A New Ransomware Detection Scheme based on Tracking File Signature and File Entropy

by

Brijesh Jethva

B.Eng., Gujarat Technological University, 2014

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

SUPERVISORY COMMITTEE

A New Ransomware Detection Scheme based on Tracking File Signature and File Entropy

by

Brijesh Jethva

B.Eng., Gujarat Technological University, 2014

Supervisory Committee

Dr. Issa Traore, Department of Electrical and Computer Engineering
Supervisor

Dr. Mihai Sima, Department of Electrical and Computer Engineering

Departmental Member

ABSTRACT

Ransomware is a type of malware that hijack victims' computers, by encrypting or locking corresponding files, and demanding the payment of some ransom in cryptocurrency for the restoration of the files. The last few years have witnessed a sudden rise in ransomware attack incidents, causing significant amount of financial loss to individuals, institutions, and businesses. In reaction to that, ransomware detection has become an important topic for research in recent years. Currently, there are three types of ransomware detection techniques available in the wild: static, dynamic and hybrid. Unfortunately, the current static detection techniques can be easily evaded by code-obfuscation and encryption techniques. Furthermore, current dynamic and hybrid techniques face difficulties to detect novel ransomware.

In the current thesis, we present an upgraded dynamic ransomware detection model with two new sets of features: grouped registry key operation, and combined file entropy and file signature. We analyze the new feature model by exploring and comparing 3 different linear machine learning techniques: SVM, Logistic Regression and Random Forest. The proposed approach help achieves improved detection accuracy and provides the ability to detect novel ransomware. Furthermore, the proposed approach helps differentiate user-triggered encryption from ransomware-triggered encryption, which allows saving as many files as possible during an attack.

To conduct our study, we use a new public ransomware detection dataset collected at the ISOT lab, which consists of 666 ransomware and 103 benign binaries. Our experimental results show that our proposed approach achieves relatively high accuracy in detecting both previously seen and novel ransomware samples.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would first like to express my sincere gratitude to my supervisor, Dr. Issa Traore for his continuous support and motivation for me to pursue my studies and research at the University of Victoria. I am greatly appreciative to Dr. Issa Traore, who provided me an opportunity as his research student and provided ISOT laboratory environment for my work. It would not have been possible to conduct this research without his continuous encouragement and excellent mentorship. I am also thankful to thank Dr. Mihai Sima and Dr. Venkatesh Srinivasan, for serving on my supervisory committee.

I was lucky to be surrounded by amazing friends and colleagues throughout my journey of masters. Special thanks to the University of Victoria for providing me the beautiful campus, TA and co-op opportunities. I would also like to thank my first employer Infosys Ltd. for introducing me to the IT world and nurturing me as an IT professional.

Finally, I owe a deep sense of gratitude to my loving and supportive parents, my younger brother Vishal and my lovely wife Aayushi for always being there and providing me continuous encouragement throughout my years of study.

DEDICATION

To my Pillars of
Strength, Mom, Dad,
Vishal and Aayushi

Chapter 1 : Introduction

1.1 Context

In this modern era, as the use of digital devices is increasing day by day, the threats on these digital devices are also growing. There are many malicious programs, such as virus, worm, or spyware released in the wild, which can seriously harm digital systems. Among the current malicious software, ransomware appears to be one of the most disconcerting.

Over the last few years, there has been a significant growth in the number of ransomware attacks. Cyber Criminals are getting more innovative, and the damage is only getting worse. According to a study by Datto, a leading cybersecurity company [1], ransomware is responsible for more than US $75 billion extortion annually. The healthcare and financial service industries are the top targets of attackers. Over 50% of the participants in the study, believed their business was not ready to handle ransomware threat. Cryptolocker ransomware alone managed to infect approximately 250 thousand computers worldwide, including an entire police department that had to pay a ransom to decrypt their documents [2]. In 2017, NotPetya and Wannacry ransomware were wakeup calls to businesses all around the world. The Hollywood Presbyterian Medical Center, in February 2016, paid a ransom amount of 40 Bitcoins valued $17,000 at the time after being hit by a ransomware attack that crashed the hospital's entire network [3]. In May 2016, the University of Calgary paid US $16,129 after ransomware handicapped multiple systems [4].

The first ransomware ever used was PC CYBORG/AIDS. It was delivered using a floppy disk, and it mainly counts for the number of times the system reboots. When system reboot count reaches 90, it hides directories and encrypts all the file names in the system root directory[5]. Until a few years ago, ransomware incidents were not significant. However, with the evolution of robust

encryption techniques, ransomware started making headlines as the most notable malware, and as mentioned above, ransomware infections have costed users a considerable amount of time and money over the past several years.

There are two types of ransomware currently available: locker ransomware and crypto-ransomware. Locker ransomware locks the computer system to prevent the user from using it. Crypto-ransomware encrypts the user's files to make them inaccessible to victims. Very often crypto-ransomware does not encrypt the whole hard-disk but searches for specific extensions only. The user is threatened to pay a ransom by holding hostage her data or system. Users can regain access to their files only through anonymous payment mechanisms, such as cryptocurrencies.

1.2 Research Problem

Ransomware detection techniques fall under the same general categories of existing malware detection approaches. There are different approaches for malware analysis, including static analysis, reverse engineering, and dynamic analysis.

Malware detection based on static analysis is a well-known approach, which consists of analyzing the code of an application/software before deploying it in an operational environment. If the static study finds any malicious routines in the binary code, it will be detected by the Antivirus or firewall and prevented from running. The most common type of analysis is signature-based analysis where specific signatures (code patterns) are extracted from the application and compared against a repository of known malicious signatures. This repository needs continuous update over time as new malware is released. Signature-based detection can detect only known malware. This technique is not helpful to detect novel malware. To evade detection, malicious

software developers are continually changing malware code in such a way that each version appears different from the previous one.

Due to the limitations of signature-based malware detection, it is necessary to have better insight into malware's behavior, and leverage such understanding to improve detection. Reverse engineering is one of the ways to achieve an in-depth understanding of the internal mechanics of malware code. Reverse engineering of the malware involves disassembling or sometimes decompiling the corresponding binary code. Binary instructions are converted to code mnemonics through this process, allowing the analyst to establish a better understanding of how the program is executed and what system it impacts. However, due to the increasing complexity of malicious programs, there is a growing possibility or likelihood that disassemblers may fail sometimes, or the decompiler may produce obfuscated code. This process also can be very tedious and take a significant amount of time and resources. Reverse engineering for a large number of malware families is extremely time-consuming and resource intensive, with low success rates. Hence the focus is now shifting towards dynamic malware analysis.

Dynamic malware analysis consists of the live monitoring of processes to identify anomalous behaviors. This involves analyzing all requests to access specific files, processes, connection or services, including each low level instruction executed at the operating system level or any other programs that have been invoked.

Most of the work done, till now, on dynamic ransomware detection system focuses on training a machine learning model on a limited types of features (i.e, API calls, dlls, mutex, etc.) or on features specific to particular ransomware family or ransomware binary referred to as binary features. Also, Windows default API calls share a major portion of the features used to train a machine learning model, making models biased towards API calls. Instead of using default APIs,

malware authors can customize the encryption techniques and write their own programs to encrypt the files. Binary features are also not helpful to detect new variants of the ransomware as these may contain new set of processes. The detection of novel ransomware family remains an open challenge that has not received sufficient attention in the existing literature. Since novel ransomware is always designed with improvements to evade detection systems, further research is required to evaluate the effectiveness of classification approaches in identifying novel ransomware strains.

Furthermore, considering that a key characteristic of ransomware infection is encryption, it is necessary to detect ransomware infection in the system during early stages for minimum file loss.

The purpose of the research conducted in this thesis is to detect novel and previously unseen ransomwares and creating a forward looking system for monitoring the ransomware system activity. We make a step forward such vision by proposing, implementing and evaluating an approach that combines automatic detection and file backup on windows system. We introduce an upgraded behavioral based ransomware detection system by exploring different machine learning classifiers and introducing two new set of features: groupped registry key operations, and combined file entropy and file signature[1].

High entropy operations during ransomware attacks are helpful to detect anomalous behaviour. However, sometimes files are encrypted by users for legitimate security purpose. In this case, current detection models based on file entropy calculation generate false positives, identifying non-malicious operations as malicious.

[1] Not to be confused with malware signature.

While file entropy and registry key operations were considered in one way or another in the existing literature, there has not been a systematic focus on how to utilize these features to improve ransomware detection accuracy and novel ransomware detection. Our work tackles this challenge. Our preliminary assessment guided us to design a detection system based on a combined analysis of entropy of write operations, file signature and data collected from security reports.

1.3 Approach Outline

The main idea behind our approach is that ransomware behavior when executed on Windows platform exhibit properties that differ from legitimate software applications.

Our proposed approach involves studying different ransomware families execution reports and extracting a set of features from generated reports to correctly distinguish between ransomware and benign applications. We observed through exploratory study that ransomware target specific registry key areas during early stage execution. We observed also that ransomware execution involves continuous high entropy operations of unknown file extensions.

Based on these observations, we identified a set of features that potentially can help recognize the typical ransomware patterns. The extracted features are passed through feature selection methods to avoid overfitting, and then classified using machine learning techniques. We investigated in this thesis three different classification techniques, namely, logistic regression, support vector machine (SVM) and random forest. Experimental evaluation was conducted using a dataset consisting of a wide variety of ransmomware and benign application samples.

1.4 Thesis Contribution

The main contribution of this work is the design of a new a framework to detect a ransomware, with high degree of accuracy and mininum file loss, by introducing a set of new features and a consolidated machine learning model that classifies effectively ransomware and benign applications. The new features help achieves improved accuracy, and provide the ability to detect novel ransomware, and identify user-triggered and ransomware-triggered encryptions. This potentially can help protect as many files as possible against malicious ransomware-triggered encryption.

This is an important step towards detecting emerging malware, those that avoid static based detection by using obfuscated coding techniques. While previous ransomware detection models have been evaluated using ransomware samples drawn from a relatively small number of families, our evaluation relies on the newly collected ISOT dataset, which contains the broadest number of ransomware families available in a public dataset. Experimental evaluation on aforementioned collected ransomware dataset and benign applications, yields for regularized logistic regression, the best performing of all algorithms, a detection rate of 100%, an accuracy of 98.7% and a false positive rate of 1.41 %.

1.5 Thesis Outline

The outline of the thesis is as follows. Chapter 1 gives an outline of the context of the research, formulates the research problem, and summarizes the contribution made.

Chapter 2 provides background information on ransomware, and summarizes and discusses the related works.

Chapter 3 presents the ISOT dataset collection procedure, and describes the dataset.

Chapter 4 presents the proposed features obtained through exploratory analysis of sample ransomware binaries and legitimate applications in a sandbox environment.

Chapter 5 presents the experiments conducted to asess the impact of the derived features on performance and evaluate the accuracy of the proposed detection scheme.

Chapter 6 makes concluding remarks and discusses future work.

Chapter 2 : Background and Related Works

Understanding the behavior and execution characteristics of ransomware plays an important role in designing adequate detection system. In this chapter, we start with presenting a case study on ransomware and then provide an overview of related works done on ransomware detection.

2.1 Background on Ransomware

2.1.1 Ransomware Anatomy

Ransomware uses various social engineering tactics to make the victim afraid of falling for real-world consequences (i.e., owing a fine, facing arrest and prosecution), and the delivery or infection can be done through multiple attack vectors, such as exploit kits, malicious pdf files, phishing, and malicious advertisements campaigns [6]. Figure 2.1 illustrates a typical ransomware attack scenario.

In most of the cases, ransomware gets inside the system when the user clicks on the phishing email links. Once the user clicks on the malicious link, the malicious payload is downloaded in the backend and starts its execution. To hide its identity, the ransomware does not get executed as a standalone process. Instead, it uses a host file so-called dropper file, which helps it hide its identity. For example, it may use the Windows explorer process in front, but in the backend, it will create legitimate-looking fake svchost process. It also ensures that it keeps running on infected systems, persists across reboots and executes even if the system is started in “safe mode”. To become persistent across reboots, it makes registry key additions (in Windows) and also adds itself to the system startup processes.

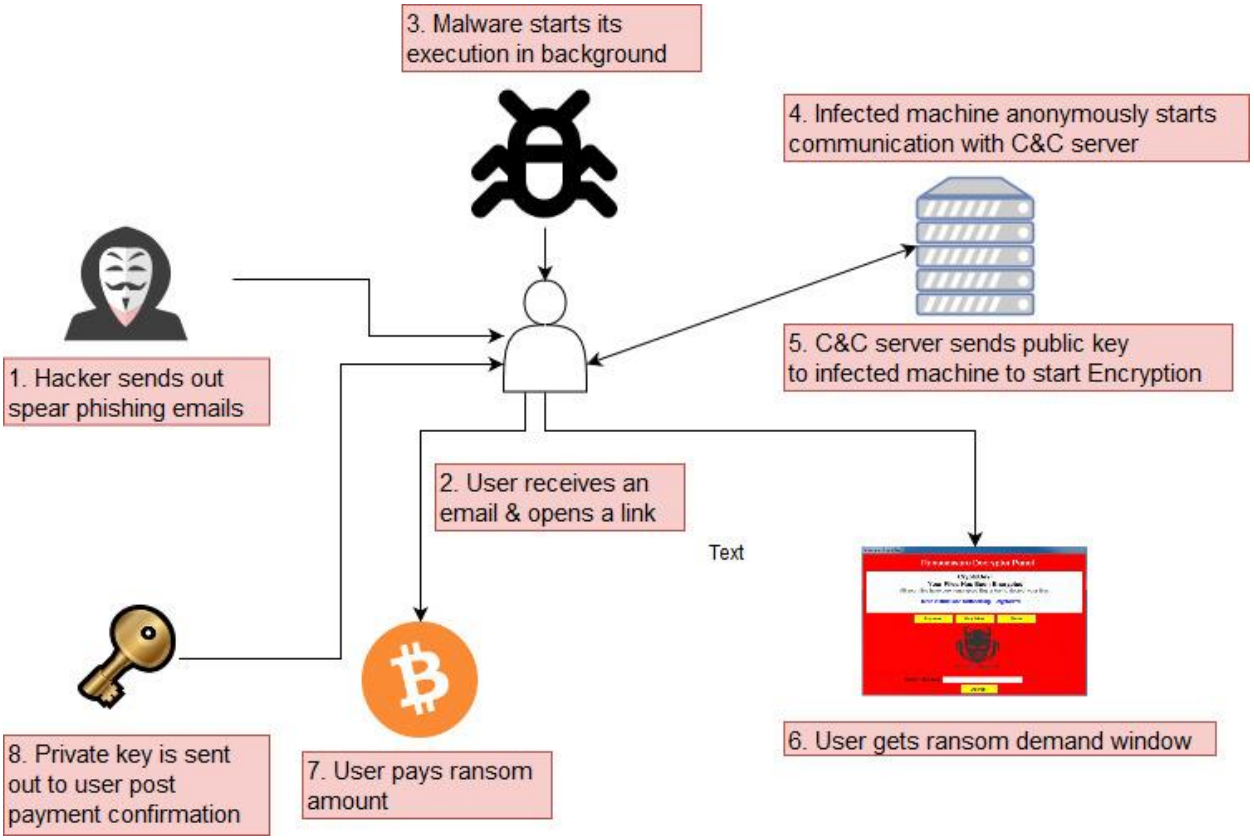

Figure 2.1 *Ransomware attack scenario*

After deploying itself on the victim's machine, the ransomware payload will then contact the command and control (C&C) server, which is operated remotely by the hacker. The C&C server validates the incoming request from the infected machine and generates a pair of keys, consisting of a public key and a private key. The public key is sent to the ransomware payload and used to encrypt the files on the infected machine[7]. Obviously, the files encrypted with the public key can only be decrypted by using the private key which is held in the C&C server. The communications between the C&C server and the infected machine is secured by the TOR browser. The ransomware creates a thread to download and install the TOR client to make communication anonymous. Services like security center, any antivirus program protecting the system, Windows error reporting tools and Windows updates are disabled one by one. The malware also deletes Windows shadow copy backup to make the system unrecoverable. As soon as the backend process

finishes encrypting all the desired files, it generates a persistent window on the user desktop that displays a ransom note, as shown in Figure 2.2. The ransom note informs the user that her machine has been attacked and can only be recovered by paying a ransom. Payment instructions are also provided, and usually, in these cases, a new and unique virtual currency address is created for each user to make transactions untraceable [2].

