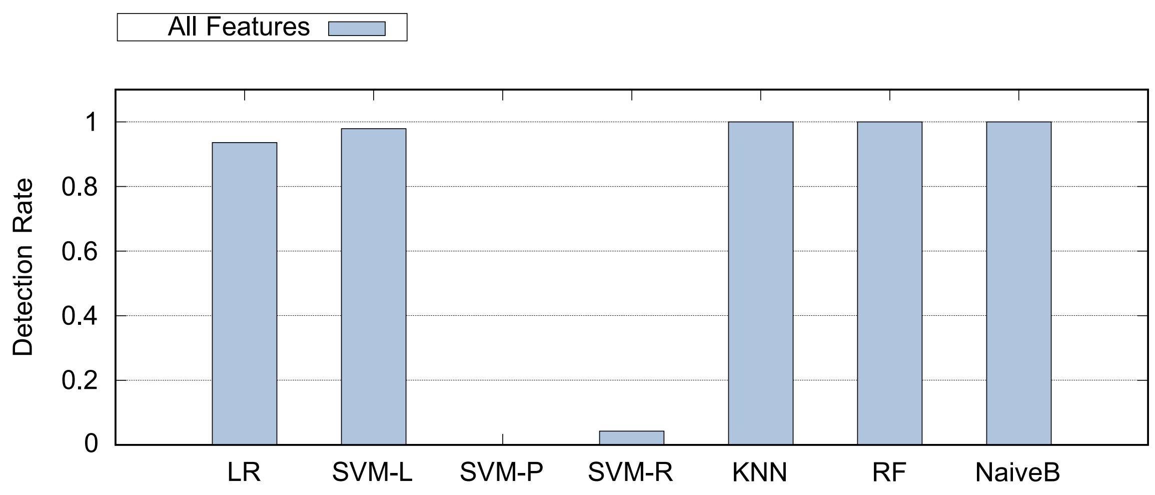

Figure 5.1: Comparison of ransomware detection rates across multiple families when combining different machine learning models with the full set of available features.

Model	**Accuracy**	**TPR**	**FPR**	**Precision**	**F-score**	**AUC**
Logistic Regression	0.871	0.936	0.196	0.830	0.880	0.870
SVM-Linear	0.989	0.979	0	1	0.989	0.989
SVM-Polynomial	0.495	0	0	0	0	0.500
SVM-RBF	0.516	0.043	0	1	0.082	0.521
KNN	1	1	0	1	1	1
Random Forest	1	1	0	1	1	1
Naive Bayes	1	1	0	1	1	1

Table 5.1: Summary of quality metrics under different models when using the full set of features.

using the full feature set which consisted of 158 features, KNN, Random Forest, and Naive Bayes algorithms are able to detect all ransomware samples within our test set. Furthermore, we observe that all of the algorithms with the exception of Logistic Regression were able to correctly classify all benign samples. This underscores the effectiveness of using instruction opcodes for detecting ransomware at runtime. However, we note that the Logistic Regression algorithm performed poorly when tested with benign applications. Unlike the other algorithms, it misclassified benign applications at a rate of 19.6%. A summary of all machine learning models we considered in this study against the quality metrics described in section 2.3 is shown in Table 5.1.

In order to better understand the performance of each model, we examine the number of false positives and false negatives each algorithm produces when using our test set. False positives occur when a model missclassifies a benign sample as ransomware. Although the goal is to keep the metric as low as possible, a false positive doesn't result in the user's system being infected with ransomware. It translates falsely notifying the user of the presence of ransomware, which would likely be perceived a nuisance to the end user. On the other hand, a false negative has more serious consequences since it implies that ransomware was incorrectly misclassified as a benign application. Therefore, maintaining the false negative metric as low as possible is of utmost importance when evaluating a malware detection solution. Figure 5.2 shows the breakdown of the false positives and false negatives observed when using different machine learning algorithms with the full feature set. On average, our solution produces 1.3 false positives with logistic regression exhibiting the highest count of 9 false positives. The remaining algorithms had no false positives. On the other hand, the average number of false negatives was relatively high. On average, 13.7 false negatives were produced. This implies that, on average, almost 14 ransomware went undetected. We find that the SVM-P algorithm was the least effective in detecting ransomware with 47 undetected samples, followed by SVM-RBF with 45 undetected samples. However, unlike the aforementioned models, the KNN, Random Forest, and Naive Bayes algorithms achieved the best performance detecting all the ransomware samples present within our test set. Given these KNN, Random Forest, and Naive Bayes did not yield any false positives, we consider these to be the most suitable models for detection in the event that the full feature set is used.

