Effective Crypto Ransomawre Detection Using Hardware Performance Counters

John Podolanko

Department of Computer Science & Engineering

The University of Texas at Arlington

Supervisor

Jiang Ming, PhD

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

May 2019

Abstract

Systems affected by malware in the past 10 years has risen from 29 million to 780 million, which tells us it is a rapidly growing threat. Viruses, ransomware, worms, backdoors, botnets, etc. all come under malware. Ransomware alone is predicted to cost $11.5 billion in 2019. As the downtime, data loss, and financial damages are rising, researchers continue to look for new ways to mitigate this threat. However, the common approaches have shown to yield high false positive rates or delayed detection rates resulting in data loss. My research explores a dynamic approach for early-stage ransomware detection by modeling its behavior using hardware performance counters with low overhead. The analysis begins on a bare-metal machine running ransomware which is profiled for hardware calls using Intel® VTune™ Amplifier before it compromises the system. By using this approach, I am able to generate models using hardware performance counters extracted by VTune™ on known ransomware samples collected from VirusTotal and Hybrid Analysis, and I use that data to train the detection system using machine learning techniques. I have shown that hardware performance counters can provide effective metrics for use in detecting and mitigating the ever-growing ransomware threat faced by the world while ensuring no data is lost.

Acknowledgements

The author thanks the supervisory committee for all their guidance, support, and patience. The author also thanks the University of Texas at Arlington and the Department of Education for their support with a Graduate Assistance in Areas of National Need (GAANN) fellowship.

Contents

Chapter 1

Introduction

Ransomware is a special type of malicious software (malware) which infects a system and limits a users access to the system and its files and extorts the user to pay a ransom with the promise of recovering their files. Ransomware accomplishes this by encrypting the users files and locking the users desktop preventing all but the required functionality to make payment to the attacker. Cyber criminals gave birth to ransomware within this past decade, and ransomware has quickly become one of the most critical threats to cyber security. New and recycled ransomware programs are infecting computers across the globe every single day, and they are generating enormous profits for the cyber criminals. Despite the fact that a majority of ransomware victims refuse to pay the ransom, these cyber criminals are raking in billions of dollars annually. As a result, cyber criminals are exploring new ways to extort money from the users and acclimating to the defenses being developed to combat ransomware. In 2017, the ransomware landscape shifted dramatically with the emergence of two new self-propagating threats – WannaCry and Petya [1]. WannaCry attacked known Windows SMB network vulnerabilities using exploits like EternalBlue (courtesy of a leak by the National Security Agency), which allowed an intruder to exploit a remote code execution vulnera-

bility on a targeted system [1]. WannaCry made global headlines after infecting more than 230,000 systems in over 150 countries and causing an estimated $5 billion in damages [1], and those numbers are just what was reported to authorities. Health care companies, in particular, were the most negatively impacted by WannaCry as their patients' lives were immediately put in jeopardy by the loss of several systems. Soon after WannaCry came Bad Rabbit which notably targeted Ukraines Ministry of Infrastructure and Kievs public transportation system.

In contrast to typical malware, ransomware behaves much differently. For example, traditional malware generally maintains a goal of achieving a level of persistence, creating a backdoor for a cyber criminal to utilize, or it tries gather information such as credentials for financial websites, keystrokes or a users webcam stream and send it to a remote command and control server without arousing suspicion. Ransomware, in contrast, does not create backdoors, nor does it collect private information. Rather, ransomware creates a denial of service of a system to its own user by encrypting its files and/or locking the desktop making all access unavailable. Comparatively, ransomware also makes itself known to the user unlike traditional malware.

Given the rapid growth of ransomware attacks [1], it is critical to develop a technique to protect users from ransomware exposure and exploitation. To accomplish this, however, we require a low-level understanding of how ransomware interacts with a users system before we can build proper defenses in order to protect the users from this growing threat. Many existing ransomware detection tools focus mainly on their own levels of sophistication and their incremental improvements over ransomware attacks, but they lack a more practical and dynamic approach in ransomware detection by only focusing on keeping their signature definitions up-to-date and only performing the most basic of analysis. In this paper, I take a deeper dive and dynamically analyze the key functional properties of

ransomware and its effects on a system.

Today, behavior-based malware detection can be achieved because of the breakthroughs in dynamic analysis. Code-based static analysis analyzes the suspicious files by examining its static properties that consists of header details, hashes, packer signature, date of creation, etc. A static approach for malware detection is usually performed on binary files without executing it and disassembling it using an interactive disassembler. In contrast, dynamic analysis is carried out by keenly observing the behavior of the malware while executing it on the system. It provides new insights by tracking the changes in the system as well as any unusual behavior. In dynamic analysis, certain changes in the system should raise a flag alerting the administrator of the files that have been modified and/or added and deleted, new processes that are running, registry modifications indicating what changes took place in the system, any new services or applications that has been installed, and any modification of certain system settings. While both static and dynamic analysis strive to accomplish the same goal, the dynamic approach is often performed in virtual sandbox environment to prevent the malware from actually infecting a real users files. By using a sandbox environment, a virtual machine can easily be rolled back to a previous state once the malware has completed and been analyzed. Unfortunately, typical behavior-based malware detection generally fails to detect ransomware because it is either looking for the placement of backdoors or the gathering of private information. It also fails because ransomware engages in activity that appears similar to benign applications that use encryption or compression. Moreover, many behavior-based systems look for very specific behaviors are constantly plagued by misclassification of ransomware as seen in several anti-virus products [2; 3].

In this paper, I present a behavior-based dynamic analysis model to detect ransomware attacks and more effectively define its behavior. In this model, I

build a system that contains a realistic, artificial user environment in which ransomware samples are executed and their interactions with the system environment monitored and recorded at the CPU level. The recorded interactions of the ransomware with the file system allows the model to identify cryptographic ransomware behavior. For a ransomware attack to achieve success, ransomware simply requires access to the users files, and then it locks the system leaving it inaccessible. In my approach, many ransomware samples are analyzed allowing for detection of ransomware by observing the CPU behavior during the execution of a ransomware executable.