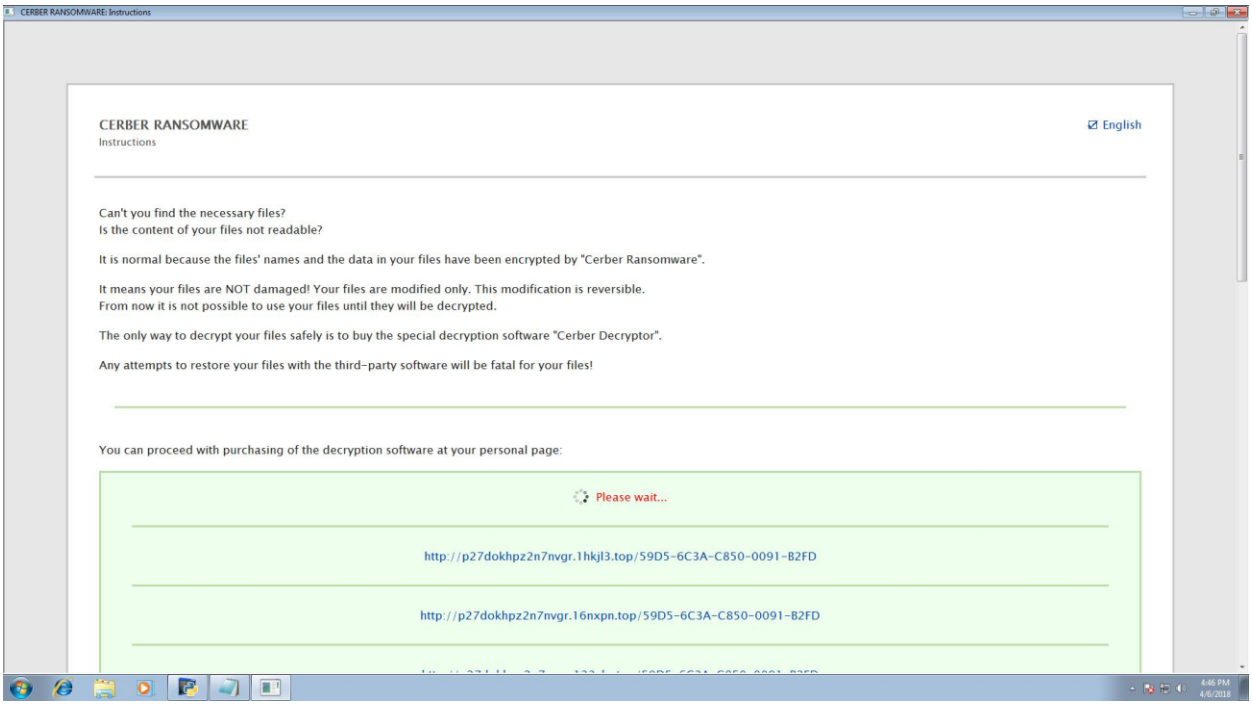

Figure 2.2 *Sample ransom note*

2.1.2 Execution Characteristics

Before engineering our feature model, we studied the cuckoo sandbox analysis reports of ransomware families, which carry a significant portion of our dataset. Despite being from different families, they share some common characteristics. Common characteristics could be helpful to find out the features to distinguish between ransomware and benign applications.

During our research, we noticed that malware authors use different techniques to deploy ransomware inside the system, such as strategic web compromise, drive-by download, phishing, vulnerability exploitation, browser exploit kits, etc. Sometimes ransomware is spread by exploiting common vulnerabilities in a LAN. Microsoft Word template files are also capable of embedding macros that can perform nefarious activities, such as downloading malware from remote sites, and executing commands in the backend [8].

We illustrate the typical characteristics of ransomware behavior by presenting a case study in the following.

At the beginning of the execution, the ransomware binary immediately copies itself to %AppData% or %LocalAppData% folder using random strings of lowercase characters (e.g., abshsdg.exe) [9]. Windows files can be recovered by built-in functionality provided by Windows called shadow copy. The ransomware detects Windows volume shadow copies in the system and deletes them to make user's data unrecoverable. Ransomware often uses below command to delete the windows shadow copies:

%WinDir%\system32\vssadmin delete shadows /all

The ransomware also adds some registry keys to Windows registry hives for persistency across the reboots. To ensure full destruction of the system files, the ransomware executes even if the system is restarted in a safe mode. An example of a registry key is as follows:

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run "<*random string*>":"<*full path to copied malware in %AppData%*>"

Ransomware generally achieves payload persistence by adding a registry key, a task scheduler, or by copying itself to an operating system startup process. Ransomware may also compromise the boot procedure of the operating system from loading itself.

Sophisticated ransomware will try to execute quietly to avoid being detected by Antivirus, for example, by injecting itself into a legitimate process and executing from %AppData% directory using standard Windows executable name.

Ransomware requires an Internet connection to download payload related files and to communicate with the command and control (C&C) server for encryption keys. Ransomware also uses the TOR anonymity network to host a payment server and facilitates untraceable ransom payment. Some ransomware utilizes Domain Generation Algorithms (DGA) that produce thousands of potential domain address per day in order to confuse defenders and escape detection. After connecting to the C&C server via HTTP, the public key exchange happens between the server and the infected machine. This communication is often SSL-encrypted. Hackers use private servers located at different ISPs often located in the eastern block countries (e.g., Russia). Sometimes, these C&C servers are hosted on legitimate infrastructure operated by third parties like Cloudflare [10].

The encryption process begins after a successful communication with the C&C server. This communication provides the public key, that is used throughout the encryption process. Most of the ransomware families use certified cryptosystems offered by Microsoft's CryptoAPI, such as the RSA and AES. To encrypt the data, these families use the RSA (CALG_RSA_KEYX) and AES (CALG_AES-256) algorithms. In this case, the ransomware calls Windows API (i.e., GetLogicalDrive()) functions to enumerate the storage on system drives. The ransomware recursively scans all the listed directories and processes each file for encryption. There are 3 classes

of ransomware according to how they process a file: class A, class B, and class C [11]. Class A ransomware opens the original file and immediately overwrites its content with encrypted data. Class B ransomware first moves the file to some random location, encrypts the file as in Class A and then moves the encrypted file back to its original location. Class C ransomware reads the original file, encrypts the content of the file, writes the encrypted content to a new file, and deletes the original file.

The encrypted data overwrites the original data in the file system, which reduces the chance of file recovery using forensics tools. As a form of bookkeeping, the list of encrypted files is stored in HTML files, text files or as registry keys. While going through each directory, ransomware generates the help files, which contains the ransom payment details.

Sometimes, specific file extensions are targeted. Generally, file formats from productivity suites like Microsoft Office, media files, Adobe Photoshop files, and son, are targeted. Some common file extensions targeted by ransomware are shown in table 2.1.

*.odt	*.ods	*.odp	*.odm	*.odb	*.doc	*.docx	*.docm
*.wps	*.xls	*.xlsx	*.xlsm	*.xlsb	*.xlk	*.ppt	*.pptx
*.pptm	*.mdb	*.accdb	*.pst	*.dwg	*.dxf	*.dxg	*.wpd
*.rtf	*.wb2	*.mdf	*.dbf	*.psd	*.pdd	*.eps	*.ai
*.indd	*.cdr	????????.jpg	????????.jpe	img_*.jpg	*.dng	*.3fr	*.arw
*.srf	*.sr2	*.bay	*.crw	*.cr2	*.dcr	*.kdc	*.erf
*.mef	*.mrw	*.nef	*.nrw	*.orf	*.raf	*.raw	*.rwl
*.rw2	*.r3d	*.ptx	*.pef	*.srw	*.x3f	*.der	*.cer

*.crt	*.pem	*.pfx	*.p12	*.p7b	*.p7c	*.pdf	*.odc

Table 2.1 *file extensions targeted by ransomware* [9]

Virtual currency is now a defacto method for ransomware payment. Most of the ransomware families demand a ransom of around 1.5 Bitcoins. Sometimes, they also prefer prepaid cards, or they also prefer other methods like PaySafeCard or Ukash [9]. All transactions are public by design. Sometimes they also include how to purchase Bitcoins from the exchange. A unique and new bitcoin address is assigned for each user so that it can be used as a reference to the victim and to receive payment as well. Furthermore, ransom notes also include ransom addresses. To notify the hackers sometimes, they have to send out the hash of the ransom payment.

2.2 Related Work on Ransomware Detection

Ransomware detection techniques are divided into three categories, which include static, dynamic, and hybrid. A significant amount of literature have been published on both dynamic and static approaches. We review and discuss, in the following, work done on all three ransomware detection techniques with a primary focus on machine learning-based approaches.

2.2.1 Machine Learning Approaches with Static Analysis

Schultz and colleagues [12] conducted one of the earlier works on using machine learning to detect malware. The authors used Portable Executable (PE), strings information, and byte sequences of binary to classify the malware using Naïve Bayes classification algorithm. Kolter et al. [13], in their work, proposed a similar approach to classify malware binaries using n-gram byte sequence with different classification algorithms, including naïve Bayes, decision trees, SVM and boosting. Boosted decision tree algorithm achieved the best performance with a true-positive rate (TPR) of 98% and a false positive rate (FPR) of 5%. Jerome and colleagues [14] extracted program

opcodes from malware binaries and translated them into a sequence of opcodes. The study published some interesting signature patterns of malware that helped improve the false positive rate and a false negative rate of the classifier. The authors used information gain (IG) to select valuable features and applied the SVM algorithm for classification. Experimental evaluation yielded a true positive rate of 81.40% and false positive rate of 2.67%.

Often, classification systems relying only on static detection cannot detect new variants of malware. Likewise, in a study conducted by Kharraz on ransomware detection, the average detection rate for new ransomware using static analysis was significantly low; only ten engines out of sixty tested could detect ransomware [2]. Moreover, static detection systems can be evaded using code obfuscation techniques. Moser et al. [15] explored this limitation of static malware detection and observed that advanced static-based detection could easily be evaded. In another similar work, Baig et al. [16] evaded static detection by modifying packed portable executables.

To overcome the drawbacks of classic signature-based detection systems, researchers have published several proposals on dynamic ransomware detection techniques. We review and discuss a sample of closely related work in the following.

2.2.2 Machine Learning Approaches with Dynamic Analysis

In 2015, Kharraz and colleagues [2] studied ransomware attacks that occurred in the wild from 2006 to 2014. The study explored 15 different ransomware families and showed that almost 94% ransomware samples implement simple locking or encrypting techniques. The authors suggested that by closely monitoring file system activity and the types of I/O request packets to the file system, it is possible to detect ransomware attacks. They also observed that Bitcoin addresses used to collect ransom payment from victims share similar transaction records, such as

a small number of transactions, small Bitcoin amounts, short activity period, etc. However, despite proposing possible strategies for ransomware detection, no concrete experimental evaluation was conducted by the authors.

In the follow-up work presented by Kharraz et al.[17], a ransomware detection system called UNVEIL was proposed. UNVEIL looks at the filesystem layer to spot the typical ransomware behavior. It uses text analysis techniques to detect ransomware threatening notes and continuously takes screenshots of the desktop to check for screen lockers. It also uses statistical analysis based on memory usage, processor usage, and disk I/O rates to detect abnormal behavior for ransomware variants. The experimental evaluation yielded 96.3% accuracy in detecting ransomware. Despite achieving relatively high accuracy, the model does not have early detection capability for ransomware attacks nor does it provide any backup mechanism. Also, the proposed system is inherently reactive and ineffective for newer ransomware samples.

On the other hand, ShieldFS, a competitor to UNVEIL developed by Continella et al. [18], is a self-healing ransommare-aware detection system with the additional capability of allowing the system to roll back malicious changes. It internally monitors low-level filesystem activities by computing the entropy of write operations, and the frequency of read, write, and folder listing operations. It also searches the memory regions of any process considered as "potentially malicious", by looking specifically for block cipher key schedules. The system combines both automatic detection and transparent file recovery in a ready to use driver. However, this methodology also has some limitations as new variants of ransomware tend to encrypt or delete the Windows shadow copy of the file system, making the chances of file recovery almost zero. Additionally, the system is more focused on file operation related features only. The memory

scanning aspect is time consuming and is plagued by the fact that there are rare chances to find a key in memory region.

CryptoDrop [11] was an early warning detection system to alert users during suspicious file activities. The system mainly focused on monitoring user data for changes. The authors divided ransomware into three major classes: class A, class B, and class C based on how they encrypt the user files. They used similarity functions to measure the dissimilarity between the original and the encrypted contents of each file. CryptoDrop was unable to determine the purpose of the changes in its audit. For example, it was not able to differentiate between the user-triggered encryption and ransomware triggered-encryption.

Daniele and colleagues presented a machine learning approach called EldeRan [19] for analyzing and detecting ransomware. In the first phase, EldeRan monitors a set of activities performed by applications and checks for attributes of ransomware. In the second phase, features like API calls, dropped files, registry keys, and directory enumerations are fed to a regularized machine learning model to learn patterns to differentiate between ransomware and benign applications. The experimental evaluation was based on a dataset involving 582 ransomware from 11 different families. An accuracy of 96.3% was obtained using dynamic analysis with a limited number of features. EldeRan was not able to extract the features when ransomware was silent for some time. Additionally, most of the features used in this system were binary. The authors focused only on the absence or presence of some of the features like registry key operations, dll operations, mutex, etc. However, in the new variants of malware, the absence of these particular operations makes the detection model ineffective. For example, a registry key operation used in one variant of ransomware might not be used by other variants or new versions of ransomware.

Chen et al. [20] proposed an approach for ransomware detection based on dynamic API calls flow graph by monitoring API call sequences of malware binaries and converting them to a set of features. They used different data mining algorithms including random forest, SVM, Naive byes and logistic regression. The logistic regression achieved the highest accuracy of 98.2% with the lowest false positive rate of 1.2%. However, the focus was only on a single feature to detect ransomware and the evaluation was based on a dataset consisting of only 168 ransomware samples.

Lanzi et al. [21] collected a large number of system calls from regular users on actual inputs and studied the diversity of system and API calls. They observed that the interactions of benign programs with the operating system are different from those of malicious programs. Kumar et al. [22] leveraged the dominance of API invocations to build a multi-layer perceptron (MLP), neural network model. Experimental evaluation of the proposed model on a dataset consisting of 7 different ransomware families yielded an accuracy of 98%.

Poudyal et al. [23] developed a reverse engineering framework for malware detection. The authors conducted a multi-level analysis of assembly codes, libraries and function calls, and applied different supervised machine learning techniques, including Bayesian Network, Random Forest, Smo and J48. The experimental evaluation yielded a detection accuracy of ransomware samples ranging from 76% to 97% based on the machine learning techniques used.

Recently, several works have been published on ransomware detection for mobile phones and the Internet of Things (IoT) as well. Karimi and Moattar [24] presented an approach that transforms a sequence of executables into a grey scale image. Then, they used Linear Discrimant Analysis (LDA) statistical method to separate two or more classes with dimension reduction functionality to improve the performance of the model. The evaluation of the proposed model was

conducted through two different experiments. The first experiment was conducted using a dataset consisting of 140 ransomware samples from two well-known families and 20 benign samples, yielding 97% accuracy. In the second experiment, the model achieved an accuracy of 97.3% with a dataset consisting of 230 ransomware samples from Locker and Koler families and 30 benign samples.

Andronio et al. [25] studied mobile ransomware families on Android devices, and introduced an approach, named HelDroid, to discriminate known and unknown ransomware samples from benign applications. HellDroid tracks and detects ransomware behavior at the application layer and uses Natural Language Processing (NLP) to recognize threatening phrases. The evaluation of the system achieved accuracy over 97% with a dataset consisting of 650 ransomware and about 81,000 benign samples. However, detection of threatening pharses is not much useful as by the time the user gets a ransom note on the screen the data is already encrypted.

2.2.3 Machine Learning Approaches with Hybrid Analysis

Hasan and Rahman [26] proposed a framework called "RansHunt" that combines static and dynamic analysis to detect ransomware. The proposed model was evaluated using a total of 1,283 different binaries which included 360 ransomware binaries of 21 different families and 923 benign binaries, achieving 97.1% accuracy. The authors introduced new network related features in the dynamic analysis, which did not contribute much to improve the detection rate. Also, the features used for the dynamic analysis were almost similar to EldeRan's features. The model is ineffective for new variants of ransomware.

Kashif and Riberio [28] presented a layered defense system for protection against crypto ransomware attacks. They combined both static and dynamic analysis to make a hybrid system.

The dynamic detection layer monitors the file system operations and entropy modifications related to massive encryption activities. Files modified by suspicious processes are backed up in other secure folder to preserve the data until the processes are classified as ransomware or benign.The proposed model was evaluated using a dataset consisting of 574 ransomware samples from 12 different ransomware families. The evaluation yielded an accuracy of 98.25%. However, like in other systems, dynamic analysis is highly dependent on API calls and file system operations. Ransomware binaries which use custom functions instead of default windows APIs are hard to detect with this system.

2.3 Summary

In this chapter, we provided background knowledge on ransomware, and then summarized and discussed related work on ransomware detection. Most of the covered papers discuss feature extraction techniques and machine learning models that could be applied to distinguish benign and ransomware behaviors correctly.

It is clear from the reviewed research that classification using static analysis is not enough to classify the ransomware effectively. Furthermore, behavioral based ransomware detection system is more effective than static based system for the detection of novel ransomware. From the above literature analysis, we can also note that most of the work focuses on a limited number of features like API calls monitoring and file operations. As a result, ransomware which do not use default Windows APIs are hard to detect with the existing models. Also, existing models are incapable to distinguish ransomware triggered encryption from user-triggered encryption.

While registry-key operations and file entropy were considered in one way or another in the existing literature, there has not been any systematic focus on how to utilize them in

combination to detect ransomware. We introduce a machine learning-based approach for automated ransomware detection with two new sets of features: groupped registry key operations and combined file-signature and file-entropy. The benefit of using the aforementioned features is three-fold: improved accuracy, improved novel ransomware detection rate, and helping identify user triggered and ransomware triggered encryption.

Chapter 3 : Dataset

Data is the foundation for any machine learning model. Hence, building the right dataset plays a pivotal role in constrcting and training a model. To our knowledge, there is no publicly available ransomware detection dataset. To fill this gap, the ISOT lab at the University has collected a new ransomware dataset to be shared freely with the research community. We use in the current thesis the aforementioned dataset to evaluate the proposed ransomware detection model. In this chapter, we start by describing the data collection environment, and then give an overview of the collected data.

3.1 Set up for Experiment

It is essential to understand ransomware behavior once it is deployed on a machine, the type of changes it incurs in the system and the goals of the breach. To get a detailed understanding of each variant, we executed all ransomware binaries, following established and commonly agreed guidelines, inside an open source automated malware analysis software called cuckoo sandbox. Figure 3.1 describes the data collection environment. We created a virtual machine environment running Windows 7 Professional with all necessary software (i.e., Python, Java, Microsoft Office, etc.) and provided controlled access to the Internet via NAT. The outbound network traffic to other machines in the local network was restricted to protect other machines from a ransomware infection. We also placed some personal user files such as pdf, jpeg, doc, and so on, under different directories of the Windows machine to monitor changes post ransomware infection. The Windows firewall and other security features in the machine were disabled to observe more ransomware behavior. We also made sure while executing the ransomware binary that no additional process was running apart from regular Windows processes.

We then executed each sample in the analysis environment for 30-45 minutes to capture the execution traces of the ransomware samples. We did a couple of full executions of ransomware binaries and concluded that 30-45 minutes threshold was sufficient for most of the ransomware samples. After each run, the Operating System (OS) was rolled back to a clean state to remove the influence of the previous infection.