5.2 Dimensionality Reduction

Although our results show that tracking the execution frequency of all instruction opcodes as features is effective, this approach can suffer from scalability issues. For instance, prior work has shown that when using other instruction sets, such as the ARM ISA, the size of the feature set go beyond 5,000 features [1]. Processing this many features at runtime can denigrate the overall performance of the system. As such, it is important to minimize the number of features

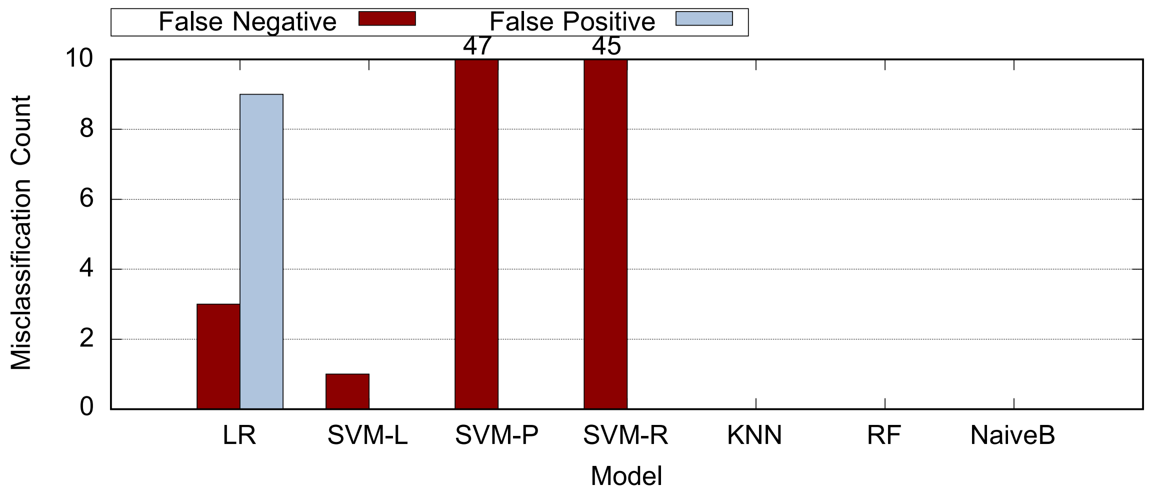

Figure 5.2: False positive and negative misclassifications when combining different machine learning models with the full set of available features.

to be used in a given model. To this end, we explore dimensionality reduction techniques on our original feature set. We use the principle component analysis (PCA) algorithm for reducing the dimensions of our original feature set.

In our study, we were able to reduce the number of features from 158 down to 18 components. This spanned instruction opcodes that are used for performing loads and stores, such as MOV and MOVX since cryptographic ransomware needs to perform several memory accesses as part of reading the data in plaintext form and saving the corresponding ciphertext. We also observe performance related instructions, such as PREFETCHW that are designed to speedup data access by prefetching expected data into the processor's caches. Such instructions are likely used in order to achieve fast encryption of user files and outpace any human response or intervention. We also observe general instructions that are also commonly found in cryptographic functions, such as XOR.

Figure 5.3 summarizes the detection rates of the different machine learning models when using the reduced feature set that we obtain from the PCA algorithm. Overall, we observe similar results relative to the full feature set approach with slight improvements for certain algorithms. On average, we observe a detection rate of 78.4% and a false positive rate of 6.5% while using