This approach utilizes the Hardware Performance Counters (HPCs) of a CPU to analyze and create dynamic signatures for detecting ransomware. As malware authors get more clever, they find more ways to evade software-based detection and analysis methods through actions like obfuscation, packing, etc. The one thing they cannot evade is the hardware requirement. I use a non-virtualized, bare-metal DeepFreeze sandbox which creates a safe environment for executing untrusted and potentially malicious executables that prevents them from spreading and does not sacrifice a users files or private information. DeepFreeze preserves a system's state allowing us to recover that state by simply restarting the system. In the meantime, I am free to install ransomware and wreak havoc on the system without fear of data loss or any threats persisting beyond a restart. In order to obtain data on ransomware behavior, I utilized a software suite by Intel called VTune™ Amplifier 2018 that records the low-level HPCs of the CPU, and those records are extracted prior to a restart. After analyzing the hardware performance counter metrics on over 7,000 ransomware samples, I managed to create a set of hardware metrics that indicate the presence of running ransomware on a system.

In my research, I analyzed over 7,000 of the latest ransomware samples from

over 15 families that encrypt a user's file system, and this approach allowed me to determine a set of hardware performance metrics that can indicate the presence of running malware on a system. Using these metrics, I was able to effectively detect 99.28% of ransomware samples from all ransomware families with a false positive rate of 0.36% and a system accuracy rate of 99.59%. The detection and accuracy rates of this model suggest that it performs better than the current behavior-based analysis systems in regard to identifying and detecting ransomware samples expediently [3]. This model can easily be deployed and used on any malware analysis system such as an anti-virus software suite.

Chapter 2

Background

Ransomware is predicted to have a global damage cost of over $8 billion as of 2018 which is almost 25 times the 2015 global damage cost of $325 million [4]. That's a pretty good indicator that ransomware is here to stay and that we haven't seen the last of it. Recent attacks such as WannaCry, Petya, and BadRabbit have caused damage on a global scale and have touched nearly every country in the world. These recent attacks were able to easily infect systems across the globe in large part due to the public release of classified government hacking tools and exploits from the National Security Agency Equation Group and the Central Intelligence Agency Vault 7 that were put to malicious use by hacking groups such as The Lazarus Group [5].

In its simplest form, ransomware is a type of malware that extorts money from an unsuspecting user by preventing the user's access to their files and applications. That said, there are dozens of different ransomware families and many have different methods of extortion. Some simply lock a user's desktop with a splash screen preventing the user from accessing their files until payment is made at which point a recovery code may (or may not) be delivered. Most families, however, encrypt the user's files and will only provide the decryption key after

payment is made. Early versions of encrypting ransomware used symmetric key encryption where the same key is used for both encryption and decryption. However, it didn't take the anti-malware community long to reverse engineer and counter symmetric key encrypting ransomware [6], so malware authors began using asymmetric key encryption in conjunction with symmetric key encryption where a randomly generated symmetric key (also called a session key) is used to encrypt the user's files and a public key is used to encrypt the symmetric key. In asymmetric key encryption, the knowledge of the public encryption key doesn't allow users to decrypt the symmetric key or anything else, therefore, the attacker will release the private key only after receipt of payment which can be used to decrypt the symmetric key used for decrypting the user's files.

While the specifics of cryptography and algorithms vary between ransomware families, most of them use Microsoft's CryptoAPI which is included in every Windows installation. CryptoWall and CryptoLocker both maliciously use the CryptoAPI and are among the most successful ransomware families. By using Microsoft's CryptoAPI, it allows the malware authors to minimize the size of their payload. In contrast, other families of ransomware have been known to package entire third-party encryption programs inside their executables. For example, Mamba packages DiskCryptor inside their executable and targets the entire disk instead of just the files [7]. While this creates a much bulkier executable, it does potentially allow Mamba to slip past anything closely monitoring Microsoft's CryptoAPI for malicious behavior. Mamba also provides an additional challenge to the anti-malware community considering the wrong signature for Mamba may result in false positives and shut down legitimate installations of DiskCryptor on non-infected systems.

As with all other malware, ransomware authors continuously find new techniques to evade detection, use new programming languages, create new naming

conventions, and use social engineering tricks to pressure victims into sending large sums of money to the attacker. One of the latest techniques involves packaging ransomware in RarSFX executable files [8]. Ransomware also uses number of different strategies to increase its potentially harmful behavior and attack user. For example, it can corrupt the users file in multiple ways either by using standard encryption, overwriting the file table, corrupting file headers, and even hijacking the boot sector. Alternatively, it can inject a single or multiple malicious processes, or it may attempt process replacement of a legitimate process. It may steal a users private information and send it to a third party for extortion. It may also establish a remote shell connection with command-and-control servers which are used by attackers to establish and manage secure transmission with the targeted systems from across the Internet. Our approach focuses on detecting the existence of any and all families of crypto ransomware based on the expectation that full data recovery should be relatively simple and achievable for all other types of non-encrypting ransomware even in the absence of a data backup/recovery policy.

Ransomware behaves very differently than other traditional forms of malware. Namely, it makes itself known to the user. On the other hand, ransomware also tends to behave in a very specific manner. To break it down, ransomware generally begins by connecting to a command and control server through the Internet. It then either generates an encryption key which it sends to the server or receives an encryption key from the server. Ransomware then begins a scan of directories. Some scans will scan all disks including network drives while others may just focus on a user's personal files. After scanning, the encryption key is used to begin encrypting the user's personal files and potentially all other files on the user's system. Everything up to this point is generally done without the knowledge of the user and remains that way until the encryption phase is complete. At this point, the ransomware makes itself known and locks the user's

desktop with a ransom note demanding some form of cryptocurrency. While the crypto libraries or specific files targeted may differ, ransomware still performs the same basic functionalities.

