AUTOMATED DETECTION AND CLASSIFICATION OF MALWARE USED IN TARGETED ATTACKS VIA MACHINE LEARNING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Yakup Korkmaz

September, 2015

AUTOMATED DETECTION AND CLASSIFICATION OF MALWARE USED IN TARGETED ATTACKS VIA MACHINE LEARNING

By Yakup Korkmaz

September, 2015

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. İbrahim Körpeoğlu(Advisor)

Prof. Dr. Özgür Ulusoy

Prof. Dr. Ali Aydın Selçuk

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

AUTOMATED DETECTION AND CLASSIFICATION OF MALWARE USED IN TARGETED ATTACKS VIA MACHINE LEARNING

Yakup Korkmaz
M.S. in Computer Engineering
Advisor: Assoc. Prof. Dr. İbrahim Körpeoğlu
September, 2015

Targeted attacks pose a great threat to governments and commercial entities. Increasing number of targeted attacks, especially Advanced Persistent Threats, are being discovered and exposed in each year by various cyber security organizations. Key characteristics of these attacks are well-funded and skilled actors persistently targeting specific entities, sophisticated tools and tactics, long-time presence in breached environments before detection and stealth operation. Malware plays a crucial role in a targeted attack for various tasks such as compromising systems, maintaining presence, communicating with the operators, carrying out commands, etc. Because of its stealthy nature, malware used in targeted attacks is expected to act different than the traditional malware when it is dynamically analyzed in a sandbox environment.

In this thesis we focused on the malware used in targeted attacks and present a method to automatically detect and classify targeted malware through machine learning using behavioral and memory features. Its worth noting that it is a first work published in the literature that classifies targeted malware and incorporates memory features into the dynamic features. The method comprises the steps of running both traditional and targeted malware in a dynamic analysis system along with a memory analysis tool, extracting features from behavioral and memory artifacts found in analysis results and employing machine learning on the extracted features. New behavioral and memory features were defined in order to classify targeted malware more effectively. Method is then evaluated over a dataset comprised of targeted and traditional malware with different supervised learning algorithms. The results show that machine learning can be employed successfully to automatically detect and classify targeted malware from dynamic analysis results using behavioral and memory features.

Keywords: Targeted attacks, Advanced Persistent Threats, dynamic analysis, memory analysis, dynamic features, targeted malware classification.

ÖZET

HEDEFLİ SALDIRILARDA KULLANILAN ZARARLI YAZILIMLARIN MAKİNE ÖĞRENİMİ KULLANILARAK TESPİTİ VE SINIFLANDIRILMASI

Yakup Korkmaz
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Doç. Dr. İbrahim Körpeoğlu
Eylül, 2015

Hedefli saldırılar devlet ve ticari kurumlar için büyük bir tehdit oluşturmaktadır. Her yıl giderek artan sayıda hedefli saldırı, özellikle Gelişmiş Sürekli Tehditler, çeşitli siber güvenlik firmaları tarafından tespit edilerek ortaya çıkarılmaktadır. Bu saldırıların temel özellikleri, iyi finanse edilen ve yetenekli aktörlerin devamlı olarak belirli kurumları hedef alması, karmaşık araç ve tekniklerin kullanımı, tespit edilene kadar sızılan ortamlarda uzun süreli kalınması ve gizli faaliyet yürütülmesidir. Zararlı yazılımlar hedefli saldırılarda sistemlerin ele geçirilmesi, kalıcılığın sağlanması, aktörlerle haberleşme, komutların yerine getirilmesi gibi çeşitli görevlerde çok hayati bir rol oynar. Gizli çalışma doğası gereği hedefli saldırılarda kullanılan zararlı yazılımların dinamik olarak kontrollü sanal bir ortamda incelendiğinde geleneksel zararlı yazılımlara göre farklı davranması beklenir.