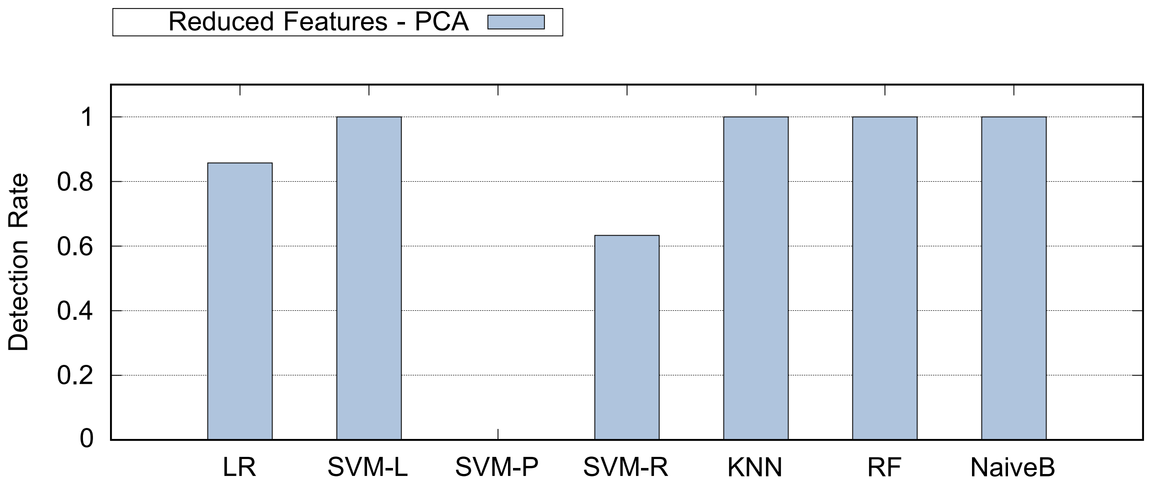

Figure 5.3: Comparison of detection rates of ransomware from 9 families when combining different machine learning models with a reduced feature set through principle component analysis (PCA).

our test dataset. More specifically, we find that the SVM-P, SVM-R, as well as logistic regression performs poorly with the reduced feature set. The other models, on the other hand, achieve detection rates of 97.8% and above. In general, we find that the detection rate improved by 7.6% when using the reduced feature set. However, the false positive rate increased by 3.7%. Overall, our results show that tracking a reduced set of instructions which consisted of 18 features is sufficient for detecting ransomware samples within our test set.

Our results highlight that the SVM-L, KNN, Random Forest, and Naive Bayes algorithms are suitable algorithms for ransomware detection since they are able to correctly classify all malicious samples within our test set. In addition, to further demonstrating the effectiveness of using instruction opcodes for detecting ransomware at runtime, these results also show that it is not necessary to track all instructions to enable accurate detection. In other words, only a limited number of instructions are needed to distinguish between benign and malicious applications. A summary of the performance of all machine learning models with the reduced feature set against the quality metrics described in section 2.3 is shown in Table 5.2.

In addition to the previously discussed metrics, we examine the number of false positives

Model	Accuracy	TPR	FPR	Precision	F-score	AUC
Logistic Regression	0.796	0.857	0.273	0.778	0.816	0.792
SVM-Linear	0.978	1	0.045	0.961	0.980	0.977
SVM-Polynomial	0.452	0	0.045	0	0	0.477
SVM-RBF	0.800	0.633	0.023	0.969	0.765	0.805
KNN	0.978	1	0.045	0.961	0.980	0.977
Random Forest	0.989	1	0.023	0.980	0.990	0.989
Naive Bayes	1	1	0	1	1	1

Table 5.2: Summary of quality metrics under different models when using a reduced feature set through PCA.

and false negatives each algorithm produces when using a reduced feature set. Figure 5.4 shows the breakdown of the false positives and false negatives observed when using different machine learning algorithms with the PCA-reduced feature set. Overall, we observe that our detection solution performs better when using a PCA-reduced feature set, yielding fewer false negatives relative to the full feature set approach. However, the PCA-reduced feature set yields a slightly higher number of false positives. On average, using machine learning models with a reduced feature set yields 2.9 false positives with logistic regression exhibiting the highest count of 12 false positives. The remaining algorithms yielded a range of false positives between 0 and 2. On the other hand, the average number of false negatives was relatively low compared to the full feature set approach. On average, 10.6 false negatives were produced. We find that the SVM-P algorithm continues to be the least effective in detecting ransomware under this configuration with 49 undetected samples, followed by SVM-RBF with 18 undetected samples. Furthermore, the Naive Bayes algorithm achieved the best performance by detecting all the ransomware samples present within our test set and yielding no false positives. Random Forest comes at a close second with similar results to Naive Bayes. Random Forest produced one false positive and no false negatives. As such, we consider both of these models to be the most suitable for detection when using a reduced feature.