While this may be attractive to behavior-based malware analysts, this behavior differs from benign encryption programs to such a small degree that typical behavior-based analysis produces an unsatisfactory number of false positives. Furthermore, some approaches [3; 6; 9; 10] still accept a degree of file loss before detecting and terminating a malicious ransomware process due to the high-level nature of the metrics used for analyzing behavior. My research differs from other approaches in that it uses low-level HPCs which can detect the subtle differences between ransomware and benign encryption software that is lost on other behavior-based analysis techniques. My research also strives to detect ransomware with zero data loss.

2.1 Hardware Performance Counters

Over the past decade, HPCs have started making their way into the sights of malware researchers across the globe and have received a lot of mixed feedback. HPCs are special micro-architectural registers that are built into most processors to store metrics of hardware and software events such as cache hits and misses, branch prediction accuracy, instruction count for each instruction, snapshots of the call stack, system call count, and dozens of other metrics [11]. They were originally designed for hardware debugging and verification purposes, but their utilization has since grown to serve many other purposes. CPU scheduling, performance tuning, integrity checking, and workload pattern identification are among these utility purposes [12; 13; 14; 15].

In effort to expand the utility of HPCs, a the performance monitoring unit

(PMU) has been added to most processors by their manufacturers to allow applications to access performance monitoring counters at reduced overhead [16; 17]. PMUs typically operate under an interrupt-based working mode in which the system is interrupted when specified events cross a predefined threshold or at predefined time intervals [11]. This means event-based sampling and time-based sampling are achievable.

HPCs can also be used to model program behavior. This has been a pivotal step allowing software testers, reverse engineers, and even malware analysts to produce unique signatures or fingerprints for individual software applications [18].

Chapter 3

Related Works

In [12] , Malone et al. propose using HPCs to detect malicious program modifications at load time and runtime by acting as dynamic integrity checkers. The main benefit of this is that it incurs almost no hardware cost since theyre built into most processors. The authors claim that HPCs are very efficient at detecting program modifications.

The biggest limitation to this research is that they tested their approach on programs running solo instead of in a dynamic environment where multiple programs are running simultaneously. Also, this only serves to protect benign programs and does nothing to detect standalone malware. Given that ransomware is generally standalone [19], this approach would require significant modification before it could expand its capabilities.

In [20] , Tang et al. continue previous works on HPCs and use them to detect anomaly-based malware by looking at micro architectural execution patterns. The authors approach goes beyond that of recent works in behavior-based malware detection to detect a much wider range of malware to include zero-day by using machine learning to establish a baseline of benign program executions and use them to detect deviations, and the detector can be used in complement

to existing signature-based detections.

Because the approach of using HPCs for malware detection is a relatively new idea, it is still far from production-ready. It is prone to a high false-positive rate, and it also requires baselines for every individual program on a computer to detect anomalies in the benign programs themselves in the event they are exploited or are under attack. This doesnt do much for standalone malware detection.

The feasibility of building a malware detector in hardware using data from existing HPCs is examined by Demme et al. in [18] . In this paper, the authors find that they can detect multiple variations of malware within families with ease given a small control set. They also propose modifications in hardware which allow the malware detector to run smoothly beneath the system software which is a great improvement over and fewer bugs than existing software-based antivirus solutions. When used in combination with software-based antivirus, the authors claim that their approach advances the state of the art in online malware detection.

This paper provides a lot of solid insight and can serve as a foundation on which many future works can be built. As with anyone who develops malware-fighting solutions, the authors voice concern over the potential for malware authors to once again adapt their programs to combat even hardware-based approaches such as the one discussed in their paper, but on the other hand it is good to see the hardware community finally join the fight against malware because it forces malware authors to step into a new arena in which they lose the advantage they have had over the anti-malware developers in the software arena since Brain was first discovered.

In [21] , Kharraz et al. view the evolution of ransomware in the wild from 2006-2014 and determine that the sophisticated destructive capabilities of most ransomware families lacks growth despite the improvements of encryption, dele-

tion and communications techniques in general. They insist that stopping advanced ransomware attacks is not as complex as commonly believed and that defenses involving file system monitoring can be practical and effective because ransomware generates file system requests much differently than benign programs.

While the file system monitoring approach is indeed practical, it alone is not enough due to overwhelming percentage of ransomware samples analyzed by the authors that produced some amount of data loss before being detected. Other than that, their analysis of the destructiveness of ransomware is thorough and was very useful in developing my own approach to combating ransomware.

Kharraz et al. take this work even further in [3] by focusing on the difference between ransomware and all other existing families of malware. In the malware arena, ransomware stands alone in comparison with all the rest which is why nearly all generic malware detection systems are losing the fight against ransomware. The authors create a dynamic analysis system called UNVEIL which is designed to specifically detect ransomware and is to be used in combination with other malware detection systems. UNVEIL essentially generates a honeypot environment and detects ransomware as soon as it interacts with the users data. UNVEIL also monitors the desktop to detect any ransomware-like behavior such as a lock screen preventing the user from accessing their files. The authors boast that UNVEIL greatly improves upon the state of the art by demonstrating its capability to identify known evasive ransomware currently immune to detection by existing anti-virus systems.

This paper really does improve upon the state of the art given a zero false-positive rate from over 13,000 samples and detects superficial and sophisticated as well as zero-day ransomware attacks. One limitation of the paper is the potential for ransomware to detect the artificially generated environment to avoid it. While it certainly raises the bar of difficulty for the ransomware author to circumvent,

it is not infeasible. The other limitation is that most ransomware samples still incur some amount of data loss before detection.

In [10] , Kharraz et al. introduce Redemption, a defense that allows additional resilience to ransomware using a slight modification of the host's operating system. This modification maintains a transparent buffer for all storage I/O while monitoring I/O request patterns of each process for behavioral metrics of ransomware called the Malice Score Calculation (MSC). In the event ransomware is detected, the authors claim that Redemption can kill the process and restore the data resulting in no data loss while only sacrificing modest overhead which averages out at 2.6% CPU utilization.