Bu tezde hedefli saldırılarda kullanılan zararlı yazılımlara odaklandık ve hedefli zararlı yazılımları davranışsal ve hafıza öznitelikleri kullanarak makine öğrenimi yoluyla tespit eden ve sınıflandıran bir yöntem sunduk. Çalışmanın hedefli zararlı yazılımların sınıflandırıldığı ve davranışsal özniteliklere hafıza özniteliklerinin eklendiği literatürdeki ilk çalışma olduğunu belirtmek gerekir. Sunulan yöntem, geleneksel ve hedefli zararlı yazılımların dinamik analiz sisteminde hafıza analizi aracıyla birlikte çalıştırılması, analiz raporlarında bulunan davranışsal ve hafıza izlerinden ayırt edici öznitelikler çıkartılması ve çıkartılan öznitelikler üzerinde makine öğrenmesi uygulanması adımlarını içermektedir. Hedefli zararlı yazılımları daha etkili sınıflandırmak üzere yeni davranışsal ve hafıza öznitelikleri tanımlandı. Yöntem, hedefli ve geleneksel zararlı yazılımlardan oluşan bir veri kümesi üzerinde farklı gözetimli öğrenme algoritmaları ile test edilerek

değerlendirildi. Elde edilen sonuçlar, dinamik analiz sonuçlarından davranışsal ve hafıza öznitelikleri çıkartılıp makine öğrenimi kullanılarak hedefli zararlı yazılımların başarılı bir şekilde tespit edilip sınıflandırılabileceğini gösterdi.

Anahtar sözcükler: Hedefli saldırılar, Gelişmiş Sürekli Tehditler, dinamik analiz, hafıza analizi, dinamik öznitelikler, hedefli zararlı yazılım sınıflandırması.

Acknowledgement

Foremost, I would like to express my sincere gratitude to my supervisor, Assoc. Prof. Dr. İbrahim Körpeoğlu, for his support and encouragement during my master study. I would like to thank especially for his patience, understanding and assistance in the thesis process.

I would like to thank Dr. Leyla Bilge for her assistance and guidance in this thesis and also for sharing her expertise with me.

My sincere thanks also goes to Prof. Dr. Ali Aydın Selçuk for his support in accelerating and finishing this thesis work.

I would like to thank the rest of my thesis committee: Prof. Dr. Özgür Ulusoy and Prof. Dr. Ali Aydın Selçuk. They were very kind to accept being a jury member and spend their time for evaluating my thesis despite their busy schedule.

Words cannot express how I am grateful to my family, my mother and father, for their sincere and endless support at every moment throughout my life and for providing me every opportunity they can by sacrificing from themselves, especially for my education.

Last but not least, I would like to express my gratitude to my beloved and precious wife Kıymet, who spent sleepless nights with me, sacrificing herself and providing me every support she can during my thesis work. This thesis would not be possible without her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last couple of years, cyber threat landscape has changed dramatically and shifted from financially motivated attacks to targeted attacks, especially Advanced Persistent Threats. Starting from the year 2010 with Operation Aurora, we have witnessed increasing number of such targeted attacks including Stuxnet, Duqu, Flame, Red October, Snake, etc. [1]. This new class of attacks become top priority cyber risks for governments and commercial entities because of its sophistication in terms of tools and techniques employed and well-funded and skilled threat actors. In targeted attacks, well-organized operators targets specific entities persistently with high motivation, evades security defenses in place, employs advanced tools and tactics, maintains long-time presence in target environment and operates slow and stealthy to avoid detection [2].

Malware plays a vital role in success of a targeted attack and is employed almost in every phase of an attack lifecycle until the operators goal is achieved. It carries out wide range of tasks including compromising systems, escalating privileges, maintaining presence, exfiltrating data, communicating with the operators over command and control servers, carrying out commands, etc. Even though these tasks are not peculiar to targeted malware only and also traditional malware could carry out most of these tasks throughout its execution, targeted malware is still expected to act different than the traditional malware because of

its stealthy nature.