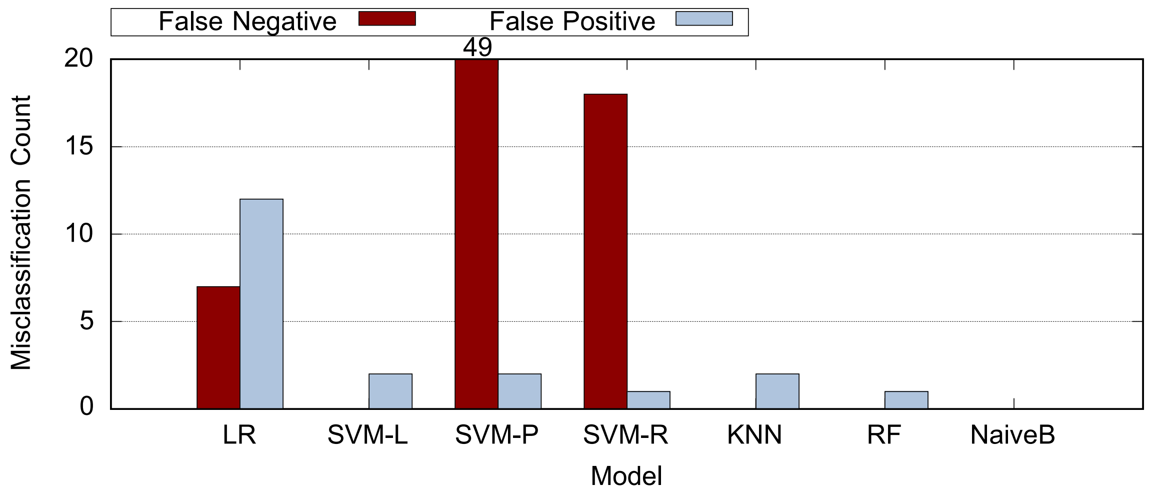

Figure 5.4: False positive and negative misclassifications when combining different machine learning models with a reduced feature set through principle component analysis (PCA).

5.3 Harnessing Cryptographic Instructions

Having the ability to reduce the number of instructions a solution needs to track during execution is important. This is because the machine learning models require less compute resources when using fewer features, and therefore, execute faster. Most importantly, developing solutions that only track a limited set of instructions makes the required hardware support less complex, and therefore, more feasible for implementation in practice. To this end, we explore a different approach for reducing number of features used with our machine learning models. To achieve this, we examine the instruction types that are commonly used in cryptographic functions. Such instructions include rotate instructions (ROL and ROR), shift instructions (SHL and SHR), and the exclusive or instruction (XOR). Using this approach allowed to reduce the number of featuresby 50% relative to what we obtained with the PCA algorithm.

Figure 5.5 summarizes the detection rates of the different machine learning models when using the crypto-reduced feature set which consisted of a total of 9 features. Overall, we observe a slightly better average in the detection rate relative to the PCA-reduced feature set approach. On average, we observe a detection rate of 78.6%, as well as a slightly improved false positive rate