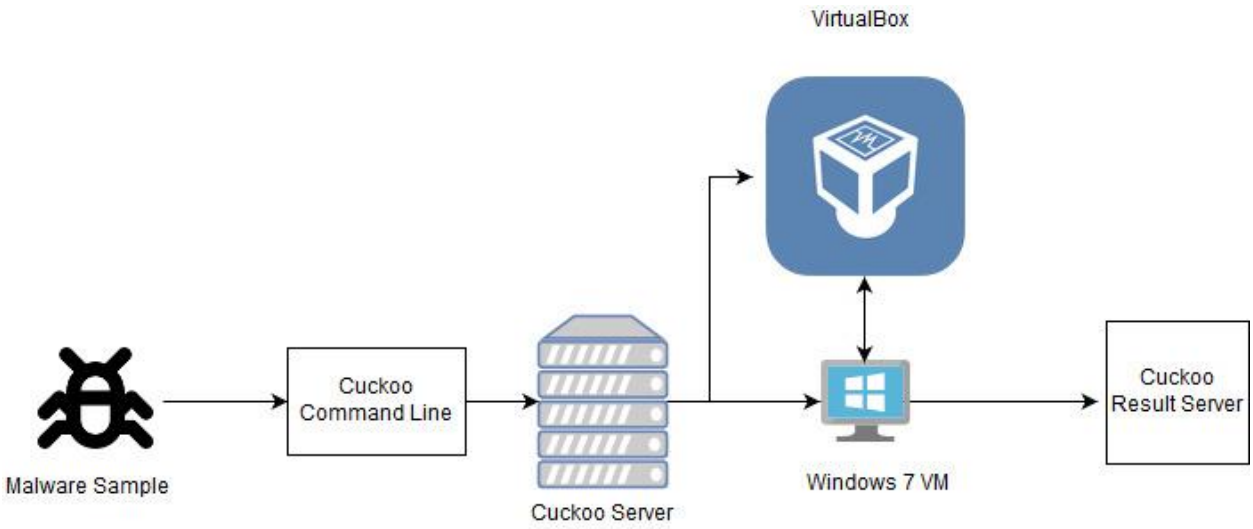

Figure 3.1 *Setup for experiment*

Cuckoo sandbox has two significant appliances: the host machine, where cuckoo is installed and the guest machines where the user can install one or more virtual operating systems for analysis. With the help of the host machine (in our case a machine running Ubuntu) and python command line utility available in the cuckoo sandbox, the user can execute any suspicious file in the guest machine. The cuckoo result server continuously runs during the suspicious file execution and reports all changes done inside the guest machine through a number of report files. Multiple report files are collected for each binary. The output data is stored in different files and directories. Figure 3.2 shows an example of an output directory structure of a sandbox analysis.

```
.
|-- analysis.log
|-- binary
|-- dump.pcap
|-- memory.dmp
|-- files
|   |-- 1234567890_dropped.exe
|-- logs
|   |-- 1232.bson
|   |-- 1540.bson
|   `-- 1118.bson
|-- reports
|   |-- report.html
|   |-- report.json
`-- shots
    |-- 0001.jpg
    |-- 0002.jpg
    |-- 0003.jpg
    `-- 0004.jpg
```

Figure 3.2 *Cuckoo analysis directory structure* [27]

The different types of generated report files and their contents are described as follows [27]:

dump.pcap

The network traffic generated during the execution of sample binary in the analysis virtual machine is stored in this file.

memory.dmp

This file contains a full memory dump of the virtual machine.

Files.json

For each dropped file, a JSON-encoded entry is stored in this file. Each entry in this file contains meta-data information about all processes that touched the file, the file path of the original file in the analysis machine, etc.

Logs

All raw logs generated by the cuckoo result server are stored in this directory in the form in files with extensions .bson.

Reports

The reports generated from the analysis machine are stored in this directory. The analysis report file report.json contains the following information about the behavior of the binary:

- Information about the analysis task, details about virtual analysis machine, duration of execution, etc.
- Information about different memory regions
- A checksum of the executed binary
- Network connections established during execution
- Different malware behavior signatures
- Imported Windows functions and libraries
- Dropped files during the execution
- Information about the different types of operations on the filesystem
- Information about system calls, arguments passed and returns values of the calls
- Information about different types of Windows registry operations
- Strings extracted from the binary file of the analyzed sample.

Figure 3.3 shows a sample report generated from the analysis.

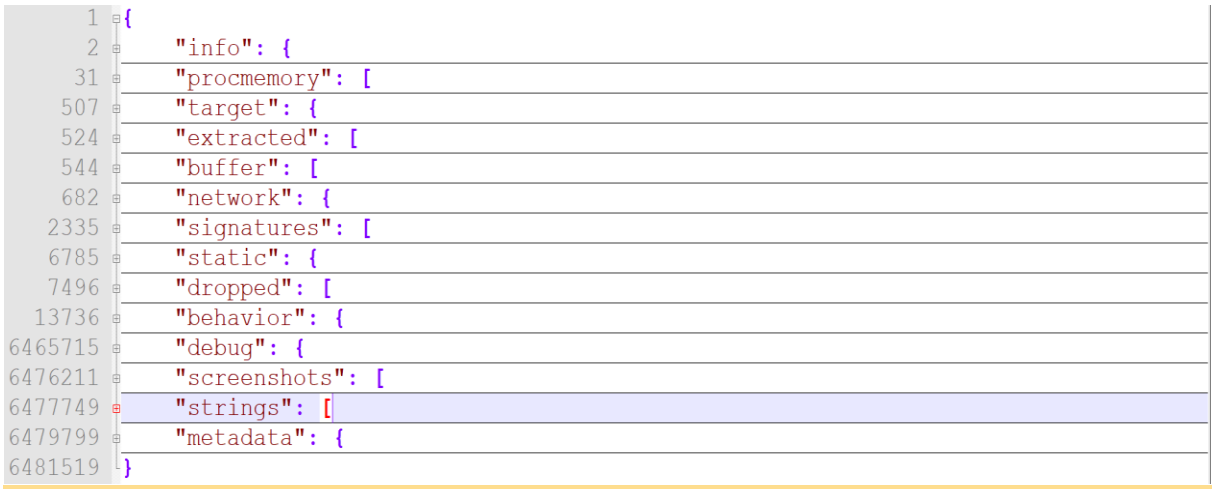

Figure 3.3 *Sample JSON report*

3.2 Data collection

To ensure the quality of the data, we collected ransomware samples from a well-known antivirus aggregator called VirusTotal. Samples were pre-classified in different ransomware families. We built a dataset of 103 benign samples and 666 ransomware samples from 20 different ransomware families. The collected ransomware samples represent the most popular ransomware versions and variants currently encountered in the wild. We only downloaded benign applications from trustworthy websites to ensure they did not contain suspicious components inside them. The benign dataset includes generic file utilities for Windows like file zippers, password managers, games, multimedia tools, developer tools, databases, etc. Table 3.1 provides a breakdown of the number of samples in each ransomware family.

Family	Number of Samples
CTBLocker	2
Cerber	122
CryptoShield	4
Crysis	8
Flawed	1
GlobeImposter	4
Jaff	3
Locky	129
Mole	4
Petya	2
Sage	5
Satan	2
Spora	5
Striked	1
TeslaCrypt	348
Unlock26	3
WannaCry	1
Win32.Blocker	18
Xorist	2
zeta	2
Total	666

Table 3.1 *Number of ransomware samples per family in the ISOT dataset*

3.3 Summary

In this chapter, we discussed the process we followed for sandbox execution of ransomware and benign binaries. We provided the final breakdown of ransomware variants by family. We also provided a brief overview of generated report files from cuckoo sandbox execution. The total size of the data collected at ISOT lab is around 429 GB. Our dataset includes screenshots of the desktop, memory dumps, network communication logs (.pcap files), behavior reports of ransomware named as "report.json" files, etc. In the next chapter, we discuss different behavioral characteristics of ransomware and present the feature model used to build our ransomware detection system.

Chapter 4 : Features Model

We have seen from our study of different ransomware families that ransomware, during execution, must follow specific patterns of behavior. As depicted in Figure 4.1, these patterns generally involve the following phases: Deployment, Installation of binary on the system, Connection with C&C server, File Encryption and Extortion. We identified a specific set of features from the behavior analysis of ransomware and previous works to distinguish ransomware from benign applications. In this section, we introduce the proposed feature model.

Figure 4.1 *Ransomware behavior pattern*

4.1 API calls

Windows operating system provides a set of programming interfaces that simplify the process of developing software; usage of Windows API makes developers free to focus on the logic of the program. From the previous work done on ransomware detection using dynamic analysis, we observed that most ransomware variants use standard Windows cryptographic APIs to encrypt the files. Therefore, the study of Windows API calls plays a vital role in the behavioral analysis of ransomware. When the system is under ransomware attack, significant changes in a file system

activity happen during a short period. (e.g., multiple file encryptions, or deletion requests). The best way to access or modify the files on Windows operating system is through the Windows API. For example, when the system call “FileOpen” is made, the operating system executes a series of instructions in the following order [28]. First, it will locate the file, check for the access permissions of the file and, give a handle back to the calling function. The ransomware can overwrite the file with the encrypted version or use secure deletion of the file using Windows Secure Deletion API. The ransomware begins the process of encryption itself by using the API function GetLogicalDrives() to enumerate the drives on the system and finishes its job by calling CreateDesktop() Windows API to create a persistent ransom note.

To study the importance of API calls in ransomware detection and how the requests generated by ransomware in windows operating system are different from those generated by benign applications, we extracted the frequency of each of the API calls initiated by ransomware and benign application during their execution in the sandbox. We identified 286 API calls of interest from our dataset, including both benign and ransomware.

The analysis of API calls revealed that API calls related to file system activities are heavily used in ransomware files compared to benign files, and certain API calls are only present in ransomware files. On further examination of calls, some API calls are present in both benign and ransomware files, but their use frequency varies in both benign and ransomware applications. The comparison of API call frequency in benign and ransomware applications is depicted in Figure 4.2. We also observed that not all ransomware families use the same API calls to achieve their goal. Table 4.1 shows the distribution of the top API calls by ransomware family.

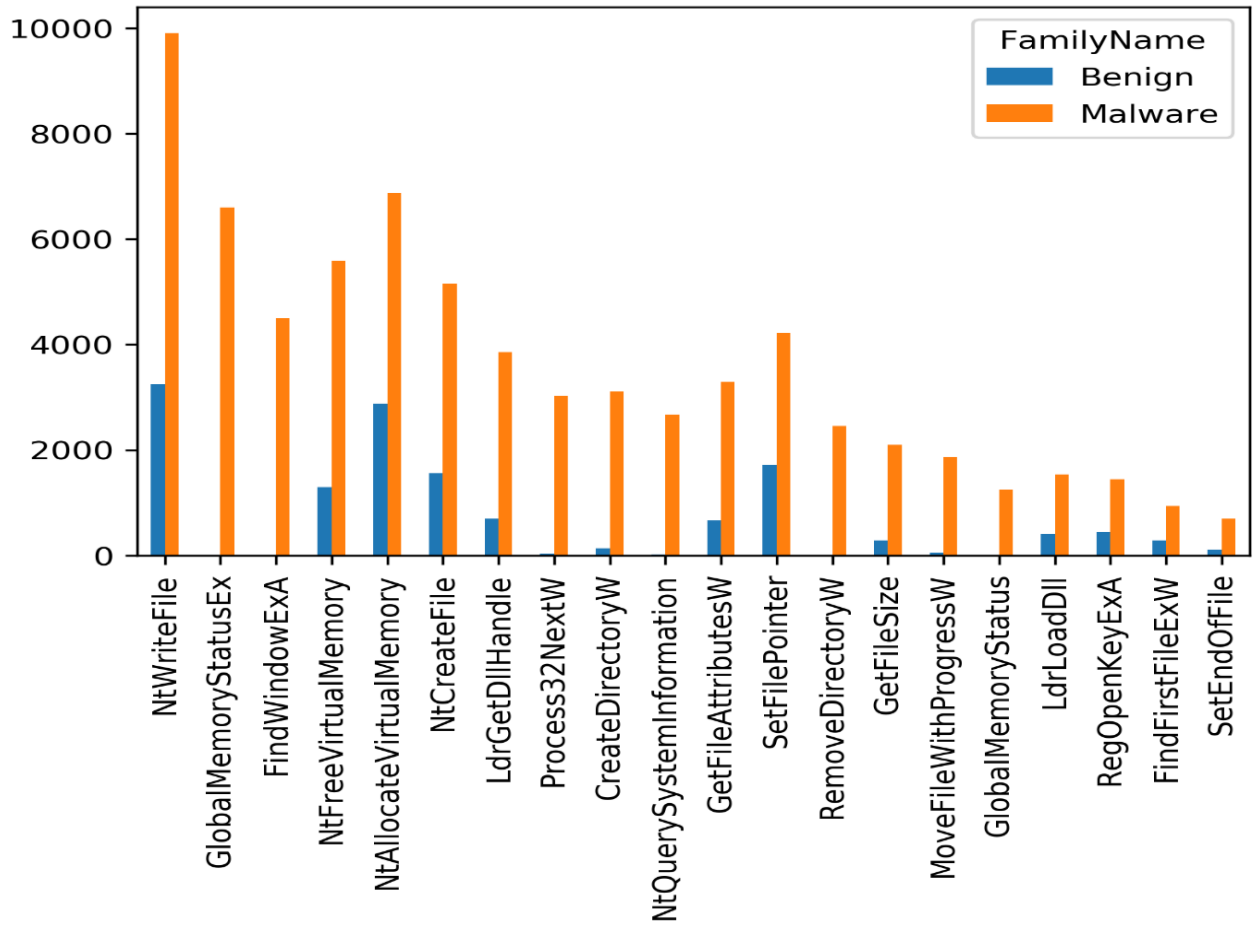

Figure 4.2 *API call frequency comparison*

Ransomware Family / Windows Api Calls	CTBLocker	Cerber	CryptoMix	CryptoShield	Crysis	Flawed	GlobeImposter	Jaff	Locky	Mole	Petya	Sage	Satan	Spora	Striked	TeslaCrypt	Unlock26	WannaCry	Win32.Blocker	Xorist	zeta
MoveFileWithProgressW		*			*				*				*			*		*			

FindResourceExW	*	*							*	*		*	*	*		*	*	*			*
CreateDirectoryW	*	*							*	*	*	*	*	*	*	*	*	*	*		
RemoveDirectoryW																*					
LoadResource	*	*			*	*			*	*	*	*	*	*		*	*	*			*
GetSystemWindowsDirectoryW	*	*			*	*	*		*	*		*	*	*		*	*	*	*		
RegQueryValueExW	*	*			*	*	*		*	*	*	*	*	*	*	*	*	*	*		*
SizeofResource	*	*			*				*		*	*		*		*	*	*			*
NtWriteFile	*	*			*				*	*	*	*	*	*	*	*	*	*	*	*	
FindWindowExA	*	*																			
NtCreateFile	*	*			*	*	*		*	*	*	*	*	*	*	*	*	*	*	*	
GetFileAttributesW	*	*			*		*		*	*	*	*	*	*	*	*	*	*	*	*	
GetFileSize	*	*							*			*	*	*		*	*	*		*	
RegOpenKeyExA	*	*							*	*	*	*	*	*		*	*	*	*		*

Table 4.1 *Distribution of API calls per Ransomware family*

4.2 File Entropy and File Signature

4.2.1 File entropy

Entropy in digital systems is a measure of randomness in a file. The concept of entropy first originated in the study of thermodynamics, but later, Claude E. Shannon applied this concept in digital communication in his work "A Mathematical Theory of Communication" [29]. A file is compressed by replacing large patterns of bits with shorter patterns of the bits. Compressed and encrypted files have a high degree of randomness. Shannon provided a formula to calculate the theoretical maximum amount for digital file compression. As per Shannon, the maximum entropy occurs when all bytes are distributed equally across the file. The entropy value is a calculation of the predictability of the next character in the file based on previous characters. It is measured in the scale of 1 to 8 where encrypted and compressed files have a high value, and standard text files have a low value. The Shannon entropy formula allows calculating the average minimum number of bits required to encode the string of symbols based on the frequency of symbols and the alphabet size. Shannon Entropy H is given by the below formula,

$$H = -\sum_{i} p_i \log_2 p_i$$

Where p_i is the probability of character *i* appearing in the alphabet stream.

To calculate the entropy of a file, we calculate the frequency of all ASCII characters, which include standard ASCII characters (0-127) and extended ASCII characters (128-255), in a given file, and then use this as the probability in the Shannon entropy formula.

4.2.2 File Signature

Some legitimate files, such as MS Office, 7-zip, pdf files are also highly compressed and have a high entropy value. Therefore, file entropy calculation alone does not help to differentiate between user-triggered encryption, and ransomware-triggered encryption. However, most of the file types have a file header and/or file footer, also called file signature or magic numbers, through which the actual format of the file can be identified[30]. For instance, JPEG image files begin with "FF D8" and end with "FF D9".

File signatures or magic numbers are the first few bytes in a file that are different for each file type. These bytes are used by the operating system to recognize the files without depending on the file extension. A file signature is not visible to users, but by using a hex editor, it can be seen. Changing or corrupting these bytes makes a file useless as they are essential for a file to be opened. Table 4.2 outlines some commonly used file types and their file signatures.

Extension	Signature	Description
PDF	25 50 44 46	PDF file
DOCX	50 4B 03 04	MS Office Open XML Format Document
7Z	37 7A BC AF 27 1C	7-zip compressed file
RAR	52 61 72 21 1A 07 00	WinRAR compressed archive
TAR	75 73 74 61 72	Tape Archive

Table 4.2 *File types and signatures*

4.2.3 Combined File Entropy and Signature

There are two strong characteristics of ransomware as follows:

1. Ransomware usually encrypts the whole file, which means it also clobbers the file signature of the data files.

2. Ransomware generally applies a decent encryption algorithm to the files. As a result of that, the file entropy will be very high.