Easy but also very efficient way to reveal malwares behavior once it infects a system is running it in a controlled environment and capture all the changes on the system and network during the analysis process. This analysis method is called dynamic analysis and quick insight into malware can be gained by running it in a dynamic analysis sandbox for a very short time (usually 3 minutes). However, it does not work well for all kinds of malware, because there are some malwares that could detect it is running in a sandbox and stop or delay its execution, or could not complete all its tasks within the analysis period, or try to deceive analyst by doing nothing malicious or suspicious. Even with its all limitations dynamic analysis is extensively used in malware analysis field because huge number of new malwares is discovered every day and they are needed to be analyzed in a fast and automated manner. In our thesis work, we used a modified version of a popular open-source automated malware analysis system, Cuckoo Sandbox, for capturing malware behavior [3].

As sophistication level of targeted attacks rises gradually in time, increasing number of targeted attacks are being discovered using malware with advanced stealth techniques or even non-persistent malware that only resides in the memory [4]. Recently cyber security professionals come up with a new term for targeted attacks employing non-persistent in-memory malware and named them as Advanced Volatile Attacks, because there is no easy way to detect them other than analyzing the volatile memory. In order to detect non-persistent malware and also malware that hides its presence on the system by using advanced malware stealth techniques such as hooking, injection, hollowing, etc., memory analysis is a must and needed to be conducted along with dynamic analysis. In this thesis, we used popular open-source memory forensics tool called The Volatility Framework [5], which offers wide range of plugins for various memory related analysis tasks. Cuckoo sandbox is configured to dump full memory of the guest analysis system just before the execution ends and run Volatility with selected plugins on the memory dump after the execution.

The goal of this thesis is to utilize machine learning to detect and classify

targeted malware using behavioral and memory features extracted from dynamic and memory analysis results. Previous works in this area focuses on either classifying binaries as benign and malicious or classifying malware into certain malware families. Instead of classifying malware into families, it is possible to distinguish targeted malware from traditional malware and all of its families via machine learning because of the characteristics of targeted attacks and the way they are operated by the threat actors.

In summary, this thesis makes the following contributions:

- We present a method to automatically detect and classify targeted malware through machine learning using behavioral and memory features.
- We provide detailed description of the analysis system, method steps, features extracted and machine learning algorithms.
- We present novel set of memory features extracted from memory analysis and also new dynamic features. We provide detailed list and description of all dynamic and memory features.
- We provide a comprehensive evaluation of the proposed method showing that our targeted malware classification method achieves high accuracy and low false positive rate.

The thesis is structured as follows: Chapter 2 presents the related work and foundations of the malware classification problem in terms of features used. Chapter 3 details the system and methodology for extracting dynamic and memory features from the analysis results to perform machine learning. Chapter 4 evaluates experiment results after explaining the machine learning algorithms and evaluation metrics used. Finally Chapter 5 concludes thesis.

Chapter 2

Background

Finding a suitable way to represent binary files lies at the heart of every method or system presented so far to detect or classify malware via machine learning. In this section a literature background about malware classification is provided with a focus on the use of static and dynamic features in malware classification.

2.1 Static Feature Extraction

Static analysis is the process of examining a binary file without actually executing it in order to determine if it is malicious or not. Extracting static features from a binary file to perform malware classification is carried out by static analysis tools or techniques.

[6] were among the first who introduced static features for malware detection by employing several different classifiers. They conducted experiments based on three different types of static features including Portable Executable (PE), strings and byte sequence n-grams. From PE header of a binary, they extracted dynamic link library (DLL) information and constructed three different feature vectors representing if a DLL was used, if a specific function inside a DLL was called and count of unique function calls inside each DLL. Encoded strings inside

a binary was extracted and used as a feature. They also converted binary files into hexadecimal codes and used byte sequences of codes as n-gram features. Based on these three features, different classifiers are employed to classify new binaries as malicious or clean. While string features yielded the highest accuracy, best detection rate was achieved by using byte sequence n-grams.

Their work inspired and encouraged others to try similar approaches for malware classification. [7] adopted and enhanced the byte sequence n-grams technique. They achieved better results in detecting malicious binaries via classifiers including Support Vector Machine, Decision Tree and boosted versions of them. [8] extracted API calls from PE header of a binary file similar to what [6] did in their work and used them as features for classifying a binary as malicious or not.