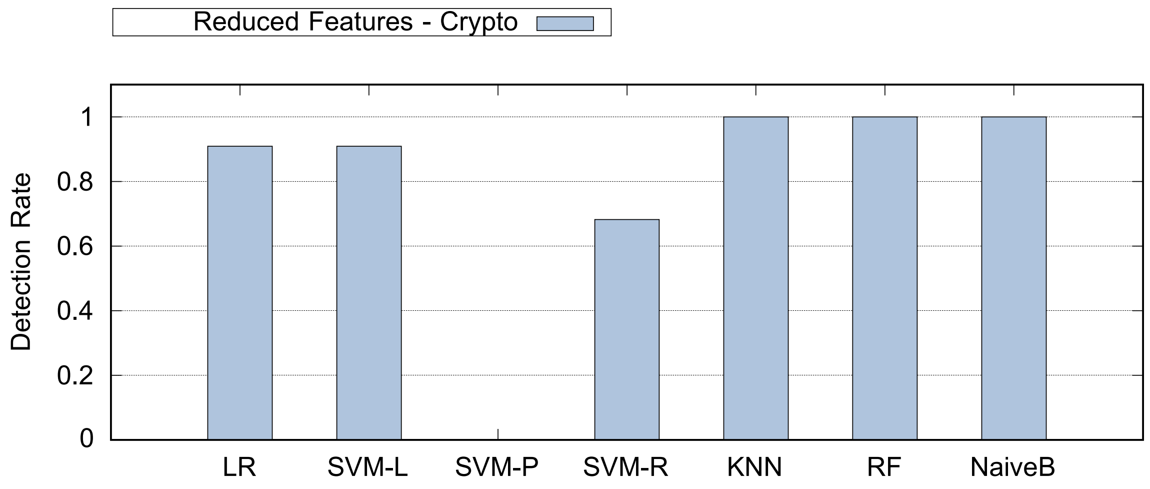

Figure 5.5: Comparison of detection rates of ransomware from 9 families when combining different machine learning models with a reduced feature set that consists of instructions relevant to cryptographic algorithms.

of 4.1% while using our test dataset. We find that with the exception of SVM-P and SVM-RBF, most algorithms achieve a detection rate of 90% or better. More specifically, we find that the KNN, Random Forest, and Naive Bayes algorithms are able to classify all the malicious samples correctly. Such results underscore the fact that tracking instruction types that are commonly used in cryptographic functions which in our study was 9 features, is sufficient for detecting ransomware samples within our test set. We also find that two of the aforementioned algorithms, Random Forest and Naive Bayes, are able to correctly classify benign applications. A summary of the quality metrics for our machine learning models with the crypto-reduced feature set is shown in Table 5.3.

To better understand the performance of each model with the crypto-reduced approach, we examine the number of false positives and false negatives each algorithm produces when using the newly reduced feature set based on cryptographic algorithms. Figure 5.6 shows the breakdown of the false positives and false negatives observed when using different machine learning algorithms with the crypto-reduced feature set. Overall, we observe that our detection solution performs remarkably well using a crypto-reduced feature set, yielding fewer false negatives relative to the

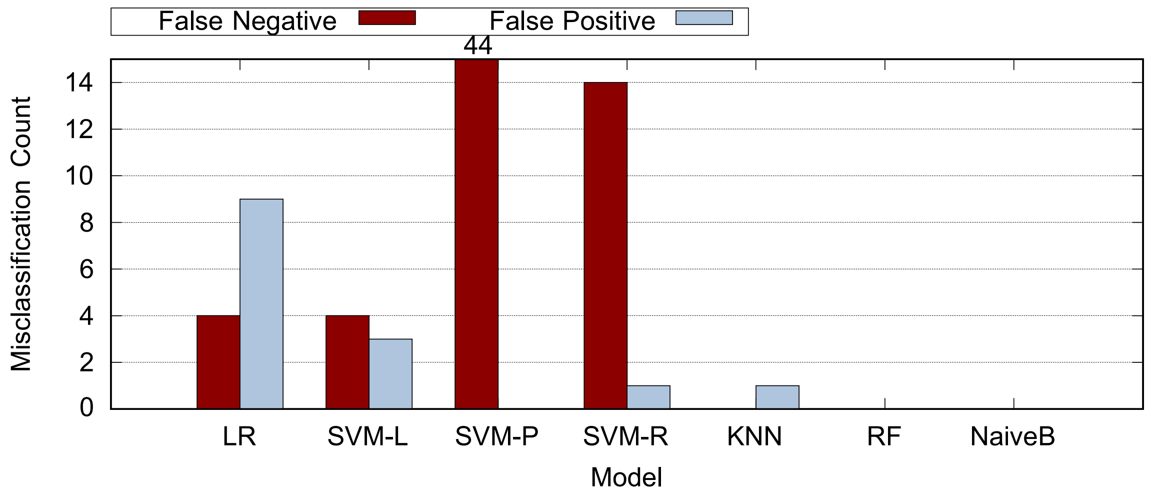

Figure 5.6: False positive and negative misclassifications when combining different machine learning models with a reduced feature set that consists of instructions relevant to cryptographic algorithms.