Kharraz et al. build on their two previous works and continue to push the envelope in the fight against ransomware. They boast a true positive rate of 100% and a false positive rate of 0.5% which is achievable based on a specific MSC calculation. They worked with a very large dataset of over 9,000 ransomware samples and even tested Redemption against benign applications that most commonly produce false positives among other ransomware detection approaches. In contrast, Redemption's monitor and the MSC function are not immune to attack. They can easily be bypassed and tricked with minimal effort, or an attacker can simply overwhelm the user with alert messages to the point the user turns off protection. This also doesn't take into account that Redemption is a kernel-level process which, if exploited, can result in far greater damage to the system.

In [9] , Alam et al. approach ransomware prevention using HPCs. Their two-step approach used the Linux tool, `perf`, to establish a system baseline of a Windows virtual machine sandbox before setting the ransomware loose. Once the ransomware was executed in the sandbox, they determined the deviation from the baseline to create their metrics for detection.

While Alam et al. use the same high-level approach as us – using HPCs as a

method to detect and prevent ransomware – my methods differ greatly. For one, Alam et al. use the `perf` tool in Linux to collect baseline deviation metrics on a Windows virtual machine while I used Intel® VTune™ and Faronics DeepFreeze on a non-virtualized Windows system which is able to take far more detailed metrics of only the ransomware process and its subprocesses. Their approach also comes with some data loss, and they propose a backup/recovery method for ensuring data persistence. My approach, on the other hand, detects ransomware without the loss of a single file. Lastly, the only information I know about the sample set used in [9] was that they used WannaCry. They didn't specify the number of samples analyzed nor any other families of ransomware analyzed, so their conclusion is not supported by a significant representation of the ransomware population.

In [22] , Zhou et al. attempt to put the question to rest regarding whether HPCs can detect malware. They compare their own experiments to eight other related works while avoiding the drawbacks mentioned in the those related works to quantitatively analyze how those drawbacks led the authors of those other works to the conclusion that HPCs can be used as reliable detection metrics.

Their main claim is that, to better simulate a real-world scenario, they test their models with measured hardware performance counter values from programs that have not been observed during training. This ensures that malware samples from the same family of malware are available during training but not the exact same malware that a user may encounter. Otherwise, machine learning is not needed because the hash can be used. They claim that, in this real-world training approach, HPCs cannot distinguish between benign and malicious software. However, their results do not support this claim. In their Random Forest model, for example, the Area Under Curve percentage was reduced from 91.8% to 89.9% when switching to the real-world training approach. This is not a large enough

reduction to claim HPCs cannot distinguish between benign and malicious software. The two main differences in their machine learning approach from mine is the aforementioned real-world training approach and their use of Principal Component Analysis dimensionality reduction on their samples.

Lastly, they claim that all but one of their tested machine learning algorithms — Decision Tree, Naive Bayes, Neural Network, AdaBoost, Random Forest, and K-Nearest Neighbor — when used against ransomware resulted in a precision of 0% (AdaBoost showed 50.85%). This is very misleading because they embed a single ransomware sample into an otherwise benign program, Notepad++, and that embedded function encrypts one file every 5 seconds. In other words, the infected Notepad++ will behave like a benign Notepad++ a vast majority of the time while occasionally performing some ransomware functionality. This single sample easily skews the results to favor the authors' claim.

In [6] , Kolodenker et al. propose a new system, PayBreak, to effectively combat ransomware and prevent any data loss. It does this by essentially creating a key escrow inaccessible to ransomware that holds every key used in encryption in a secure manner thus allowing the decryption of any files encrypted by ransomware. PayBreak demonstrates the ability to restore all files lost to twelve different ransomware families, and it does so with negligible performance overhead.

While complete data recovery or complete prevention of data loss is the ideal result of combating ransomware, PayBreak only manages to effectively work with only 60% of all ransomware families leaving eight common families of ransomware that can decimate a users system to go uninhibited. PayBreak also lacks a basic robustness allowing it to be evaded simply by ransomware authors rolling back to older versions of crypto libraries or through basic obfuscation and evasion techniques as stated by the authors themselves. Their approach was essentially

just a proof-of-concept, and it is uncertain whether the authors will pursue any future work on PayBreak or not.

Chapter 4

Proposal and Hypothesis

In this section, I outline ten requirements which have to be met in order to make a significant contribution to the ongoing fight against ransomware, propose an experimental ransomware detection model, and form a hypothesis on the ability of my model to deliver all ten requirements.

4.1 Requirements

In order to make a significant contribution to the fight against ransomware, I derived a list of requirements which needed to be met to set my work apart from all others. I also believe that these requirements should be the standard for anyone looking to enter and/or continue the fight against ransomware. These requirements are:

1. A ransomware detection model must not rely on signature-based ransomware detection only.

2. A ransomware detection model must demonstrate the ability to detect zero-day ransomware.

3. A ransomware detection model must demonstrate the ability to detect ransomware from all ransomware families.

4. A ransomware detection model must achieve a true positive detection rate of 99.9% or greater.

5. A ransomware detection model must achieve a false positive detection rate of less than 0.5%.

6. A ransomware detection model must achieve an accuracy rate of 99.5% or greater.

7. A ransomware detection model must ensure absolutely **no data loss** or guarantee 100% data recovery.

8. A ransomware detection model must employ defensive techniques to prevent malicious tampering or detection evasion.

9. A ransomware detection process must be transparent to the user, but it can present detections and alerts to the user upon request.

10. A ransomware detection process must be able to terminate any discovered ransomware processes.

After carefully analyzing a number of related works, it was quickly made clear that there is no one solution to fit all requirements. Most of those works achieved five of those requirements or fewer. Only the latest works of Kharraz et al. [10] demonstrated the ability to meet seven out of ten requirements. There was no mention of a detection accuracy (#6), and the absence of defensive techniques to protect their model (#8) and lack of user transparency (#9) were the only things preventing them from meeting all ten requirements.

4.2 Proposed Ransomware Detection Model

The most important feature in my model is ensuring absolutely no data loss. The guarantee of full system integrity will set this model apart from all other existing ransomware detection models. Accomplishing this requires that I monitor and analyze the CPU hardware performance counters and classify a ransomware sample before the first file becomes lost to the user — either by encryption of an original file or deletion of an unencrypted original file. Once full system integrity is achieved, my model will focus on delivering the most accurate results achievable by using a collection of supervised machine learning classification techniques to ultimately detect ransomware in its earliest stages upon being executed.