Therefore, features derived from the combination of file signature and file entropy can effectively help identify ransomware-triggered encryption. To study the impact of ransomware infection on file entropy and file signature, we deployed some user files in our sandbox environment. Post successful execution of ransomware and benign binaries in a sandbox environment, we analyzed a total number of 157,187 user files from infected and normal Windows machines. The dataset was a combination of regular user files (i.e., *.docx, *.pdf, *.jpeg, etc.) and ransomware-encrypted files. Most of the user files post ransomware execution were encrypted. On the other hand, after benign binary execution, the files were unmodified. We also noticed that most of the ransomware encrypted files were missing file signatures. We then calculated the Shannon entropy of all files and filtered out the files where the file signatures were missing. The process flow chart is shown in Figure 4.3.

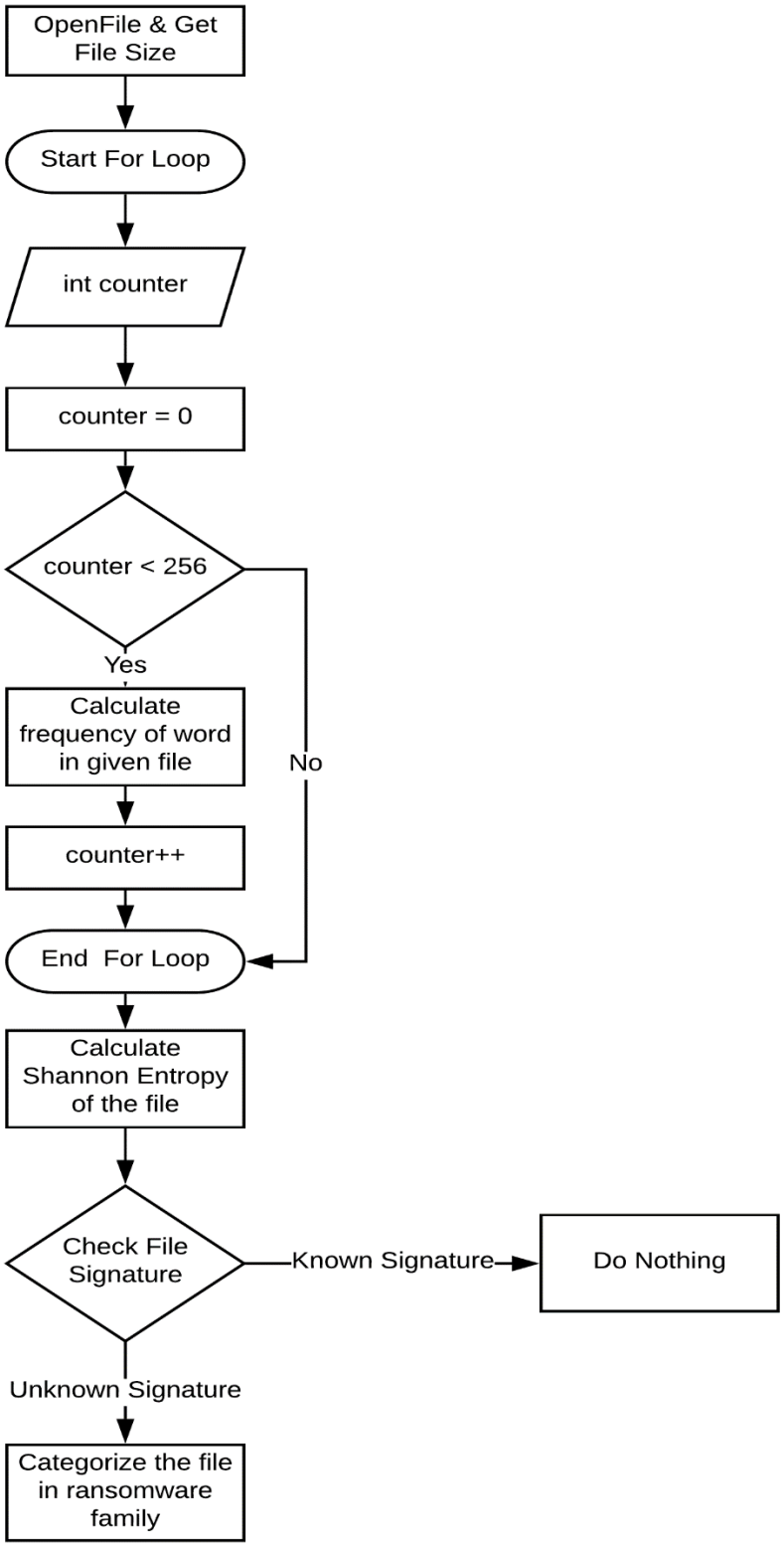

Figure 4.3 *File Entropy Calculation Process Flowchart*

We grouped the filtered files by ransomware families and calculated the average entropy of files per family. On one hand, for all ransomware families the average entropy values were above 7. On the other hand, after executing the benign binaries, because the files remain unchanged, the average entropy of the files was around 4.5. Figure 4.4 shows the average entropy for the different ransomware families. As we can see from the figure 4.4, filtered files of ransomware infected machines have very high average entropy compared to the files in uninfected machines.

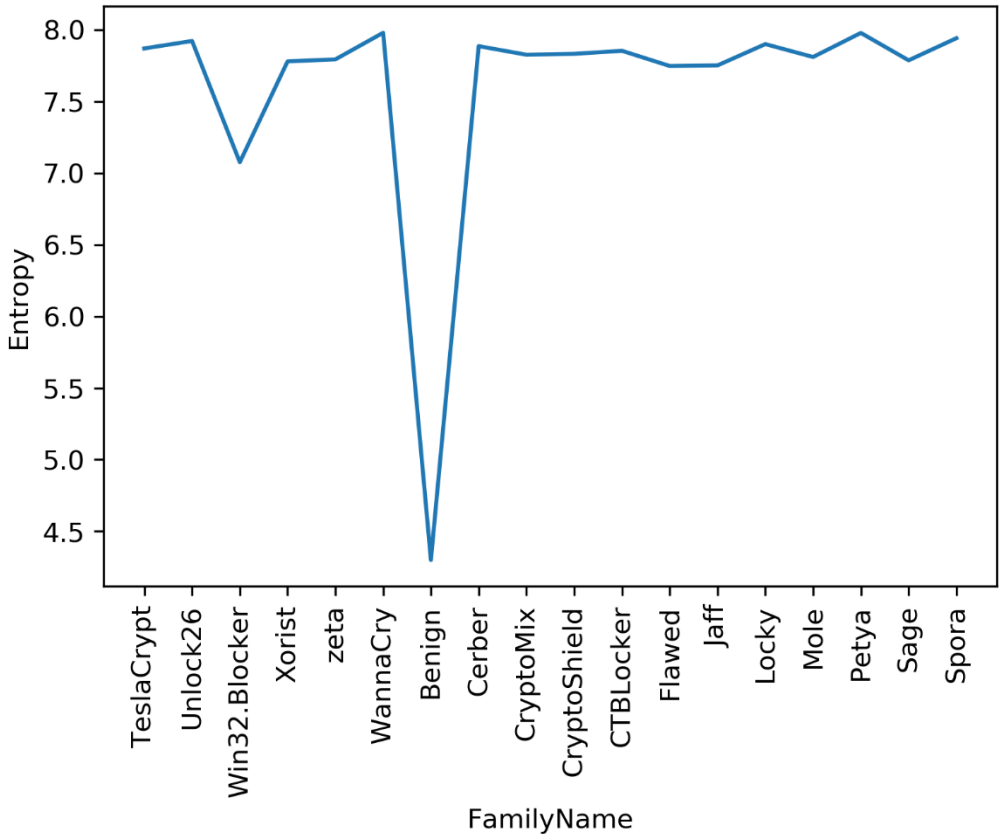

Figure 4.4 *Average entropy of encrypted files per family*

4.3 Registry Key operations

The Windows registry is a hierarchical database used in Windows Operating Systems to manage centrally system configurations and settings [31]. The data is structured in a key-value format where each key can have any number of values, and the values can be in any form (e.g., numeric, string, etc.). Whenever the user installs any software program, the initial configurations are stored as key-value pairs in the registry. When a user runs the software, the system components retrieve their run-time configuration from the registry database. Our sandbox analysis shows that the software executes four types of registry key operations to maintain the persistency of configurations across the reboots. We collected a total number of 27,739 unique registry key operations from the collected JSON reports (benign and ransomware). Table 4.3 provides a breakdown of the collected registry operations.

Registry key Operation	Count
Opened	5201
Deleted	199
Read	17693
Written	4646
Total Count	**27739**

Table 4.3 *Registry key operations and their counts*

Registry key operations can be unique per software. Two different software might not use the same registry keys operations. Figure 4.5 depicts the example of Windows registry key structure. From figure 4.5 we can observe that the highlighted registry hive “Apple Application support” has 5 configurations stored as key-value pairs. A registry hive is a local group of keys, subkeys, and values in the registry. In the above example, Name represents Key and Data represents Value, and the hierarchy of highlighted registry key areas is as follows:

HKEY_LOCAL_MACHINE→SOFTWARE→APPLE INC→Apple Application Support

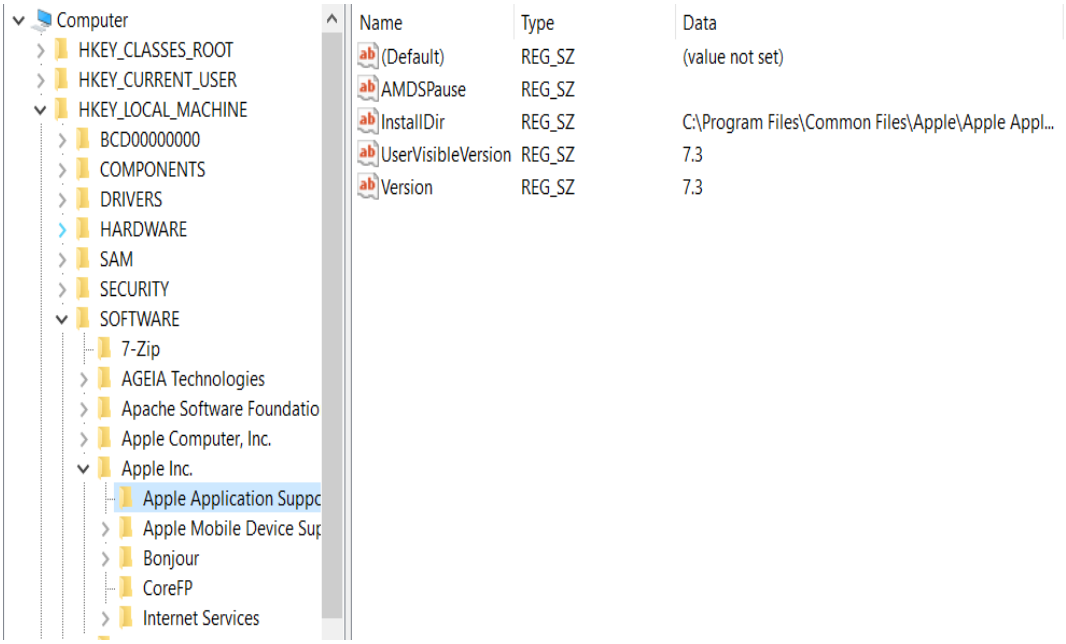

Figure 4.5 *Windows registry key structure*

To analyze the most impacted registry hives during the ransomware attack, we counted the total number of registry key operations done on each registry hive. There might be a case where one registry hive has multiple child registry hives and only one of the child registry hives is impacted during ransomware execution. In this case, considering child class and parent class of registry hives both as features, increases the chances of selecting repetitive data. To reduce the chances of training a model on repetitive data, we identified the registry key hives which are in linear correlation with each other. We calculated the Pearson correlation coefficient for each parent hive and its child hives. Pearson correlation coefficient is a measure of the linear relation between two variables [32]. The value of Pearson correlation coefficient lies between +1 and -1. A value

of 0 indicates that there is no linear relation between two variables. Values of 1 and -1 indicate positive and negative linear relations, respectively.

The Pearson correlation between two variables *x* and *y* is given by the following formula:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})\,(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

Where,

- n is the sample size
- x_i, y_i are the individual sample points indexed with i
- $\bar{x} = \frac{i}{n}\sum_{i=1}^{n} x_i$ (the sample mean); and analogously for $\bar{y}$

We identified the child hives having the correlation value 1 with its parent hive and removed those hives from our dataset. The reason for truncating the dataset to the best features is to avoid training the model on repetitive data, which ultimately overfits the model. The final breakdown of the registry hives selected as features are shown in the table 4.4.

Registry key Hives	Count
Opened	1211
Deleted	29
Read	1826
Written	931
Total Count	**3997**

Table 4.4 *Registry-key hives and their counts*

4.4 Command-line operations

The command prompt is a command-line interpreter application available in Windows operating systems. The command prompt is generally used to automate the tasks, troubleshoot operating system issues or perform administrative functions. For example, to list all the files and directories present in any specific location, the user can execute ‘dir’ command in a command prompt. Since most of the users use graphical user interface for convenience, ransomware leverages, in the backend, a part of an operating system that computer users rarely come in contact with. The ransomware utilizes this functionality to achieve goals like delete the master boot record, delete windows shadow copy, etc. We analyzed and extracted in total 2,770 important command line operations from our benign and ransomware binary execution reports.

4.5 Windows DLLs

A dynamic-link library (DLL) is a program that consists of functions and data which can be utilized by another application or module for code reusability purpose. Windows executables or programs may contain different modules, and each module of the developed program is distributed and contained in DLLs. By using DLLs, programmers can develop modular applications and functionality can be reused and updated easily. Windows APIs are implemented as a set of DLL files. In the backend, all Windows APIs use dynamic linking libraries. We extracted 404 common DLL files used by ransomware and benign applications from generated JSON reports.

4.6 Directories Enumerated

Ransomware, during encryption process, goes through all directories or specific set of directories of file system to encrypt the files. During this period, the number of directory changes

are very high because ransomware tries to encrypt as many files as possible. From collected reports we identified 34,809 directory enumerations and converted them to numeric features to generate count of directory enumerations in each binary execution.

4.7 Mutex

A mutex is a program object generally used by malicious software as a locking mechanism to avoid simultaneous access to a resource on the system. It is also used in parallel programming to allow multiple code threads to share the same resources and to avoid infection of the system more than once. For example, as soon as malware infects the system, the first step it takes is to obtain a handle to a "named" mutex. If the process fails, malware exits the code. Mutexes are character strings. Antivirus software based on static malware detection techniques looks for previously known mutex names to check the existence of the malware on the system. To evade detection, sometimes some malware avoids the usage of common mutex names and dynamically generate mutexes at runtime. Ransomware also utilizes this functionality to avoid infection of the system more than once. We extracted 603 important mutexes from collected JSON reports.

4.8 Embedded Strings

Strings are ASCII and Unicode character sequences embedded within the binary. String extraction from malware binaries can give a clue about the functionality of the program. For example, if malware tries to resolve the domain name, it might be stored as a string in a binary. It can also give important information like IP addresses, file names, registry key operations, etc. We extracted a total of 8,582 strings from ransomware and benign binaries.

4.9 Miscellaneous features

Ransomware also exhibits some common malware characteristics, such as getting control of keyboard shortcuts, changes in the boot sequence, stealing information from browsers, etc. We identified 105 such characteristics and used them as binary features. Some of the most important among these features are explained below.

Packer Entropy: To bypass the signature detection, malware authors use code obfuscation techniques. One of the purposes of packing malware is compressing and ciphering the real code. High entropy values in packed executables indicate a random distribution of the bytes, a prevalent property in compressed and ciphered data. High Packer Entropy is one of the measures for the detection of packed executables.

BCDEdit: BCDEdit is a command line tool available in Windows operating systems, to manage Boot configuration data. BCDEdit can be used for various purposes, such as adding a new boot menu option, creating a new store, modifying the existing store. Some of the ransomware variants use this command line option to change the boot menu options.

Deletion of Windows shadow copy: Ransomware delete all system backup files to make sure files cannot be recovered once encrypted.

Table 4.5 provides a breakdown of all the extracted features, their types and the number of features per category.

Feature Name	Feature Type	Number of Features
API calls	Numeric	286
Registry Key Operations	Numeric	3,997

Command line operations	Binary	2,770
DLLs	Binary	404
Directories Enumerated	Numeric	34,809
Mutex	Binary	603
Embedded Strings	Binary	8,582
Miscellaneous	Binary	105
File Entropy and Signature	Numeric	NaN

Table 4.5 *Feature set*

4.10 Summary

This chapter discusses nine different features sets and introduces our proposed feature model. It is observed that behavioral characteristics of ransomware in Windows environment is different from benign applications. We selected a total number of 51,556 features to process with our machine learning model. In the next chapter, we conduct different experiments and compare the performance results of different machine learning classifiers.

Chapter 5 : Experiments and detection architecture

One of the most important steps before training a machine learning classifier and post feature selection is data preprocessing. Data preprocessing is a technique used in data science to transform raw data into understandable data. Steps in data preprocessing include[33] data cleaning, data correction, standardization, noisy data reduction, etc. In this chapter, we start by discussing the data standardization technique we used to transform the data in a proper form, and then present the feature selection technique used. Finally, we discuss and compare the performance results of the different machine learning classifiers considered and introduce our final system for Windows ransomware detection.

5.1 Data Standardization

Data standardization, also known as data scaling, is the method used to normalize the range of features values. The objective of data standardization is to change the values of numeric features to use a standard scale. This process ensures that feature values are changed without distorting differences in the range of values or losing information. Feature scaling is an essential part of data preprocessing when dataset features are of varying scales. In our dataset, we have two types of values: numeric and binary. Some machine learning classifiers calculate the Euclidean distance between two feature vectors [34]. The features with wide range of values will have more influence on the machine learning model than the features with low range of values. Therefore, the range of features should be standardized for an equal contribution of features to the final distance. To achieve feature scaling, we used a min-max scaler to transform the data such that data has all values between 0 and 1.