full feature set approach. On average, using machine learning models with this reduced feature set yields 2 false positives with logistic regression still exhibiting the highest count of 9 false positives. The remaining algorithms yielded a range of false positives between 0 and 3. Similarly, the average number of false negatives was relatively low compared to the PCA-reduced feature set approach. On average, this approach yielded 9.4 false negatives, marking a 1.1% improvement over the PCA-reduced approach. We find that the SVM-P algorithm still continues to be the least effective in detecting ransomware under this configuration with 44 undetected samples, followed by SVM-RBF with 14 undetected samples. Furthermore, the Random Forest and Naive Bayes algorithms achieved the best performance by detecting all the ransomware samples present within our test set and producing no false positives. KNN comes at a close third with similar results for false negatives. However, the KNN algorithm had a slightly elevated false positive count with only one misclassified benign application.

Finally, we find that using cryptographic instructions can be effective against other ransomware types that are not cryptographic based. Although our study focused on evaluating cryptographic ransomware, our dataset included a limited number of locker ransomware and Remote

Model	**Accuracy**	**TPR**	**FPR**	**Precision**	**F-score**	**AUC**
Logistic Regression	0.796	0.909	0.184	0.816	0.860	0.863
SVM-Linear	0.925	0.909	0.061	0.930	0.920	0.924
SVM-Polynomial	0.484	0	0	0	0	0.500
SVM-RBF	0.495	0.682	0.020	0.968	0.800	0.831
KNN	1	1	0.020	0.978	0.989	0.990
Random Forest	1	1	0	1	1	1
Naive Bayes	1	1	0	1	1	1

Table 5.3: Summary of quality metrics under different models when using a reduced feature set that consists of instructions that are relevant to cryptographic algorithms.

Access Trojans (RAT). We believe this is due to the fact that malware in general includes encrypted content such as encrypted payloads and strings. As a result, such programs make use of cryptographic functions during their execution. We believe this is an interesting result that we would like to pursue further as future work.

CHAPTER 6

CONCLUSION

In this work, we characterize the behavior of ransomware at the instruction level. We demonstrate that instruction opcodes of software programs are reliable features for detecting ransomware at runtime. Furthermore, we demonstrate that a minimal set of cryptographic instructions could be harnessed as features for distinguishing benign and ransomware programs. We evaluate the effectiveness of using machine learning models in detecting ransomware with high accuracy and low false positive rates. Finally, we evaluate the robustness of our approach against real ransomware available in a comprehensive dataset.

REFERENCES

[1] N. Lachtar, D. Ibdah, and A. Bacha, “The case for native instructions in the detection of mobile ransomware,” *IEEE Letters of the Computer Society*, vol. 2, no. 2, pp. 16–19, June 2019, doi: 10.1109/LOCS.2019.2918091.

[2] “How to protect your networks from ransomware.” The United States Department of Justice, https://www.justice.gov/criminal-ccips/file/872771/download (accessed Apr. 3, 2021).

[3] “2018 mobile threat landscape.” Trend Micro, https://www.trendmicro.com/vinfo/nz/security/research-and-analysis/threat-reports/roundup/2018-mobile-threat-landscape (accessed Oct. 15, 2020).

[4] A. Greenberg, “The untold story of notpetya, the most devastating cyberattack in history.” Wired, https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world (accessed Feb. 15, 2021).

[5] Y. Shohet, “Ransomware attacks hit manufacturing - are you vulnerable?” IndustryWeek, https://www.industryweek.com/technology-and-iiot/ransomware-attacks-hit-manufacturing-are-you-vulnerable (accessed Jan. 3, 2021).

[6] G. Torbet, “Baltimore ransomware attack will cost the city over $18 million.” engadget, 2019, https://www.engadget.com/2019/06/06/baltimore-ransomware-18-million-damages (accessed Feb. 15, 2021).

[7] S. Morgan, “Global ransomware damage costs predicted to reach $20 billion (usd) by 2021.” Cybercrime Magazine, 2019, https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-20-billion-usd-by-2021 (accessed Jan. 3, 2021).