In order to set a ransomware detection threshold, machine learning will be used to analyze thousands of existing ransomware samples and find unique metric ranges for hardware performance counters that are indicative of ransomware. These metric ranges will include branch prediction hit/miss percentages, functions in the call stack, frequency of calls, cache hit/miss percentages, common values and memory addresses found in registers, and other memory values. Once those metrics have been determined, they could potentially be integrated into anti-virus software with negligible impact on system performance to greatly enhance ransomware detection capabilities. These metrics can be tuned over time as ransomware evolves and pushed with regular updates. For the scope of this paper, I will prove that accurate detection of ransomware while achieving zero data loss is feasible.

4.3 Hypothesis

Due to the uncertain nature of machine learning algorithms, I considered requirements #4, #5, and #6 to be the most difficult three to achieve simultaneously

within a single machine learning model. As such, I believed my model could achieve two of those three requirements with relative ease. Overall, I hypothesized that this model would achieve a minimum of eight out of ten requirements upon the successful development and implementation of my model.

Chapter 5

Approach

In this section, I outline and analyze the pros and cons of several approaches and environments for performing dynamic malware analysis and machine learning techniques before finally selecting the optimal approach for our experiments.

5.1 Selecting A Malware Analysis Environment

To analyze ransomware samples, I required a sandbox in which I could perform static and dynamic analysis. There are some added benefits to creating a virtualized sandbox using dynamic analysis tools such as Cuckoo Sandbox [23] and VMRay Analyzer. For one, they provide the ability to take multiple snapshots and screenshots, map the process and subprocesses, perform detailed logging, simulate the Internet, and analyze all incoming and outgoing network traffic. In addition, these features can be automated and customized which is great for dynamic analysis of the malicious binaries. The downside to using a virtualized sandbox is that using hardware performance counter tools can be limited or rendered ineffective. Also, special care and configurations are commonly required to get past anti-virtual machine techniques commonly written into malware samples

to evade dynamic analysis [24]. These techniques range from:

1. Checking for files, directories, or registry keys that indicate the existence of virtualization software such as `C:\Program Files\VMware Tools`.

2. Checking for known MAC address prefixes assigned to virtual network interfaces.

3. Checking for processes indicating a virtual machine such as `Vmtoolsd.exe` and `vboxservice.exe`.

The other option was to use a standalone, potentially air-gapped, bare-metal system which – if designed incorrectly – could be hazardous to other systems on the network. In addition, air-gapping a system for malware analysis may cause the malware process to terminate before performing signature functionality if it cannot communicate with a command and control server via the Internet. The positives to using a standalone, bare-metal system is that it is immune to anti-virtual machine techniques. It also works much more effectively with HPCs to produce accurate results.

5.2 Selecting A Hardware Performance Counter Tool

One of the tools I considered for collecting HPCs is Performance Monitor (or PerfMon) by Microsoft Windows SysInternals. PerfMon runs on all versions of Windows since XP and has the ability to map processes, collect CPU and memory utilization statistics, and log changes to a hard drive. Furthermore, it is easy to learn and intuitive to operate. On the downside, the metrics it collects are high-level and not detailed enough to create signatures for binaries.

I briefly considered using `perf` in conjunction with the Cuckoo Sandbox, but there were too many factors that would compromise my results. For one, `perf` is a Linux-based tool, and all of the ransomware samples I analyzed could only be run on Windows, so it couldn't be run within the virtual machine itself. Even if I used `perf` on the host machine to target the ransomware subprocess spawned inside the Cuckoo virtual machine, it didn't have the ability to follow additional subprocesses spawned by the initial subprocess. If I attempted to capture HPCs for the entire system before starting the dynamic analysis and then calculating the difference after starting the dynamic analysis, the numbers would still be skewed due to the operations performed by the Cuckoo Sandbox outside of the ransomware binary itself.

The final tool considered was Intel® VTune™ Amplifier 2018. With this tool comes a very large learning curve, and the inability to collect accurate data if installed inside a virtual machine. On the other hand, VTune™ can record 27 low-level metrics for CPU performance alone as opposed to simple utilization statistics recorded by the other aforementioned tools. These metrics include cache hit and miss percentages, branch prediction hit and miss percentages, snapshots of the call stack, the number of times each system call is made, and several others. In addition, VTune™ can be configured for collecting full-system metrics as well as metrics for a unique process which can follow and record metrics for any subprocesses spawned by the parent process.

5.3 Selecting Machine Learning Classifiers and Environment

Detection and the binary classification of ransomware and other benign binaries required supervised learning algorithms. Of those, considered using k-Nearest

Neighbor (kNN), Support Vector Machine (SVM), Random Forest, Decision Tree, and Naive Bayes. I decided to pick three for comparative reasons.

The classifiers with the highest accuracy are kNN, SVM, and Random Forest, and as such they were immediately top contenders. This is because they all avoid linear regression and make few or no expectations and assumptions about the data distribution. On the other hand, Decision Tree and Naive Bayes provide more simplified results, and – given the binary nature of classification (ransomware or not ransomware) – they are much easier to implement, require less complexity in the model, and are computationally cheap allowing for quicker classifications meaning faster detections.

5.4 Final Selections

Because my research focuses on ransomware detection based on HPCs than the actual analysis of the ransomware binaries themselves, it was absolutely critical that I obtained the most low-level, accurate metrics possible for each ransomware sample. For this reason alone, I selected VTune™ as my HPC collection tool. In order to optimize the data collection from VTune™, I were required to sacrifice the more desired dynamic analysis environments – Cuckoo Sandbox and VMRay – and go with the standalone, bare-metal system.

I picked three machine learning classifiers for comparative purposes: kNN, SVM, and Random Forest. Among all other classifiers, SVM is often hailed as the most accurate easily making it worth running my results data through. It also handles feature mapping of very large data sets with relative ease. Random Forest was chosen for the same reasons but didn't require feature mapping making it a good contrast to SVM. Lastly, I picked kNN because it is commonly used for predictive classification and because of its close adherence to the real world

in that it doesn't follow linear regression models because real world data rarely follows theoretical assumption guidelines. Despite its lack of a learning phase, kNN still yields relatively high accuracy. The learning machine also had ample CPU power and memory to sustain the computationally expensive resource costs.