For a min-max scaler given an explicit feature range = (min, max) the formula is as follows:

$$X\_scaled = X\_std \times (max - min) + min$$

Where X_std is the standard deviation, which is calculated as follows:

$$X\_std = (X - X.min) / (X.max - X.min)$$

5.2 Feature selection

Feature selection is one of the essential steps in building a machine learning classifier. It consists of automatically or manually identifying the most important features and discarding irrelevant or less important features that do not contribute much to the model effectiveness. Often, in a dataset with a large number of features, there is a segment of data which is entirely irrelevant and unimportant. Discarding insignificant data helps improve accuracy, avoid unnecessary resource allocation, and eliminate redundant data. There are three different methods available for feature selection, including the following: filter methods, wrapper methods, and embedded methods. Filter methods, which are also called univariant selection methods, apply statistical measures to all available features, and assign scores to them according to the target class. Important features are assigned high score values, and unimportant ones are assigned low values. The wrapper techniques, on the other hand, prepare different combinations of different features, and evaluate and compare them to other combinations. The recursive feature elimination method is an example of wrapper method where unimportant features are discarded one by one recursively. Embedded methods learn the best features of the model while the model is being created. The most commonly used embedded feature selection methods are regularization methods[35].

From selected features, we wanted to identify the features which are most important to classify the ransomware. This requirement makes filter methods the best option for us. Also, we regularize our models manually at the later stage to reduce overfitting. This way, we used two

feature selection methods, filter methods in the data pre-processing stage and embedded methods while training a machine learning model. An example of filter methods is the Chi-squared statistical test. We used Scikit-learn library SelectKbest with Chi-squared test to select the relevant features.

5.2.1 Chi-Square (CHI) Test

In statistics, the Chi-square test is applied to test the independence of two events. Two events A and B, are called independent when their joint probability is: $P(AB) = P(A)\,P(B)$ or equivalently the conditional probability $P(A|B) = P(A)$ and $P(B|A) = P(B)$. Given the occurrences of two events, we can get two counts: observed counts and expected counts. The Chi-square score helps to measure how much expected count 'E' and observed count 'O' deviate from each other.

The calculation of Chi-square statistic is quite straightforward, and done as follows:

$$x^2 = \sum \frac{(f_0 - f_e)^2}{f_e}$$

Where f_0 is the observed frequency and f_e is the expected frequency if no relationship existed between the variables.

As per the equation, the Chi-square statistic is based on the difference between what is observed in the data and what would be expected if there is no relationship between the variables. A very small chi square value means observed data fits expected data very well. In this case, there is a relationship. A very large Chi-square value means that the data do not fit well, so there is no relationship. All features are assigned a score after calculation of Chi-square statistics between

every feature variable and the target variable. We discarded features with low scores and features with high scores were marked as important and selected to train a machine learning model.

5.3 Machine Learning Classification

In our dataset, malware data dominate benign data, which is not the case in the real world. In machine learning and data science, this scenario is called imbalanced class distribution. There are chances that the predictive model developed using this dataset could be biased and inaccurate. Various data sampling techniques are available in data science to overcome these challenges. We evaluated our detection model using both the original dataset and the resampled dataset. We studied and compared three different machine learning models, including, linear SVM, Logistic Regression, and Random Forest classifiers. We used python Scikit-learn libraries for training our data.

5.3.1 Machine Learning in Imbalanced dataset

We explored the Chi-square feature selection range between 50 to 500 features. Once, feature selection is applied to the model, the final step is to train a classifier for detection of ransomware. In our case, because of having a high number of features, linear classifiers are a good choice. We chose Logistic Regression, SVM, and Random Forest for training a machine learning model and applied 10-fold cross-validation on trained models. Figure 5.1 depicts the accuracy of the classifiers when varying the number of features. In the case of SVM, the average accuracy of the model stayed around 94%, while for Logistic regression and Random Forest, the accuracy varied from 94 to about 97%. It can be noted that for Logistic regression and random forest classifiers, as the number of features increases, accuracy also increases. However, with an increase in the number of features, the chances of overfitting also increase. We observed that with 400

features, we achieved the highest average accuracy in all models. Therefore, we used this number as the cut-off to select our features.

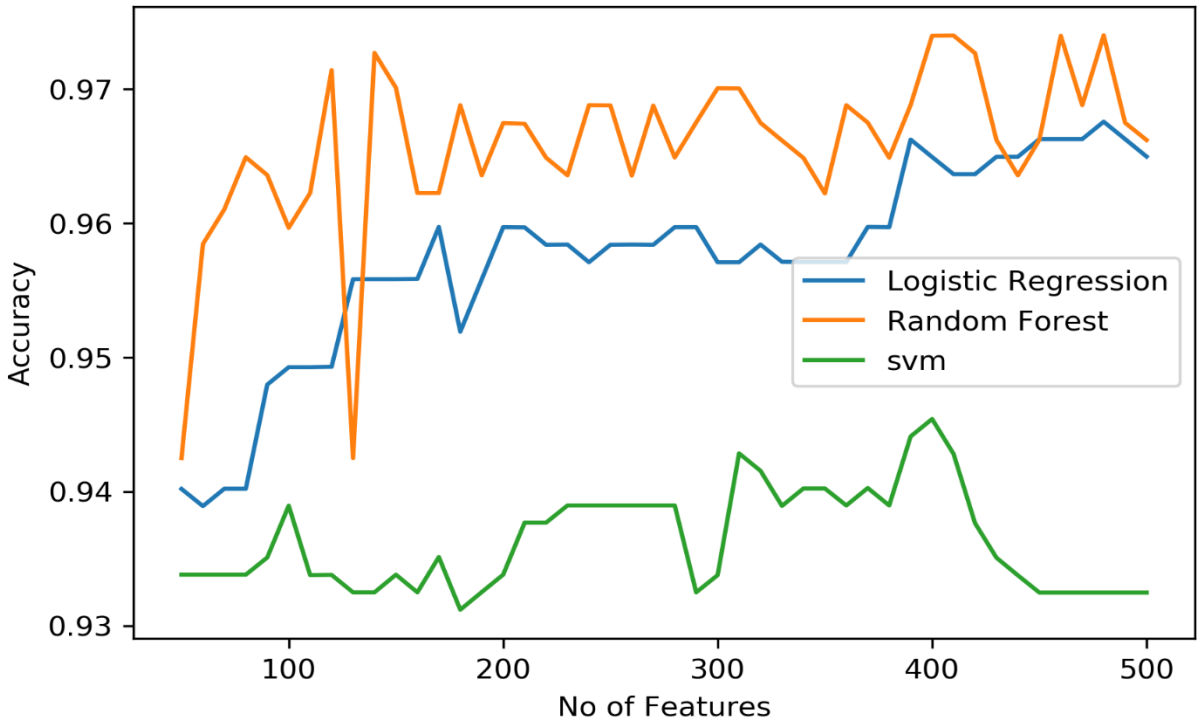

Figure 5.1 *Classification accuracy when varying the number of features*

Table 5.1 shows the distribution of the 400 important features selected through the chi-square feature selection method. From the distribution it can be observed that API calls and registry key operations cover major portions of the selected features. However, the presence of other features also plays an important role to achieve high detection rate.

Feature Type	Top 100
API calls	159
Registry Operations	225

Command line Operations	1
Directories enumerated	1
Miscellaneous	7
mutex	5
Embedded strings	2

Table 5.1 *Top 400 features distribution*

From figure 5.1 we can conclude that, our detection models achieved the highest accuracies while using random forest and logistic regression algorithms. Also, random forest and logistic regression algorithms are easy to train and execute compared to SVM. Compared to random forest, logistic regression is less prone to overfitting. However, the maximum likelihood estimation of posterior probability used in the logistic regression algorithm is also prone to overfit if a large number of features are used to train the classifier[36]. This problem can be addressed by model regularization. We first trained a logistic regression model without any regularization and checked the performance metrics. The training was done on 80% of the dataset and the remaining 20% was kept aside for testing. Table 5.2 and Figure 5.2 shows the classification results and the confusion matrix, respectively, for normal logistic regression classifier.

	precision (%)	recall (%)	f1-score (%)	support
Benign	92	73	81	15
Ransomware	97	99	98	140
micro avg	97	97	97	155
macro avg	94	86	90	155
weighted avg	97	97	97	155

Table 5.2 *Classification results for Logistic Regression*

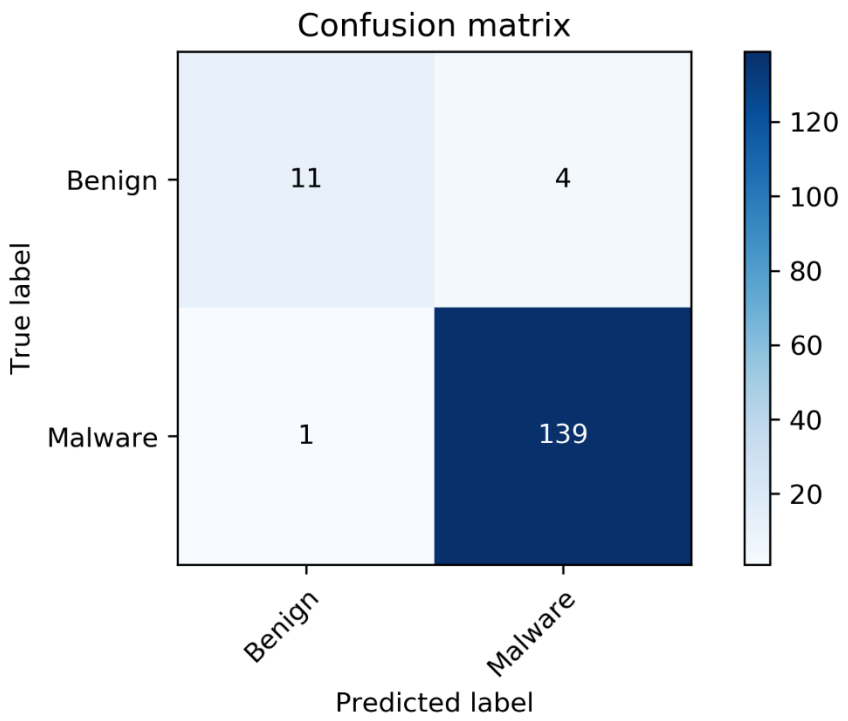

Figure 5.2 *Confusion matrix for logistic regression*

We can observe from Figure 5.2 that the model has falsely predicted 1 ransomware and 4 benign samples. The classification results achieve the accuracy of 96.7%. We also wanted to compare achieved results with regularized logistic regression classifier. Regularization is a technique that allows finding a good bias-variance trade-off by tuning the complexity of the model. Regularization does not improve classification performance on the dataset on which the algorithm initially learns. However, it can improve the prediction accuracy of new and unseen data. By adding the bias, we can remove overfitting; but adding too much bias also result in under-fitting. The concept behind regularization is to introduce additional bias to penalize extreme parameter weights.

L2 regularization equation is defined as follows [37]:

$$\frac{\lambda}{2}\|w\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} {w_j}^2$$

where λ is a regularization parameter, w is the parameter weight matrix, and w_j are the individual parameter weights.

Note that in our system, we have two mechanisms to prevent overfitting: the feature selection with the Chi-square and the regularization of the Logistic Regression. Regularized logistic regression is competitive to SVM and easy to train when using different regularization values. To apply regularization to our logistic regression term, we used C as a regularization parameter available in a Scikit-learn library. Regularization parameter available in Scikit-learn library is inverse to the regularization parameter λ:

$$C = 1/\lambda.$$

Lambda (λ) controls the trade-off between allowing the model to increase its complexity as much as possible with trying to keep it simple. For example, if λ is very low or null, the model will have enough power to increase its complexity (overfit) by assigning high values to the weights for each parameter. On the other hand, if we increase the value of λ, the model will tend to underfit, as the model will become too simple. Parameter C will work conversely. For small values of C, the model will underfit the data, and for big values of C, the model will overfit the data. To make an appropriate selection of value for C, we ran our logistic regression model through 10-fold cross validation in the narrow range of C between 0.001 and 100. Figure 5.3 depicts the model accuracy when varying C. As shown in the figure, with increase in the value of C the model accuracy also increases. From the graph we can observe that with the value of C=100, we obtain the highest accuracy at around 97.5%.

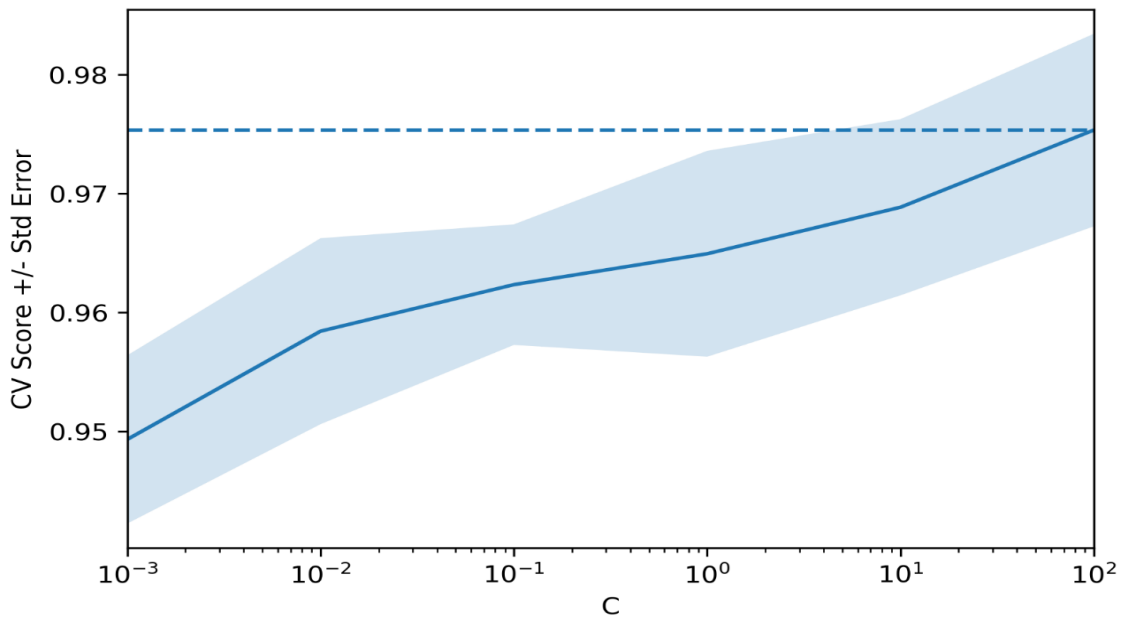

Figure 5.3 *Logistic regression accuracy for different values of the regularization parameter C*

We divided the dataset into 80:20 ratio to evaluate the regularized logistic regression classifier. We trained a logistic regression model with the value of C=100 using 80% of the data and the remaining 20% data was used to test the trained classifier. Table 5.3 shows the evaluation results for the classifier. For ransomware, we achieved a recall of 100%, which means we were able to detect all ransomware in the testing stage. While, from the confusion matrix shown in Figure 5.4, we can also observe that our model achieved an accuracy of 98.7% with a false positive rate of 1.41% as the model misclassified two benign applications as ransomware applications.

	precision (%)	recall (%)	f1-score (%)	support
Benign	100	87	93	15
Ransomware	99	100	99	140
micro avg	99	99	99	155
macro avg	99	93	96	155
weighted avg	99	99	99	155

Table 5.3 *Classification report for regularized logistic regression classifier*

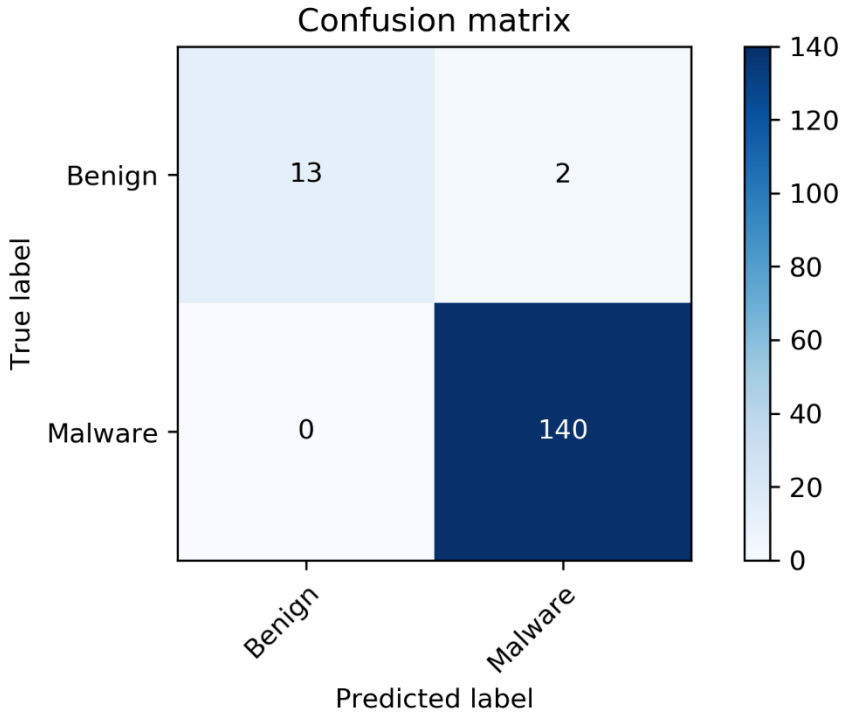

Figure 5.4 Confusion matrix for regularized logistic regression classifier

5.3.2 Hyper-parameter Tuning

While we managed to improve the base logistic regression model, we also wanted to check the accuracy of random forest and SVM models by tuning the model parameters. A hyperparameter is a parameter whose value is set before training the machine learning model. The model parameters are configuration variables internal to the model. A model can run with default parameters. However, the performance of the model can be improved by tuning the parameters. To achieve the best performance, we identified the parameters which have the best impact on classification results for both models. For random forest, we chose "n_estimators" and "max_depth" hyperparameters, and for SVM we tuned the parameter C available in Scikit-learn

library. Figures 5.5 and 5.6 depict the accuracy of the random forest through 10-fold cross-validation when varying the value of n_estimators and max_depth parameters, respectively.