[8] “Virus total.” VirusTotal, https://www.virustotal.com (accessed Apr. 5, 2021).

[9] “The monero project.” Monero, https://www.getmonero.org/ (accessed Feb. 15, 2021).

[10] C. T. U. R. Team, “Wcry ransomware analysis.” SecureWorks, May 2017, https://www.secureworks.com/research/wcry-ransomware-analysis (accessed Jan. 3, 2021).

[11] “Wannacry ransomware attack.” Wikipedia, https://en.wikipedia.org/wiki/WannaCry (accessed Feb. 17, 2021).

[12] N. Scaife, H. Carter, P. Traynor, and K. R. Butler, “Cryptolock (and drop it): stopping ransomware attacks on user data,” in *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. Los Alamitos, CA, USA: IEEE Computer Society, 2016, pp. 303–312, doi: 10.1109/ICDCS.2016.46.

[13] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, “Uncovering the face of android ransomware: Characterization and real-time detection,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1286–1300, May 2018, doi: 10.1109/TIFS.2017.2787905.

[14] Y. Takeuchi, K. Sakai, and S. Fukumoto, “Detecting ransomware using support vector machines,” in *Proceedings of the 47th International Conference on Parallel Processing Companion*. New York, NY, USA: ACM, 2018, pp. 1–6.

[15] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, “UNVEIL: A large scale, automated approach to detecting ransomware,” in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016.

[16] O. Ami, Y. Elovici, and D. Hendler, “Ransomware prevention using application authentication-based file access control,” in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2018, pp. 1610–1619.

[17] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. Lupu, “Automated dynamic analysis of ransomware: Benefits,” *Limitations and use for Detection. arXiv preprint*, 2016.

[18] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens, and J.-L. Lanet, “Ransomware’s early mitigation mechanisms,” in *Proceedings of the 13th International Conference on Availability, Reliability and Security*. New York, NY, USA: ACM, 2018, pp. 2:1–2:10.

[19] C. Pascariu and I. Barbu, “Ransomware honeypot: Honeypot solution designed to detect a ransomware infection identify the ransomware family,” in *2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2019, pp. 1–4, doi: 10.1109/ECAI46879.2019.9042158.

[20] N. Naik, P. Jenkins, J. Gillett, H. Mouratidis, K. Naik, and J. Song, “Lockout-tagout ransomware: A detection method for ransomware using fuzzy hashing and clustering,” in *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019, pp. 641–648, doi: 10.1109/SSCI44817.2019.9003148.

[21] D. Vidyarthi, C. R. S. Kumar, S. Rakshit, and S. Chansarkar, “Static malware analysis to identify ransomware properties,” *International journal of computer science issues*, vol. 16, no. 3, p. 10–17, 2019, doi: 10.5281/zenodo.3252963.

[22] J. Chen, C. Wang, Z. Zhao, K. Chen, R. Du, and G.-J. Ahn, “Uncovering the face of android ransomware: Characterization and real-time detection,” *IEEE transactions on information forensics and security*, vol. 13, no. 5, p. 1286–1300, 2018, doi: 10.1109/TIFS.2017.2787905.

[23] N. Lachtar, D. Ibdah, and A. Bacha, “Towards mobile malware detection through convolutional neural networks,” in *IEEE Embedded Systems Letters (ESL)*, vol. 19, no. 2, 2020, pp. 126–129, doi: 10.1109/LES.2020.3035875.

[24] J. Ko, J. Jo, D. Kim, S. Choi, and J. Kwak, “Real time android ransomware detection by analyzed android applications,” in *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, 2019, pp. 1–5, doi: 10.23919/ELINFOCOM.2019.8706349.

[25] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, “Paybreak: Defense against cryptographic ransomware,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (Asia CCS)*. New York, NY, USA: ACM, 2017, pp. 599–611.

[26] J. Lee, J. Lee, and J. Hong, “How to make efficient decoy files for ransomware detection?” in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. ACM, 2017, pp. 208–212.

[27] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, “Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 2231–2244.

[28] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, “Shieldfs: a self-healing, ransomware-aware filesystem,” in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. New York, NY, USA: ACM, 2016, pp. 336–347.