Decision Tree learning was strongly considered but ultimately discarded due to its tendency to "overfit" the results based on expectations and because the order in which the input is fed into the system can have a dramatic impact on the tree structure. Those characteristics has a negative impact on the accuracy of the results, and accuracy is critical to success as mentioned in the aforementioned Requirements section. Naive Bayes was also discarded due to its nature to make assumptions regarding the shape of the data distribution.

Chapter 6

Experiment Architecture

To begin my experiment, I created a repository of 7,234 known ransomware samples obtained from VirusTotal and Hybrid Analysis. This repository includes samples from all known crypto ransomware families that affect Windows-based computers such as Bad Rabbit, Cerber, CryptoLocker, CryptoWall, Crysis, Mamba, NotPetya, Petya, WannaCry, and others. Each individual sample was run through a VirusTotal analysis to ensure that it was indeed ransomware and for classification of its specific ransomware family.

To analyze each sample, I created a Windows 7 Professional 64-bit image and cloned it across several laptops. The image was built to look like a real user's system and not a basic, freshly installed Windows system. I installed the latest versions of 7-Zip, Adobe Flash, Adobe Reader, Java, iTunes, Spotify, Dropbox, Google Drive, Google Chrome, Mozilla Firefox, Discord, BitTorrent, Skype, Microsoft Office 2013, and even the latest Windows Updates and Windows Defender signatures. Although, Windows Defender was deactivated for the analysis. I also added 15 gigabytes of random personal files into the user's home directory structure which included music, photos, documents, downloaded executables, and other random files.

To maximize efficiency in collecting HPCs for 7,234 ransomware samples, I installed Intel® VTune™ Amplifier 2018. In addition, I installed Faronics Deep-Freeze to create a recoverable state on each system which could be recovered by performing a restart of the machine. This recoverable state could withstand complete file system encryption and desktop locking. This image was cloned onto four Dell Latitude E5720 laptops with Intel Core i3 2.0 GHZ CPUs, 16 GB SDRAM, and 240 GB solid state drives.

Each sample was run through Olly Debugger (OllyDbg) v1.1 to look for functions that would hinder my analysis such as wait timers and logic bombs. In the event that these things were discovered, I loaded the sample into IDA Pro v7.0 and overwrote those functions with NOP (0x90) instructions or rewrote the addresses of some of the jump calls to bypass any functions that hindered a quick execution. I then ran the ransomware sample through OllyDbg again to verify that the critical funcationality had not been compomised. This was necessary for 529 samples.

To collect HPCs on the ransomware samples, I booted the laptop and inserted a USB flash drive containing the sample for analysis. I opened up VTune™ and configured it to run in HPC mode for a maximum of 180 seconds to ensure that all of the initial steps performed by ransomware were captured and measured up to and including the encryption of the first file. Once those metrics were recorded, they were stored in the Google Drive sync folder because of the nature of ransomware to encrypt the user's files. Google Drive has a version control feature built-in which allowed us to recover the previous version of the uploaded analysis data in the event that the ransomware encrypted the VTune™ analysis data and synchronized it to the cloud before the laptop was restarted. This was not a common issue considering I made sure that the Google Drive client showed that the data was synced and up-to-date before immediately restarting

the laptop. The VTune™ analysis data was then extracted from a different computer, converted to comma-separated value (CSV) format, and labeled.

I also collected 973 samples of known benign software programs which have commonly shown to trigger false positive alerts in other research. These programs include 7-Zip, WinRAR, AxCrypt, VeraCrypt, TrueCrypt, DiskCryptor, and GNU Privacy Guard. They were all run in several different modes to incorporate combinations of different encryption methods (AES, Blowfish, DES, etc.), different compression rates, and other built-in features native to the programs.

The CSV results for each sample — malicious and benign — were labeled only by the SHA-1 hash of the original ransomware binary and fed into another system running a 6-core i7 Extreme CPU overclocked to 3.9 GHz, 64 GB of memory, and 1 TB solid state drive. This system was used to conduct machine learning, and it was configured to use the following machine learning classifiers: k-Nearest Neighbor (kNN), Random Forests, and Support Vector Machine (SVM). The CSV results were shuffled and merged into three training sets:

1. The first training set was heavily populated with benign samples and sparsely populated with malicious samples.

2. The second training set contained a more closely balanced number of benign and malicious samples, but it still had a malicious-to-benign ratio of 1:2.

3. The third set was heavily populated with malicious samples and sparsely populated with benign samples.

The testing sets for each training set all contained a sparsely populated number of malicious samples as the number of malicious samples a typical classification engine encounters on a "real world" system would also be sparse. However, the testing sets did differ from the training sets in size and are shown in Table 6.1.

The number of malicious and benign samples in the training and testing sets are shown in Table 6.2 and Table 6.3 respectively.

Set	# in Training Set	# in Testing Set
1	32,450	2,000
2	450	34,000
3	17,225	17,225

Table 6.1: Number of training and testing samples per set

Set	# of Malicious	# of Benign
1	1,450	31,000
2	150	300
3	15,000	2,225

Table 6.2: Number of malicious and benign samples in training set

Set	# Malicious	# Benign
1	100	1,900
2	1,000	33,000
3	2,225	15,000

Table 6.3: Number of malicious and benign samples in testing set

Chapter 7

Results and Analysis

For detection purposes, I used a simple binary classification of true for malicious files and false for benign files. I merged and shuffled the samples and used a training set of 32,450 and a testing set of 2,000 (100 of which were malicious). The results of this first approach for k-Nearest Neighbor, Random Forests, and Support Vector Machine are shown in Table 7.1.