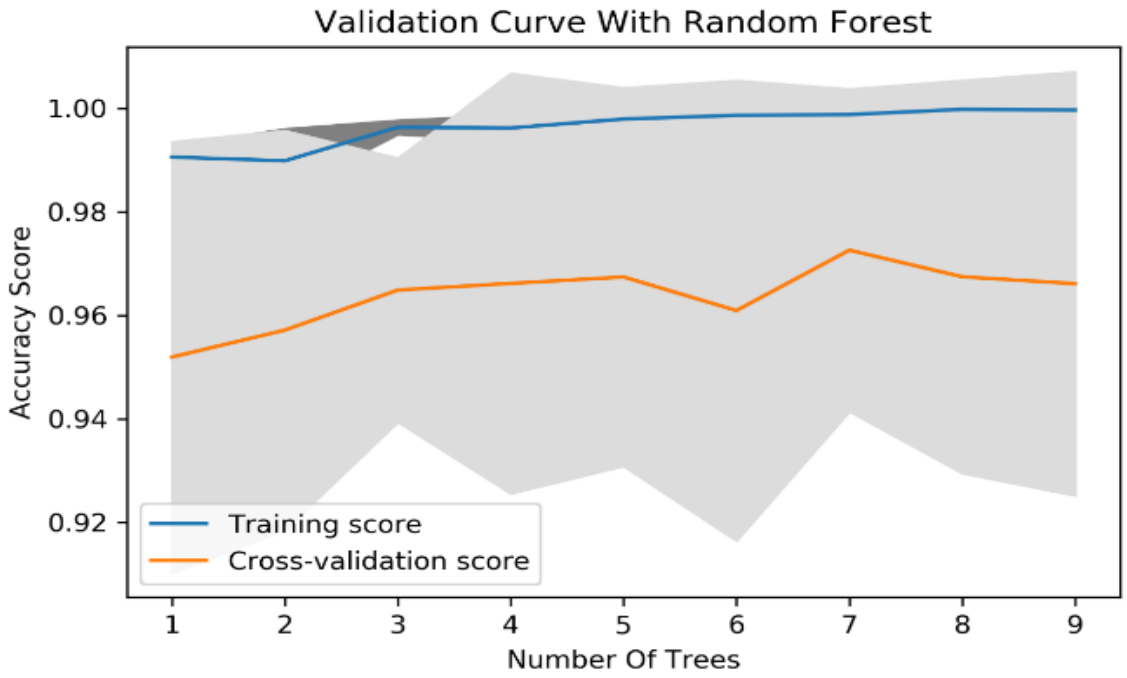

Figure 5.5 *Random forest 10-fold cross validation score for different values of "n_estimators"*

We can observe from both figures that the classifier achieved highest accuracy at value 7 in both parameters. We finally trained our random forest model using the aforementioned best parameters; table 5.4 summarizes the obtained results. Random forest was also successful to detect all ransomware samples. However, the number of false positives increased in this case. From the confusion matrix shown in figure 5.7, we can observe that 4 benign applications were misclassified as ransomware.

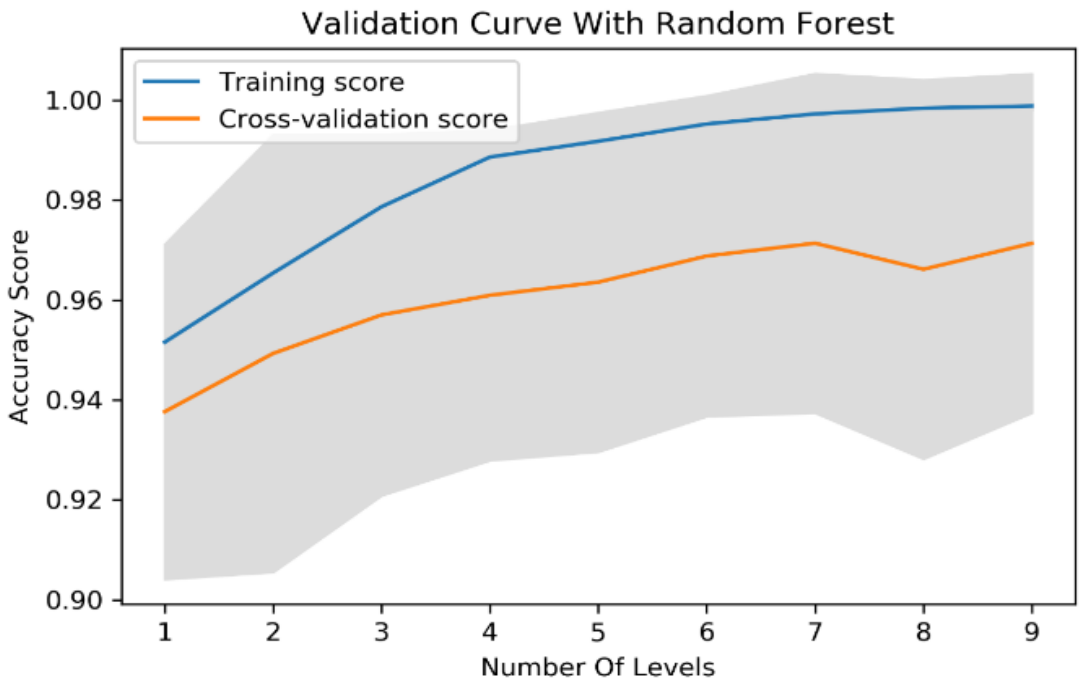

Figure 5.6 *Random forest 10-fold cross-validation score for different values of "max_depth "*

	precision (%)	recall (%)	f1-score (%)	support
Benign	100	73	85	15
Ransomware	97	100	99	140
micro avg	97	97	97	155
macro avg	99	87	92	155
weighted avg	97	97	97	155

Table 5.4 *Classification report for fandom forest post hyperparameter tuning*

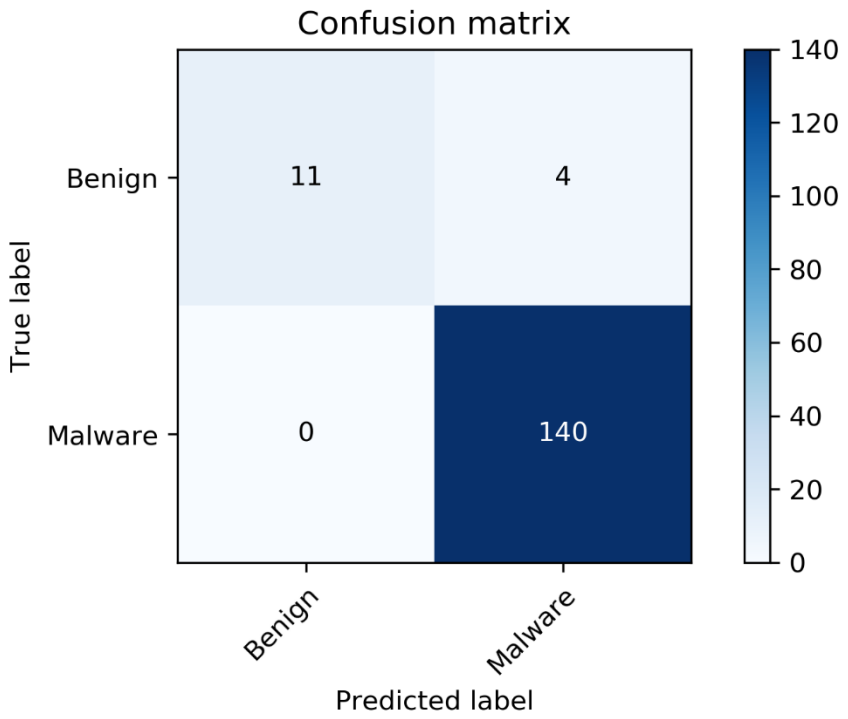

Figure 5.7 *Confusion matrix for random forest classifier post hyperparameter tuning*

For support vector machine, we chose parameter C, the penalty parameter of the error term. This parameter controls the trade-off between classifying the training samples correctly and achieving smooth decision boundary. We did 10-fold cross-validation by varying the value of this parameter in the range of 0 to 500. It can be observed from Figure 5.8 that after reaching a value of 50, the accuracy becomes steady. We chose 50 as a tuned value for SVM and generated classification metrics for the tuned model. As we can observe from Figure 5.9, classification accuracy is decreased to 95% in this case. SVM was not able to detect one ransomware sample, and it misclassified six benign samples as ransomware.

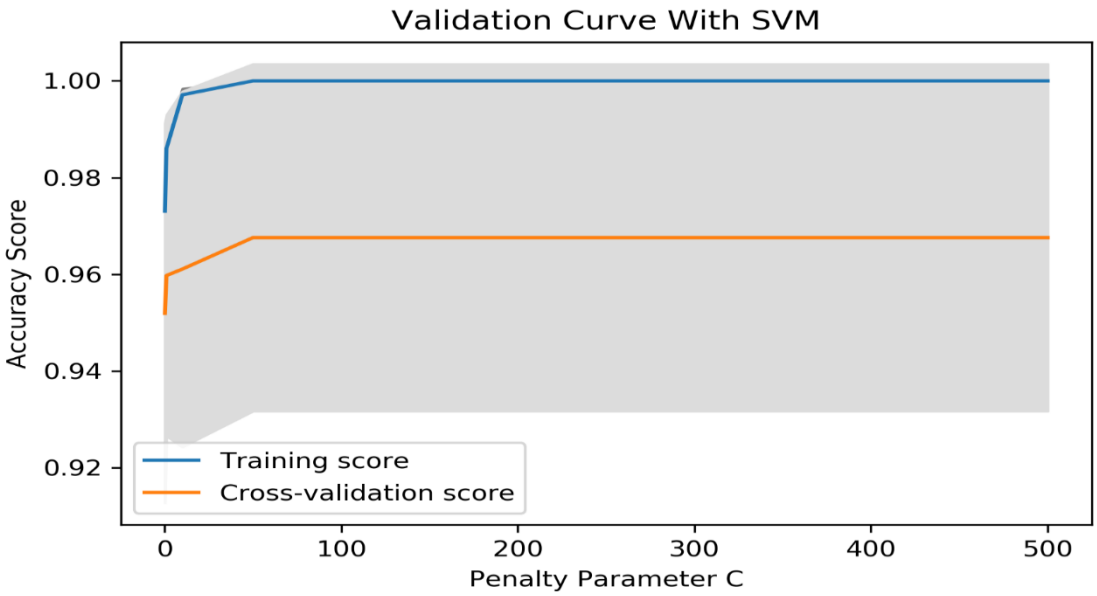

Figure 5.8 *SVM 10-fold cross-validation score for different values of parameter C*

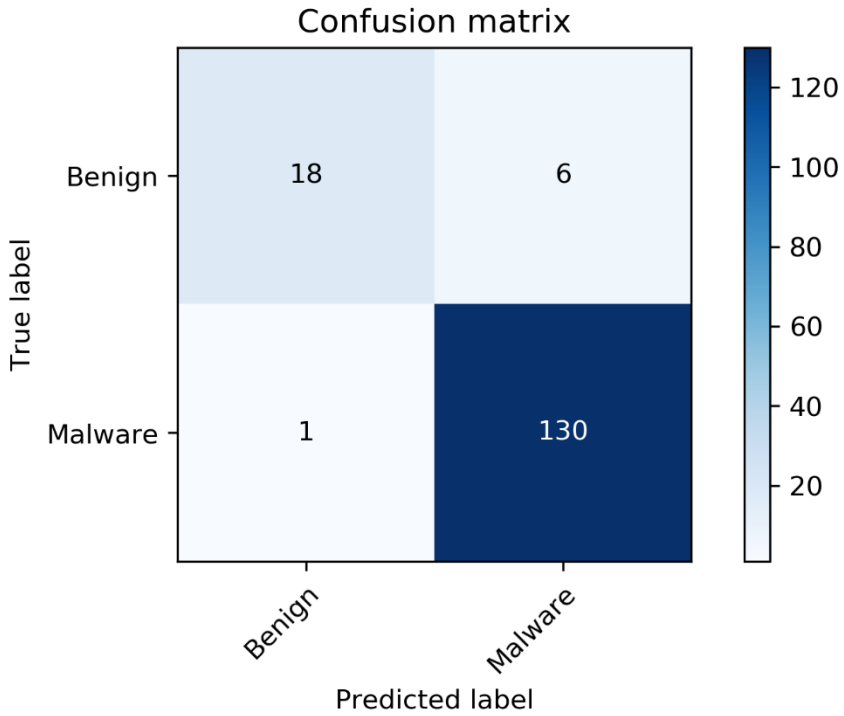

Figure 5.9 *Confusion matrix for SVM post hyperparameter tuning*

	precision (%)	recall (%)	f1-score (%)	support
Benign	95	75	84	24
Ransomware	96	99	97	131
micro avg	95	95	95	155
macro avg	95	87	91	155
weighted avg	95	95	95	155

Table 5.5 *Classification report for SVM post hyper-parameter tuning*

5.3.3 Machine Learning using Balanced dataset

We balanced the dataset using data resampling based on the synthetic minority oversampling technique (SMOTE). SMOTE uses K-nearest neighbours 'algorithm to generate new data for underrepresented class in the dataset. Instead of just replicating the minority class, this statistical technique takes samples of each target class nearest to the decision boundary and generates the new samples which combine the features of the target class and its neighbors. In our dataset, we have benign samples as a minority class. Before applying the SMOTE technique, we split our data in 80:20 ratio for testing and training purpose. Post data split, our dataset consisted of 616 samples in total, consisting of 536 ransomware and 80 benign samples. SMOTE oversampled our minority class and balanced the dataset in equal distribution of each class, as shown in figure 5.10. We trained the dataset on a total of 1,072 benign and ransomware samples using regularized machine learning model. Testing was done on a total of 155 samples consisting of 132 ransomware and 23 benign samples. As we can observe from the Figure 5.11, the accuracy decreased as compared to the previous classifier. The model achieved an accuracy of about 97.5 %, and it generated three false negatives and one false positive. However, these results are always debatable as SMOTE blindly generates the minority classes without regard to the majority class. As a result of that, it can increase the overlapping of the classes and introduce additional noise.

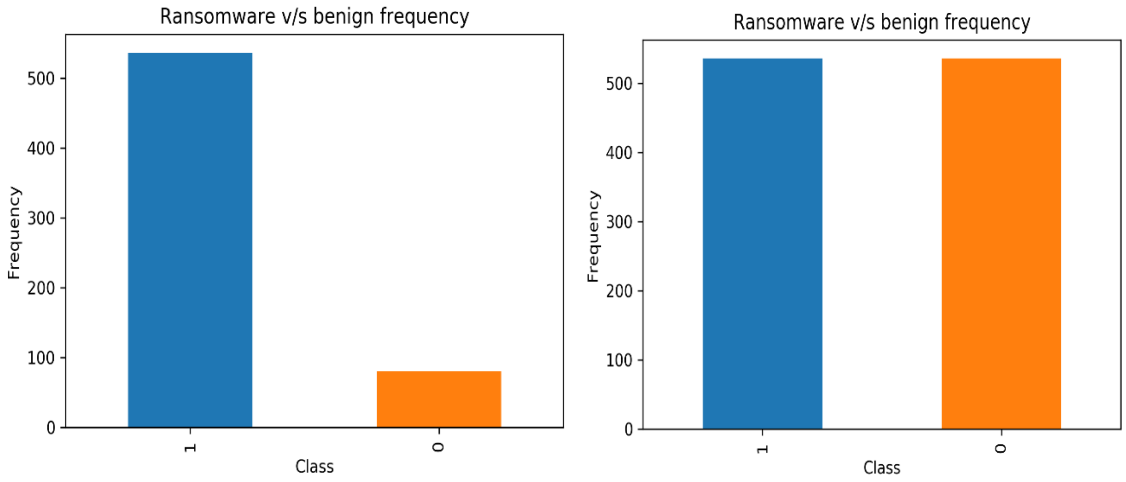

Figure 5.10 *Class distribution before and after SMOTE*

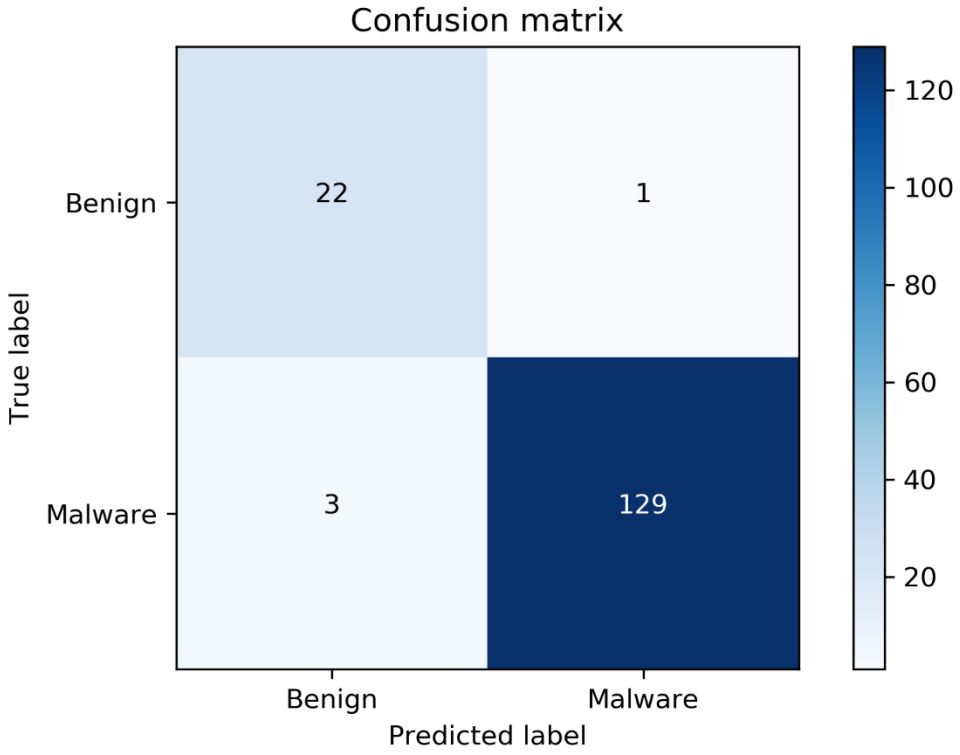

Figure 5.11 *Confusion matrix of regularized logistic regression after SMOTE*

	precision (%)	recall (%)	f1-score (%)	support
Benign	88	96	92	23
Ransomware	99	98	98	132
micro avg	97	97	97	155
macro avg	94	97	95	155
weighted avg	98	97	97	155

Table 5.6 *Classification report for regularized logistic regression post SMOTE*

5.4 Novel Ransomware Detection

Machine learning has great potential to detect the new variants of ransomware. We achieved the best results in the regularized logistic regression model and decided to use it for further data testing. To test our machine-learning model for novelty detection, we excluded some ransomware families while training the regularized logistic regression classifier. We excluded two families one by one: Cerber and Locky. We tested each family individually and calculated the performance metrics for each test. The test dataset also contained 30% of benign samples from our ISOT dataset, which were not part of the training dataset. Benign data was randomly selected from our current dataset to evaluate each family. It can be observed in figure 5.12, for each family, the model classified all ransomware samples correctly. However, the model that evaluated cerber family identified two benign samples as ransomware. Correspondingly, the model which was used to evaluate Locky ransomware family identified four benign samples as ransomware. Table 5.7 and Table 5.8 show the classification reports for both ransomware families. Recall for new ransomware families is 100%, which means our model successfully detected all previously unseen ransomware samples. However, the machine learning models trained for Cerber and Locky ransomware families achieved false-positive rates of 1.61% and 3 %, respectively.