[29] M. Baykara and B. Sekin, “A novel approach to ransomware: Designing a safe zone system,” in *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 2018, p. 1–5, doi: 10.1109/ISDFS.2018.8355317.

[30] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, “Paybreak: Defense against cryptographic ransomware,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (Asia CCS)*. New York, NY, USA: ACM, 2017, pp. 599–611.

[31] J. Huang, J. Xu, X. Xing, P. Liu, and M. K. Qureshi, “Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2017, pp. 2231–2244.

[32] X. Jiang and Y. Zhou, “Dissecting android malware: Characterization and evolution,” in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 95–109, doi: 10.1109/SP.2012.16.

[33] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “Madam: Effective and efficient behavior-based android malware detection and prevention,” *IEEE transactions on dependable and secure computing*, vol. 15, no. 1, p. 83–97, 2018, doi: 10.1109/TDSC.2016.2536605.

[34] W.-C. Wu and S.-H. Hung, “Droiddolphin: a dynamic android malware detection framework using big data and machine learning,” in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*. New York, NY, USA: ACM, 2014, pp. 247–252.

[35] H. Cai and B. G. Ryder, “Droidfax: A toolkit for systematic characterization of android applications,” in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 643–647, doi: 10.1109/ICSME.2017.35.

[36] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, “On the feasibility of online malware detection with performance counters,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: ACM, 2013, pp. 559–570, doi: 10.1145/2485922.2485970.

[37] M. Kazdagli, V. J. Reddi, and M. Tiwari, “Quantifying and improving the efficiency of hardware-based mobile malware detectors,” in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, vol. 2016-. IEEE, 2016, p. 1–13, doi: 10.1109/MICRO.2016.7783740.

[38] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, “On the detection of kernel-level rootkits using hardware performance counters,” in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2017, pp. 483–493.

[39] M. Ozsoy, C. Donovick, I. Gorelic, N. Abu-Ghazaleh, and D. Ponomarev, “Malware-aware processors: A framework for efficient online malware detection,” in *International Symposium on High Performance Computer Architecture (HPCA)*, February 2015, pp. 651–661, doi: 10.1109/HPCA.2015.7056070.

[40] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, “Ensemble learning for low-level hardware-supported malware detection,” in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 3–25.

[41] N. Lachtar, A. A. Elkhail, A. Bacha, and H. Malik, “An application agnostic defense against the dark arts of cryptojacking,” in *51st International Conference on Dependable Systems and Networks (DSN)*. Taipei: IEEE/IFIP, Jun. 2021, pp. 413–426.

[42] N. Lachtar, A. Abu Elkhail, A. Bacha, and H. Malik, “A cross-stack approach towards defending against cryptojacking,” in *IEEE Computer Architecture Letters (CAL)*, vol. 19, no. 2. IEEE, 2020, pp. 126–129, doi: 10.1109/LCA.2020.3017457.

[43] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, “Rhmd: evasion-resilient hardware malware detectors,” in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. New York, NY, USA: ACM, 2017, pp. 315–327.

[44] P. P. Chan and W.-K. Song, “Static detection of android malware by using permissions and api calls,” in *Machine Learning and Cybernetics (ICMLC), 2014 International Conference on*, vol. 1. IEEE, 2014, pp. 82–87, doi: 10.1109/ICMLC.2014.7009096.

[45] M. Leeds, M. Keffeler, and T. Atkison, “A comparison of features for android malware detection,” in *Proceedings of the SouthEast Conference*. New York, NY, USA: ACM, 2017, pp. 63–68, doi: http://dx.doi.org/10.1145/3077286.3077288.

[46] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, “A multimodal deep learning method for android malware detection using various features,” *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 773–788, 2019, doi: 10.1109/TIFS.2018.2866319.

[47] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, “Cnn-based android malware detection,” in *2017 International Conference on Software Security and Assurance (ICSSA)*. IEEE, 2017, pp. 60–65, doi: 10.1109/ACCESS.2020.3028370.

[48] A. Tal, “Intel software development emulator.” Intel Inc., 2020, http://www.intel.com/software/sde (accessed Apr. 5, 2021).