Classifier	TP	FP	TN	FN	Accuracy
kNN	91%	0.11%	99.89%	9%	99.45%
SVM	91%	0.05%	99.95%	9%	99.5%
RF	96%	0.26%	99.74%	4%	99.55%

Table 7.1: Set 1 Results

My second approach used a training set of 450 samples (150 malicious and 300 benign). I then used a testing set of approximately 34,000 (1,000 malicious and the rest benign). The results are shown in Table 7.2.

Classifier	TP	FP	TN	FN	Accuracy
kNN	96.9%	3.78%	96.22%	3.1%	96.24%
SVM	91.4%	0.04%	99.96%	8.6%	99.71%
RF	99.3%	1.53%	98.47%	0.7%	98.5%

Table 7.2: Set 2 Results

My third approach used a training set of 17,225 samples (15,000 malicious and 2,225 benign). I then used a testing set of identical size but with the malicious and benign numbers swapped. The results are shown in Table 7.3.

Classifier	TP	FP	TN	FN	Accuracy
kNN	99.91%	1.62%	98.38%	0.09%	99.35%
SVM	99.28%	0.36%	99.64%	1.08%	99.59%
RF	99.96%	0.73%	99.27%	0.04%	99.36%

Table 7.3: Set 3 Results

In the first set, the training set was heavily populated with benign samples and sparsely populated with malicious, and the testing set matched this ratio. The accuracy was over 99% with low false positive rate for all classifiers, but the false negative rate was 9% for both k-Nearest Neighbor and Support Vector Machine and 4% for Random Forest. With such high false negative rates, the model was falsely classifying malicious samples as benign.

In the second set, the training set was more balanced with a similar number of malicious and benign samples while the testing set was heavily populated with benign samples. For k-Nearest Neighbor, the false negative rate was reduced from 9% to 3.1%, the accuracy was reduced from 99.45% to 96.24%, and the false positive rate increased to 3.78%. There was negligible change between the first and second sets for Support Vector Machine. For Random Forest, the false negative rate decreased from 4% to 0.7%, the accuracy decreased from 99.55% to 98.5%, and the false positive rate increased to 1.53%.

In the third set, the training set was heavily populated with malicious samples and sparsely populated with benign. The testing set was made up of about 13% malicious samples. Both k-Nearest Neighbor and Random Forest achieved my goal for a true positive rate of 99.91% and 99.96% respectively, but they both fell slightly short of the goal for false positive rate and accuracy. On the

other hand, Support Vector Machine missed the true positive goal by 0.62% but exceeded the goals for both false positive and accuracy with rates of 0.36% and 99.59% respectively. It also showed the most improvement from the first two sets. Table 7.4 shows the comparison of the third set to the goals listed in my proposal.

Classifier	TP Rate	TP Goal	Diff
kNN	99.91%	99.9%	+0.01%
SVM	99.28%	99.9%	-0.62%
RF	99.96%	99.9%	+0.06%
	FP Rate	FP Goal	Diff
kNN	1.62%	0.5%	-1.12%
SVM	0.36%	0.5%	+0.14%
RF	0.73%	0.5%	-0.23%
	Acc Rate	Acc Goal	Diff
kNN	99.35%	99.5%	-0.15%
SVM	99.59%	99.5%	+0.09%
RF	99.36%	99.5%	-0.14%

Table 7.4: Set 3 Results vs. Goals

Chapter 8

Discussion and Future Work

The detection model that I have proposed in this paper is a proof-of-concept. My intent was to prove that ransomware can be detected before any data was lost to the user which was a plaguing problem for other models seen to date. While it still requires tuning and improvement, it has shown to be effective and that it can feasibly be built upon or integrated into other existing malware analysis or anti-virus solutions, and I will discuss these two approaches in this section.

8.1 This Model As A Foundation

This model could serve as a foundation upon to build a full anti-virus suite capable of using HPC behavior analytics to discover all other forms of malware known to exist, and it could be adapted to quickly incorporate future evolutions in the malware arena, but this approach doesn't come without a significant set of challenges. One of these challenges is the copious amounts of manual analysis required to retrieve data for the vast number malware samples to build a sufficient training set for other families of malware. This is a human-centric approach in a world quickly speeding towards automation. While some factors could be

automated, it would require the design and implementation of a new system just to automate these analyses assuming no modifications to the binaries were required.

Another challenge posed by this approach is the infancy of the model. It would need to garner significant acceptance as a viable solution by a cyber security corporation and/or research academics throughout the cyber security community. To do that, more work is required to improve the existing model and address certain limitations before it would be ready for such acceptance as discussed in the Limitations and Assumptions section below.

Despite these challenges, this model could potentially serve as a foundation for a new evolution of behavior-based anti-virus and security software, and I aim to improve this model so it can.

8.2 This Model As An Integrated Solution

In contrast to using my model as a foundation, this model could more easily be integrated as a software module into existing anti-virus software suites. Many anti-virus or endpoint security suites come with several "bolt-on" modules to address several aspects of security such as anti-virus detection and removal, disk and file encryption, local firewall, disaster recovery (backup and restore), and Internet proxy. This ransomware detection model could quickly be polished and bolted on to the framework as its own module. The development teams on the back-end would need to stay up-to-date with the latest ransomware samples as they already do for anti-virus signature detection, and they would just need to tune the HPC detection metrics in addition to their weekly or monthly updates.

This approach could also gain much faster acceptance as a viable solution if cyber security companies and research academics integrated it into existing

solutions with trusted foundations. Still, this would also require an amount of human-centric work but not to the extent of the foundational approach of this model.

8.3 Limitations and Assumptions

The main critical assumption posed by my research is that the modification of 529 different samples had little-to-no affect on my results. While it made up less than 10% of the overall number of samples, it is plausible that the modification of these files could have had a more positive impact on my results by making all samples more alike as opposed to having a more varying degree of differentiation between samples. While this may be concerning to some, my model intentionally focused on the required functions of ransomware: the call to a command and control server, the encryption key generation, the file system scan, the loading of crypto libraries, and the beginning of encryption.