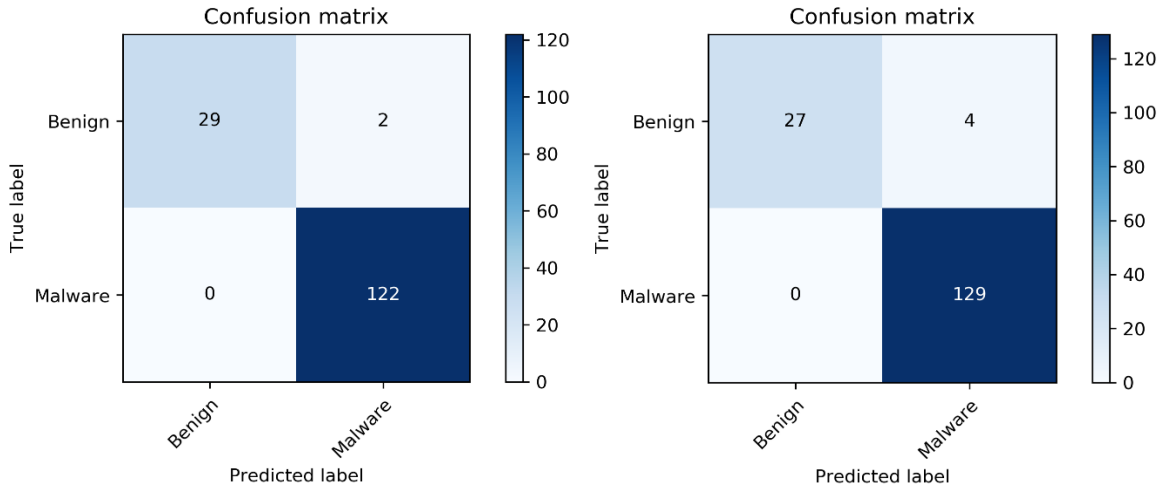

Figure 5.12 *Confusion matrix for Cerber(Left) and Locky(Right) ransomware families*

	precision (%)	recall (%)	f1-score (%)	support
Benign	100	94	97	31
Ransomware	98	100	99	122
micro avg	99	99	99	153
macro avg	99	97	98	153
weighted avg	99	99	99	153

Table 5.7 *Classification report for Cerber ransomware family*

	precision (%)	recall (%)	f1-score (%)	support
Benign	100	87	93	31
Ransomware	97	100	98	129
micro avg	97	97	97	160
macro avg	98	94	96	160
weighted avg	98	97	97	160

Table 5.8 *Classification report for Locky ransomware family*

5.5 Ransomware-triggered vs. User-triggered Encryption

We wanted to identify the difference between ransomware-triggered and user-triggered encryption. For this purpose, we collected all user files from guest machines post binary execution. We also collected the timestamps when ransomware and benign binaries were executed in a sandbox environment. Then, we sorted all the files according to their last access time and filtered out the files for which the entropy values were greater than 7 and the file signatures were missing. To check the file signature, we used "python-magic" library available in python. As depicted in Figure 5.13, Teslacrypt binary was executed around 10:42 PM of 4th April 2018. After almost 21 minutes of the successful execution of the binary, this Teslacrypt variant started generating the new files by appending '.mp3' in the filename. The files were identified as "data" files by the python-magic library. Extension value "data" means python-magic library was unable to identify the file type.

FamilyName	ID	FileName	Time	Entropy	Extension
TeslaCrypt	18	binary	2018-04-04 22:42:46.506609974-04:00		
TeslaCrypt	18	2d8d62c25e4b3c58_threading.py.mp3	2018-04-04 23:03:05.522609829-04:00	7.995654	data
TeslaCrypt	18	45bc84d034dc6957_time_hashlib.py.mp3	2018-04-04 23:03:06.714609828-04:00	7.894587	data
TeslaCrypt	18	fbeb89dab4af5eea_test_simplehttpserver.py.mp3	2018-04-04 23:03:07.142609828-04:00	7.767438	data
TeslaCrypt	18	9e166adcc6e4b5c0_unicode_escape.py.mp3	2018-04-04 23:03:07.142609828-04:00	7.747635	data
TeslaCrypt	18	f841c710f6332348_nonmultipart.py.mp3	2018-04-04 23:03:07.178609828-04:00	7.563487	data
TeslaCrypt	18	994be0c490ae905e_000310.ppt.mp3	2018-04-04 23:03:07.178609828-04:00	7.999364	data
TeslaCrypt	18	47412c4362fb98f8_099459.pdf.mp3	2018-04-04 23:03:07.178609828-04:00	7.99848	data
TeslaCrypt	18	8b075c5c07de7d69_queues.py.mp3	2018-04-04 23:03:07.202609828-04:00	7.980874	data
TeslaCrypt	18	0535d532d3bf0759_sbcharsetprober.py.mp3	2018-04-04 23:03:07.290609828-04:00	7.94485	data
TeslaCrypt	18	d348b026f7910598_raw_unicode_escape.py.mp3	2018-04-04 23:03:07.346609828-04:00	7.750226	data
TeslaCrypt	18	62f738440e72bc85_repr.py.mp3	2018-04-04 23:03:07.346609828-04:00	7.940089	data
TeslaCrypt	18	b4637ade823bc5d3_fix_except.py.mp3	2018-04-04 23:03:07.366609828-04:00	7.912052	data
TeslaCrypt	18	085da15528f96858__structures.py.mp3	2018-04-04 23:03:07.662609828-04:00	7.74818	data
TeslaCrypt	18	3e159c6e0e5287fe_glob.py.mp3	2018-04-04 23:03:07.662609828-04:00	7.945073	data
TeslaCrypt	18	8a85ec2ca73d39d3_palmimageplugin.py.mp3	2018-04-04 23:03:07.682609828-04:00	7.973191	data
TeslaCrypt	18	092b341d493b2804_099594.pdf.mp3	2018-04-04 23:03:07.834609828-04:00	7.995892	data
TeslaCrypt	18	0c01634b2fbfe292_test_codecencodings_hk.py.mp3	2018-04-04 23:03:07.866609828-04:00	7.611813	data

Figure 5.13 *Teslacrypt encrypted files with timestamp*

FamilyName	ID	FileName	Time	Entropy	Extension
zeta	563	binary	2018-04-18 11:09:10.066854722-04:00		
zeta	563	965b6a4bce52fd73_ar_jo.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:34.934854505-04:00	7.902286	data
zeta	563	5d4ea3dda699f8de_fractions.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:35.026854504-04:00	7.992504	data
zeta	563	eb402003cd883f9f_es_pa.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.926854504-04:00	7.08214	data
zeta	563	b42559f788114a34_test_funcattrs.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.962854504-04:00	7.986876	data
zeta	563	5f777056d34558dd_pythread.h.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:37.986854504-04:00	7.843095	data
zeta	563	f43381fa94795bc3_albumartsmall.jpg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.026854504-04:00	7.964216	data
zeta	563	da3be7867462eb63_099483.pdf.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.046854504-04:00	7.99506	data
zeta	563	43beebdfd465dfb3_test_pstats.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.046854504-04:00	7.809335	data
zeta	563	f98990fd10b58834_dummy_threading.py.id_60f3beebafa7737_email_enc6@dr.com_.s	2018-04-18 11:39:38.046854504-04:00	7.93857	data
zeta	563	3d119652fb11b276_es_pe.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.050854504-04:00	7.143268	data
zeta	563	7b70fffa7ee35b4d_592572.pptx.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.062854504-04:00	7.999946	data
zeta	563	f1036e8f256693c4_lv.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.066854504-04:00	7.856969	data
zeta	563	dd20f209d0f3afa5_test_dictviews.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.090854504-04:00	7.980894	data
zeta	563	7fbb677e14b5989a_test_dbtables.py.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.110854504-04:00	7.987712	data
zeta	563	bd605e68cc1e38d1_099097.jpg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.146854504-04:00	7.995059	data
zeta	563	aa342ed4eaf446db_es_cr.msg.id_60f3beebafa7737_email_enc6@dr.com_.scl	2018-04-18 11:39:38.150854504-04:00	7.232002	data

Figure 5.14 *Zeta encrypted files with Timestamp*

In addition to that, the files which were missing the file signature had an entropy value close to 8

Figure 5.14 shows the filename, timestamp, entropy, and extension values of the files post execution of one of the ransomware binaries from zeta family. After 30 minutes of binary execution, this variant also started generating ".scl" files of high entropy and unknown file signature.

We did timestamp analysis for all generated user files and all ransomware binaries. Post analysis we came to the conclusion that ransomware, after specific period of time from execution, starts generating random files of high entropy with unknown file signatures. On one hand, the files encrypted with legitimate encryption tools like 7-zip, WinRAR, and so on, correspond to known file formats whose file signatures can easily be identified. On the other hand, ransomware generate unknown file extensions. This behavior of ransomware can be helpful to identify the ransomware which are not detected by current anti-virus program or firewall and save the user files with minimum file loss.

To be assured about the files encrypted by ransomware are high entropy values, we filtered out the files which had missing file signatures and calculated the average entropy per ransomware family. Table 5.9 shows the average entropy of the files which are missing file signatures. We can see from the table that the average entropy for almost all the ransomware families is greater than 7. Crysis and striked families are the exceptions in our case, having an average entropy of 2.21 and 5.32, respectively. Upon checking the files of the crysis and striked ransomware families, we found out that despite the successful execution of the ransomware binaries, the binaries were unable to start the encryption process. Possible reasons for such failure may include the inability of these binaries to connect to their C&C servers, or the fact that the 30-minute threshold we used to execute the ransomware might be very low for these particular ransomwares to start encryption, and so forth.

Family	Average Entropy
CTBLocker	7.85
Cerber	7.88
CryptoMix	7.82
CryptoShield	7.83
Crysis	2.21
Flawed	7.75
GlobeImposter	7.45
Jaff	7.75
Locky	7.90
Mole	7.81
Petya	7.98
Sage	7.79
Satan	7.29
Spora	7.94
Striked	5.32
TeslaCrypt	7.87
Unlock26	7.92
WannaCry	7.98
Win32.Blocker	7.07

Xorist	7.78
zeta	7.79
Benign	4.30

Table 5.9 *Average File Entropy per Family*

5.6 Proposed Multilayer Detection Architecture

Our multilayer detection model involves two separate detectors that work in tandem. The first detector consists of a pre trained machine learning (ML) classifier, while the second detector makes its decision based on a file entropy threshold and file signature database. Figure 5.15 presents the architecture designs for the two detectors in separate tracks.

Although as per the experimental evaluation our machine learning model achieves a considerably high detection rate, there might be a case when a new variant of the ransomware evades detection. Our system continuously monitors the high-entropy operations of unknown file signatures. When a series of high-entropy operations with an unknown file signature takes place in the device, our system initiates immediately the backup of the user files and classifies the process as malicious. By utilising this system, user data can be saved with minimum file loss. Figure 5.16 presents the system architecture showing the working of the two detectors in tandem.

When a sample binary is received, features are extracted and then classified using the first detector into either legitimate/normal or ransomware. In the case where a ransomware decision is made by the first detector, an automatic backup is triggered for the files being modified. At the same time, the second detector computes the file entropy for the corresponding files and compares them against a predefined threshold. If the entropy is larger or equal to the threshold, the detector then checks the file signatures. If the file signatures do not match the expected signatures, are unknown or are unreadable, then the initial ransomware classification (i.e., made by the ML

classifier) is confirmed, and the modified files are automatically replaced by using their original version stored in the backup. Otherwise, if the file signatures match the expected ones, then the sample binary will be reclassified as normal (i.e., by reverting the initial ML decision). In the case where the initial classification for the sample binary by the ML model is normal, the second detector will apply the same process as above by computing and comparing the file entropy against the threshold and checking the file signatures. If the file entropy is lower, then the original decision of normal will be confirmed. Similarly, if the entropy is higher while the signatures do match, then the classification as normal will be confirmed as well. Otherwise, if the entropy is larger than the threshold and the signatures do not match, a warning will be generated, and the backup will be triggered automatically.

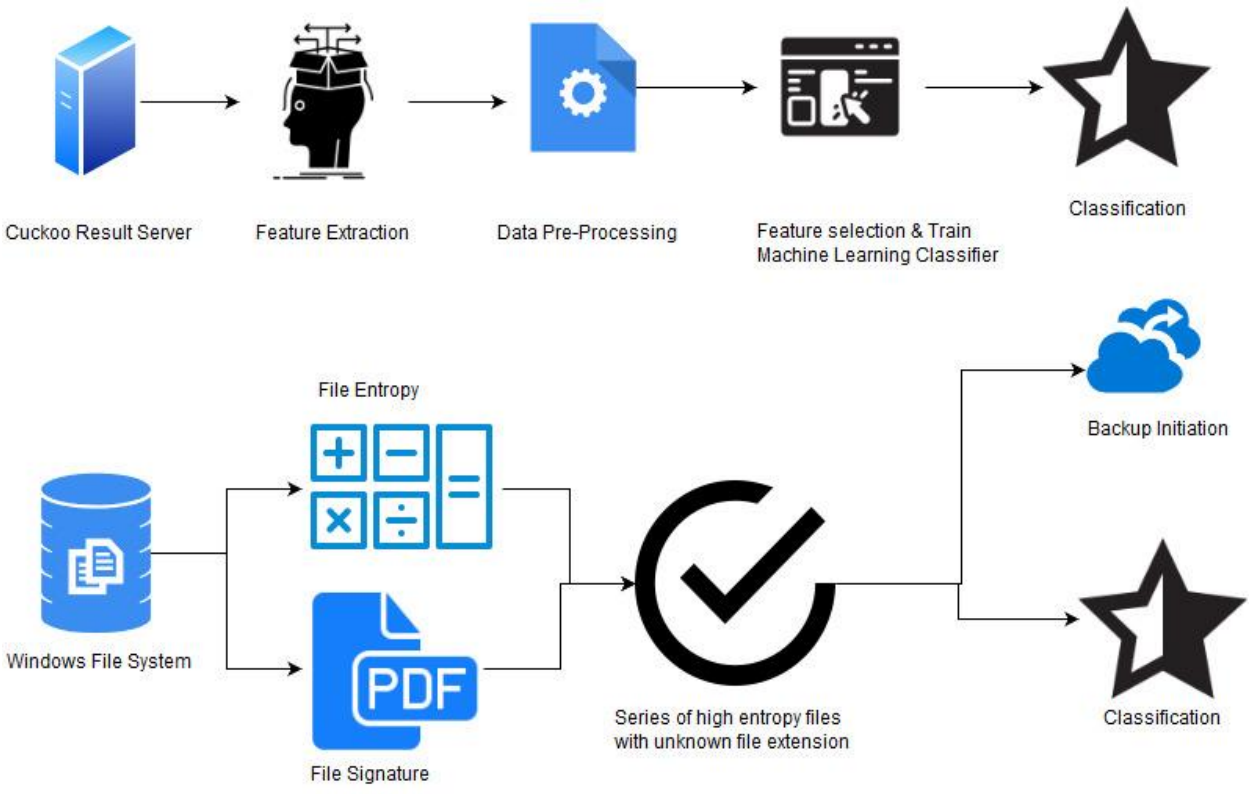

Figure 5.15 *ML and file entropy/signature detectors.*

A legitimate question about our model can be why keep the ML layer because all its decision will be challenged by the second detector, whether such a decision is ransomware or normal. A straightforward answer to that question is that the ML model provides early warning of possible ransomware and then triggers the backup immediately without delay. This way a malware can be detected with minimum file loss. In the case where the ML detector fails to detect the ransomware (i.e., in the case of a false negative), the second detector would then catch the corresponding sample and trigger the response accordingly. Of course, in the latter case the backup would be slightly delayed, and there would be more chance of loosing some files.

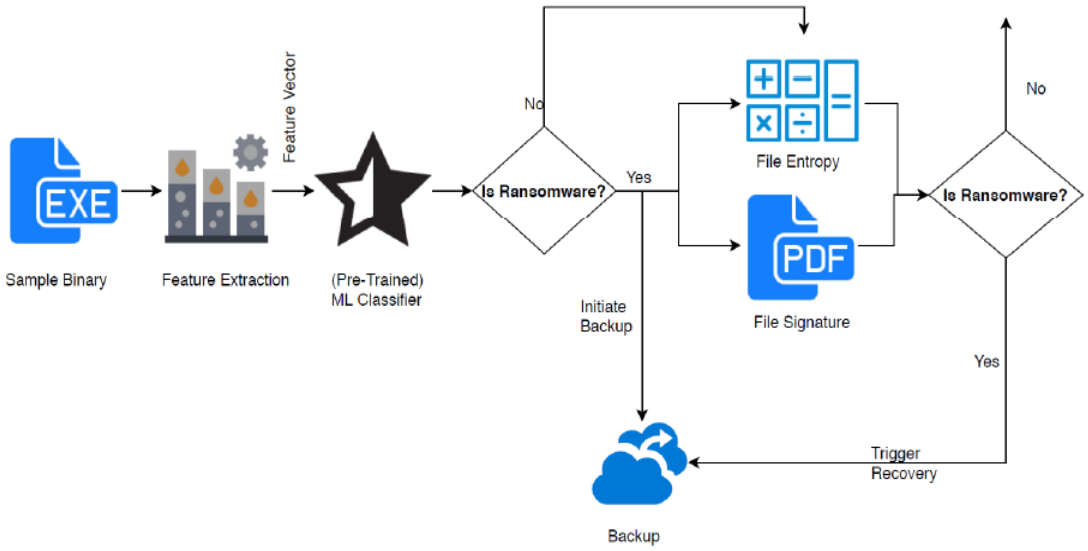

Figure 5.16 *Multilayer detection process*

To evaluate the performance of the file entropy/signature detector, our second detector, we used the same ransomware and benign samples which were used to evaluate the ML detector. The test dataset used to evaluate the regularized logistic regression model contained 155 samples, out of which 15 were benign and 140 were ransomware. For all 140 ransomware samples, we filtered

out the files which were missing file signatures and calculated the average entropy of the files per ransomware sample. On the one hand, the average entropy of the files, after the execution of the 134 ransomware samples, were higher than seven. On the other hand, for all the benign samples, the average entropy remained below seven. Under the above scenario, the new confusion matrix and classification performances achieved after passing the samples through our second detector are shown in Figure 5.17 and Table 5.10, respectively.

	precision (%)	recall (%)	f1-score (%)	support
Benign	71	100	83	15
Ransomware	100	96	98	140
micro avg	96	96	96	155
macro avg	86	98	91	155
weighted avg	97	96	96	155

Table 5.10 *Classification report for the file entropy/signature detector.*

We can see from the confusion matrix that we were successfully able to reduce the false positives by using our two detectors in tandem. The second detector successfully reclassified two ransomware samples that were falsely classified as benign by the ML detector. However, the second detector also increased false negatives by classifying six ransomware samples as benign. Those six-ransomware belonged to the teslacrypt, crysis and striked ransomware families. The above scenario makes detection more critical as ransomware samples were identified as benign. Upon checking the files, it was observed that these ransomware samples did not start any encryption process yet. As a result, the average entropy stayed below the threshold. In this case,

when the first detector classifies a binary as a ransomware while the second detector classifies it as legitimate, our system will still raise a warning by notifying about the suspicious binary.

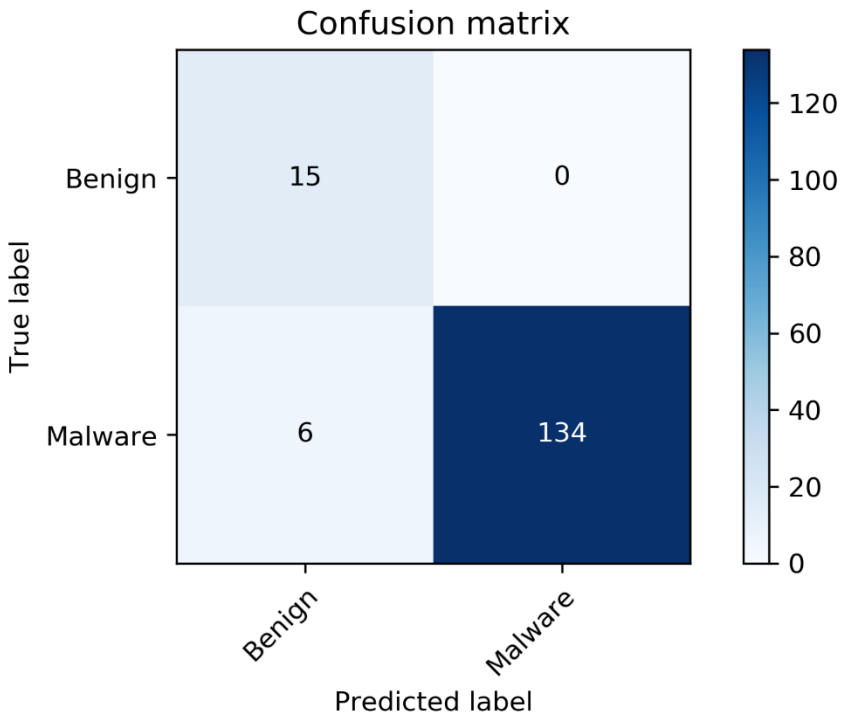

Figure 5.17 *Confusion matrix for the file entropy/signature detector*

5.7 Summary

This chapter discusses data pre-processing, feature selection and machine learning classification with three linear classifiers. We compared the results of SVM, Linear Regression and Random Forest. We also did hyper-parameter tuning of all three classifiers and observed that regularized logistic regression achieved the best performance measures among all. We also did time analysis of encrypted files generated by ransomware infection. The series of files having high entropy values and missing file signatures can be helpful to detect the ransomware which has evaded the detection. Finally, we proposed a ransomware detection system by combining machine learning model with combined file entropy and file signature models. In the next chapter, we make concluding remarks and discuss our future work.

Chapter 6 : Conclusion

6.1 Contribution Summary

Over the years, researchers have proposed various approaches to detect ransomware. Unfortunately, major works done on dynamic ransomware detection are biased towards limited feature space (e.g., API calls). Therefore, there is a need to search for new features for ransomware detection. We started our research first by providing an overview of the ransomware phenomenon and discussing its impact on businesses, institutions, and individuals. We explored the behavior of ransomware in Windows environment by presenting a case study. Then we summarized and discussed related works done on static, dynamic, and hybrid ransomware detection.

To develop our detection model, we started by surveying behavior analysis reports of ransomware variants from the literature and industry. Then, we identified the behaviors of ransomware that can be converted to a feature set. We identified important features using statistical techniques.

As our main contribution, we introduced two new sets of features: grouped registry key operation, and combined file entropy and file signature. Finally, we trained selected features on machine learning classifier and compared the results of three linear classifiers, namely, SVM, Random Forest, and Logistic Regression.

Furthermore, we tuned hyper-parameters for each classifier to achieve the best performance measures. We noticed that regularized logistic regression achieved the best results among all classifiers used for the experiment. Since the number of ransomware samples in our dataset was relatively large, benign samples became a minority class. To avoid the imbalanced class problem, we also applied the SMOTE data oversampling technique to evaluate our classifier.

Through extensive experimentation, we can say that grouped registry key operations play an essential role with other features in ransomware detection and help the model to achieve a 100% detection rate with false positive rate of 1.4%. We also claim that the proposed method is capable of detecting all the pre-seen as well as novel ransomware. Furthermore, we proposed a detection architecture that helps differentiate user-triggered encryption from ransomware-triggered encryption, thereby allowing saving as many files as possible during an attack.

We argue that, although older detection systems have an excellent detection capability, our proposed system is capable of detecting the malware which was successful in evading antivirus program or firewall running on the system. To continuously detect the anomalous activities in the Windows operating systems, the combination of file signature and file entropy plays an important role in our proposed detection strategy. The results obtained in this work can help in the future development of better antivirus programs.

6.2 Perspectives and Future Work

Future work will consist of expanding our features space by closely monitoring the latest ransomware and exploring more compelling features. Future work will also include an extension of the current ISOT ransomware dataset. There are many possibilities to improve the final accuracy and false positive rate of our ransomware detection system.

One of the possibilities is combining our system with a static-based detection system and making the system hybrid. If code obfuscation is not done, the static-based detection system is adequate to detect previously seen ransomware samples. If ransomware is detected at the early stage through static-based detection, ransomware does not have to pass through the complex

process of feature extraction and machine learning classification. The results might also help to reduce the false positive rate and increase the detection accuracy of our system.

Additionally, there are several advantages of extending our current ISOT dataset. One of the advantages is to classify the ransomware based on family. One of the future scope of our work consists of covering all possible ransomware families and identifying the expected ransomware class based on execution behavior in the Windows operating system. We will be evaluating our system on large-scale data and experimenting with other machine learning algorithms, including deep learning.

Some ransomware like Cryptowall 4.0 and Petya encrypt the Master File Table (MFT) instead of encrypting the full disk. MFT stores the information required to retrieve files from an NTFS partition. In this case, the user data still will be unmodified, but the operating system doesn't know how all data fits together. Saving the data at this point is easy and there are high chances that the data can be recovered using data recovery techniques available in digital forensics. Ransomware can target the master file tables through windows command-line utility or by developing a software which use Windows API's to access the MFT. In this case, we are also planning to extend our feature set by adding all command-line operations and APIs to our feature set which can be used to access or modify the master file table. Our detection system will continuously monitor these operations on MFT and in case of multiple changes in a short time, the system will stop the malicious processes.

Our future work also includes designing the anomaly detection system using the neural network. The anomaly detection system is very much applicable to ransomware detection. Anomaly detection being the identifier of the data points which are different from the expected pattern of the groups, it is necessary to have a large number of benign samples in the dataset.

To more effectively classify the ransomware-triggered encryption, we are also planning to build a kernel-level driver who will monitor the file system changes at MFT (Master File Table) level where file-signature is missing. The driver will continuously detect the files which are deleted or overwritten. For the files, currently being modified, the driver will make a temporary copy and hold it into temporary storage for a predefined time T. The time T will be defined after the analysis of different ransomware families and the time taken by them to encrypt the user files. If ransomware has accessed a large number of files in a short time, the threshold of the files will be low. If ransomware encrypts the files and stops for a while, the value of the time parameter T can be defined as very large. At the end of the defined period T, the decision can be made to classify the binary as ransomware or benign. Also, the encrypted or deleted files can be restored from the backup.

Chapter 7 : References

[1] "Businesses Paid $301M to Ransomware Hackers Last Year, New Datto Study Finds," *datto*, 2017. [Online]. Available: https://www.datto.com/news/datto-releases-global-state-of-the-channel-ransomware-report. [Accessed: 05-Jul-2019].

[2] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9148, pp. 3–24, 2015.

[3] Richard Winton, "Hollywood hospital pays $17,000 in bitcoin to hackers; FBI investigating - Los Angeles Times," *Los Angeles Times*, 2016. [Online]. Available: https://www.latimes.com/business/technology/la-me-ln-hollywood-hospital-bitcoin-20160217-story.html. [Accessed: 06-Jul-2019].

[4] "University of Calgary makes significant progress to address systems issues | UToday | University of Calgary," *UToday*, 2016. [Online]. Available: https://www.ucalgary.ca/utoday/issue/2016-06-08/university-calgary-makes-significant-progress-address-systems-issues. [Accessed: 05-Jul-2019].

[5] R. Shinde, P. Van Der Veeken, S. Van Schooten, and J. Van Den Berg, "Ransomware: Studying transfer and mitigation," *Int. Conf. Comput. Anal. Secur. Trends, CAST 2016*, no. July 2018, pp. 90–95, 2017.

[6] "Ransomware on Steroids: Cryptowall 2.0 - Cisco Blog," 2015. [Online]. Available: https://blogs.cisco.com/security/talos/cryptowall-2. [Accessed: 05-Jul-2019].

[7] Vince Tabora, "Cryptography + Malware = Ransomware - By," *HACKERNOON*, 2018. [Online]. Available: https://hackernoon.com/cryptography-malware-ransomware-36a8ae9eb0b9. [Accessed: 02-Jul-2019].

[8] A. L. Timothy Gallo, "1. Introduction to Ransomware - Ransomware [Book]," *O'REILLY*. [Online]. Available: https://www.oreilly.com/library/view/ransomware/9781491967874/ch01.html. [Accessed: 05-Jul-2019].

[9] Keith Jarvis, "CryptoLocker Ransomware Threat Analysis | Secureworks," 2013. [Online]. Available: https://www.secureworks.com/research/cryptolocker-ransomware. [Accessed: 05-Jul-2019].

[10] Jérôme Segura, "Browser Ransomware hides behind CloudFlare, smartens payment system - Malwarebytes Labs | Malwarebytes Labs," *MalwarebytesLABS*, 2016. [Online]. Available: https://blog.malwarebytes.com/threat-analysis/2013/12/browser-ransomware-hides-behind-cloudflare-smartens-payment-system/. [Accessed: 05-Jul-2019].

[11] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," *Proc. - Int. Conf. Distrib. Comput. Syst.*, vol. 2016-Augus, pp. 303–312, 2016.

[12] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," pp. 38–49, 2002.

[13] J. Z. Kolter and M. A. Maloof, “Learning to Detect Malicious Executables,” *Mach. Learn. Data Min. Comput. Secur.*, vol. 1, no. 212, pp. 47–63, 2006.

[14] Q. Jerome, K. Allix, R. State, and T. Engel, “Using opcode-sequences to detect malicious Android applications,” *2014 IEEE Int. Conf. Commun. ICC 2014*, pp. 914–919, 2014.

[15] A. Moser, C. Kruegel, and E. Kirda, “Limits of Static Analysis for Malware Detection - IEEE Conference Publication.”

[16] M. Baig, P. Zavarsky, R. Ruhl, and D. Lindskog, “The Study of Evasion of Packed PE from Static Detection,” *World Congr. Internet Secur.*, no. iii, pp. 99–104, 2012.

[17] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, C. Mulliner, and W. Robertson, “UNVEIL : A Large-Scale , Automated Approach to Detecting Ransomware This paper is included in the Proceedings of the,” *Proc. 2014 VIRUS Bull. Conf.*, pp. 757–772, 2016.

[18] A. Continella *et al.*, “ShieldFS,” *Proc. 32nd Annu. Conf. Comput. Secur. Appl. - ACSAC ’16*, pp. 336–347, 2016.

[19] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, “Automated Dynamic Analysis of Ransomware: Benefits, Limitations and use for Detection,” 2016.

[20] Z.-G. Chen, H.-S. Kang, S.-N. Yin, and S.-R. Kim, “Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph,” pp. 196–201, 2017.

[21] A. Lanzi, D. Balzarotti, and C. Kruegel, “AccessMiner-UsingSystemCentricModelsForMalwareProtection.”

[22] R. Vinayakumar, K. P. Soman, K. K. S. Velan, and S. Ganorkar, “Evaluating shallow and deep networks for ransomware detection and classification,” *2017 Int. Conf. Adv. Comput. Commun. Informatics, ICACCI 2017*, vol. 2017-Janua, pp. 259–265, 2017.

[23] S. Poudyal, K. P. Subedi, and D. Dasgupta, “A Framework for Analyzing Ransomware using Machine Learning,” *Proc. 2018 IEEE Symp. Ser. Comput. Intell. SSCI 2018*, pp. 1692–1699, 2019.

[24] A. Karimi and M. H. Moattar, “Android ransomware detection using reduced opcode sequence and image similarity,” *2017 7th Int. Conf. Comput. Knowl. Eng. ICCKE 2017*, vol. 2017-Janua, no. Iccke, pp. 229–234, 2017.

[25] H. Bos, F. Monrose, and G. Blanc, “Research in attacks, intrusions, and defenses: 18th international symposium, RAID 2015 Kyoto, Japan, november 2-4, 2015 proceedings,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9404, pp. 382–404, 2015.

[26] M. Cantonment, “RansHunt_ A support vector machines based ransomware analysis framework with integrated feature set - IEEE Conference Publication.pdf,” pp. 22–24, 2017.

[27] “Analysis Results — Cuckoo Sandbox v2.0.6 Book,” *Cuckoo Foundation*. [Online]. Available: https://cuckoo.readthedocs.io/en/latest/usage/results/. [Accessed: 05-Jul-2019].

[28] N. Hampton, Z. Baig, and S. Zeadally, “Ransomware behavioural analysis on windows platforms,” *J. Inf. Secur. Appl.*, vol. 40, pp. 44–51, 2018.

[29] “Calculate File Entropy – Kenneth G. Hartman, CISSP.” [Online]. Available:

https://kennethghartman.com/calculate-file-entropy/. [Accessed: 24-Jun-2019].

[30] Gary C. Kessler, “File Signatures,” 2019. [Online]. Available: https://www.garykessler.net/library/file_sigs.html. [Accessed: 02-Jul-2019].

[31] Tim Fisher, “Windows Registry (What It Is and How to Use It),” *Lifewire*. [Online]. Available: https://www.lifewire.com/windows-registry-2625992. [Accessed: 08-Jul-2019].

[32] K. Yeager, “LibGuides: SPSS Tutorials: Pearson Correlation.”

[33] J. Singh Malik, P. Goyal, and M. K. Sharma, “A Comprehensive Approach Towards Data Preprocessing Techniques & Association Rules,” *Bharati Vidyapeeth’s Inst. Comput. Appl. Manag.*, p. 12, 2007.

[34] “5.3. Preprocessing data — scikit-learn 0.21.2 documentation.” [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html. [Accessed: 24-Jun-2019].

[35] “An Introduction to Feature Selection.” [Online]. Available: https://machinelearningmastery.com/an-introduction-to-feature-selection/. [Accessed: 02-Jul-2019].

[36] Ritchie Ng, “Logistic Regression | Machine Learning, Deep Learning, and Computer Vision,” 2019. [Online]. Available: https://www.ritchieng.com/logistic-regression/. [Accessed: 15-Jul-2019].

[37] “scikit-learn: Logistic Regression, Overfitting & regularization - 2018.” [Online]. Available: https://www.bogotobogo.com/python/scikit-learn/scikit-learn_logistic_regression.php. [Accessed: 25-Jun-2019].