However, it is also possible to automate the detection of certain functions in binaries such as time-based or event-based triggers, and analysis tools can be configured or written to begin its analysis after those triggers have been set off if a more automated model was to be conceived. It is also possible to look for the existence of the typical ransomware functionality spanned across a binary file before execution and immediately flag a file for review by the user before execution. It may even be possible to use these triggers as metrics to enhance detection capabilities. Due to these possibilities, I believe that similar results can be achieved without modification of the ransomware binaries, but achieving similar results would require significant improvements over the dynamic analysis portion of my model that were not implemented in this paper.

Chapter 9

Conclusion

I have proposed a set of ten requirements necessary to ensure accurate and acceptable levels of ransomware detection and built a model around those requirements. While the model I built does not achieve ten out of ten requirements, it does meet six out of ten as it currently stands with no assumptions.

1. It doesn't rely on signatures, but rather it relies on a range of CPU behavior patterns for detection.
2. It detected ransomware not incorporated into the training set indicating that other zero-day ransomware will be detected.
3. Every family of ransomware was included in every training and testing data set, and all families were detected.
4. A true positive detection rate of 99.9% was not achieved for Support Vector Machine, but the mark was missed by a very small margin of 0.62%.
5. A false positive detection rate of 0.5% or less was achieved for Support Vector Machine by 0.14%.
6. An model accuracy rate of 99.5% was exceeded by 0.09%.

7. Given that the HPCs were taken up to the point before the first file was encrypted or deleted, this detection model ensures no data loss.

8. This model does not yet employ defensive techniques as it has not been built into an actual deployment package yet.

9. This model does not employ user transparency, but the incorporation of this model into an existing anti-virus software suite would address this requirement.

10. As with user transparency, the termination of a malicious ransomware process would be achieved by introducing this model into an existing anti-virus software suite, but it currently does not meet this requirement.

Assuming the integration into existing anti-virus software suites as discussed for future work, my model will meet eight — potentially nine — out of ten requirements with the assumptions discussed for future work as it stands. Given that other machine learning research indicates the real possibility of malicious attacks on the data sets for continuous learning models to intentionally throw off the classifier's results, the potential to meet this requirement is a toss-up for future work. It is plausible that more tuning of the training and testing data sets could raise the true positive rate without sacrificing the false positive of accuracy attributes meeting a potential ten out of ten requirements, but — as it stands — I have proven that ransomware can accurately and effectively be detected resulting in no data loss to the user with the user of hardware performance counters.

References

[1] D. O'Brien, "Ransomware 2017," *Symantec Internet Security Threat Report*, 2017. 1, 2

[2] V. Total, "Virustotal-free online virus, malware and url scanner," *Online: https://www. virustotal. com/en*, 2018. 3

[3] A. Kharraz, S. Arshad, C. Mulliner, W. K. Robertson, and E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware.," in *USENIX Security Symposium*, pp. 757–772, 2016. 3, 5, 9, 13

[4] S. Morgan, "Global ransomware damage costs predicted to exceed $8 billion in 2018." `https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-exceed-8-billion-in-2018/`, 2018. 6

[5] I. Sherr, "Wannacry ransomware: everything you need to know." `https://www.cnet.com/news/wannacry-wannacrypt-uiwix-ransomware-everything-you-need-to-know/`, 2017. 6

[6] 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, pp. 599–611, ACM, 2017. 7, 9, 16

[7] P. Ducklin, "Mamba ransomware strikes at your whole disk, not just your files," *Naked Security, Sophos*, vol. 27, 2016. 7

[8] V. C. Craciun, A. Mogage, and E. Simion, "Trends in design of ransomware viruses.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 598, 2018. 8

[9] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and A. Chattopadhyay, "Rapper: Ransomware prevention via performance counters," *arXiv preprint arXiv:1802.03909*, 2018. 9, 14, 15

[10] A. Kharraz and E. Kirda, "Redemption: real-time protection against ransomware at end-hosts," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 98–119, Springer, 2017. 9, 14, 19

[11] M. B. Bahador, M. Abadi, and A. Tajoddin, "Hpcmalhunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*, pp. 703–708, IEEE, 2014. 9, 10

[12] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proceedings of the sixth ACM workshop on Scalable trusted computing*, pp. 71–76, ACM, 2011. 9, 11

[13] J. R. Bulpin and I. Pratt, "Hyper-threading aware process scheduling heuristics.," in *USENIX Annual Technical Conference, General Track*, pp. 399–402, 2005. 9

[14] G. Contreras and M. Martonosi, "Power prediction for intel xscale/spl reg/processors using performance monitoring unit events," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, pp. 221–226, IEEE, 2005. 9

[15] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control.," in *OSDI*, vol. 4, pp. 16–16, 2004. 9

[16] N. Herath and A. Fogh, "Cpu hardware performance counters for security. blackhat usa 2015 briefing.(2015)," 2015. 10

[17] S. Vogl and C. Eckert, "Using hardware performance events for instruction-level monitoring on the x86 architecture," in *Proceedings of the 2012 European workshop on system security EuroSec*, vol. 12, 2012. 10

[18] 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 *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 559–570, ACM, 2013. 10, 12

[19] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," *arXiv preprint arXiv:1609.03020*, 2016. 11

[20] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 109–129, Springer, 2014. 11

[21] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *Interna-*

tional Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 3–24, Springer, 2015. 12

[22] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pp. 457–468, ACM, 2018. 15

[23] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, "The cuckoo sandbox," 2018. 22

[24] M. Sikorski and A. Honig, *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012. 23

[25] 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*, pp. 303–312, IEEE, 2016.

[26] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. Van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 65–79, IEEE, 2012.

[27] C. Kolbitsch, E. Kirda, and C. Kruegel, "The power of procrastination: detection and mitigation of execution-stalling malicious code," in *Proceedings of the 18th ACM conference on Computer and communications security*, pp. 285–296, ACM, 2011.

[28] A. B. Kardile *et al.*, *Crypto Ransomware Analysis and Detection Using Process Monitor*. PhD thesis, 2017.

[29] A. Zimba, "Malware-free intrusion: a novel approach to ransomware infection vectors," *International Journal of Computer Science and Information Security*, vol. 15, no. 2, p. 317, 2017.