More advanced static analysis and reverse engineering task to extract features from binaries was carried out in [9] compared to other works mentioned here. Binary was first disassembled and then length and frequency of function names were extracted. Based on the function name length features, they perform malware classification between different malware families and their results suggested that function name length is significant as a feature in distinguishing malware families.

While static features widely-used to detect or classify malware via machine learning, there exist some limitations. Authors assume that the malware is unpacked or not encrypted and static features can be extracted right away from the binary. However, it is very common for a malware to be packed or encrypted and in some cases it is not possible to fully unpack or decrypt malware. There also exist wide variety of obfuscation techniques that can thwart the whole process [10].

2.2 Dynamic Feature Extraction

Dynamic analysis is the process of examining a binary file by executing it in a controlled environment and capturing its behavior in order to determine if it is malicious or not. Controlled environment where the analysis is conducted could be a specially designed sandbox [3, 11, 12] offering virtual, emulated or even bare-metal environment, but also a single computer equipped with dynamic analysis tools.

[13] constructed a high level behavior profile consisting of system change counts for process, file, registry and network categories for each binary sample after running it in a virtual environment and collecting system events. Using the behavioral profiles representing malware behavior as features and Normalized Compression Distance (NCD) as distance metric, they conducted hierarchical clustering on malware samples to cluster them into families. [14] extended the previous work [13] by constructing a low level behavior profile using a generalized form of system resources and system calls after running the sample in a dynamic analysis sandbox [12]. They achieved high run-time performance by performing hierarchical clustering using Locality Sensitive Hashing (LSH) and Jaccard index as distance metric. While both of the previous works used an unsupervised algorithm for the malware classification, [15] performed malware classification using Support Vector Machines (SVM) by extracting features from analysis report generated after running a malware sample in a dynamic analysis system [11]. Feature vectors in the works [14, 15] were representing malware behavior using a generalized form of the system calls and their parameters captured during the analysis.

[16] executed binaries in a virtual environment and captured system calls and their parameters with the help of an automated tool. Each captured system call and parameter treated as separate string and a global list comprised of all strings observed throughout the work is compiled. Feature vector for each binary was constructed using this list and consisted of Boolean values for each string in the global list specifying whether it was encountered during the analysis of the sample or not. They used supervised learning classifiers both for classifying a binary as

malicious or not and for classifying malware into malware families. [17] extended the previous work by incorporating static features including printable strings and function length frequency into the dynamic system call features. Function length frequency feature vector consists of counts of functions for fixed length ranges. They conducted experiments for each feature separately and also for the integrated features using different supervised learning classifiers. All experimented classifiers achieved best results using the integrated features. [18] presented a method for combining features from six different sources including three static sources, two dynamic sources and one source containing statistics about the other sources. They achieved high accuracy classifying binaries as malicious or not using SVM classifier.

[19] proposed a system consisting of two components, one for analyzing binaries dynamically and one for classification and clustering. 65 features in three categories including file, registry and network were extracted from the artifacts captured during the dynamic analysis. Extracted features were in low granularity, mostly counts, e.g. number of files created, unique number of extensions of the created files, file size, etc. Memory artifacts were also collected during the analysis phase but used only for signature matching to enhanced labeling, but not included in the feature vector. Experiments conducted using wide range of supervised learning classifiers in order to classify binaries as malicious or not and achieved high accuracy. They also performed clustering using several different distance metrics and parameters and presented results in terms of accuracy and performance.

There is also considerable amount of work that utilizes DNS features for classification. [24] identified malicious domains by monitoring the DNS traffic passively, extracted 15 features from the monitored traffic and achieved high detection rate. Types of features defined in terms of granularity are similar to the features presented in this thesis.

In this thesis, we defined new features as well as features that were already explored by others in the literature to tackle the malware classification problem.

Because we used different types of dynamic features together with memory features, there is no single work that encompasses all of them. Therefore, closest work in the literature in terms of features utilized for classification are [16] and [19].

Chapter 3

Methodology

In this section we provide detailed information about the proposed method. Section 3.1 describes design of the system in which analysis and classification tasks are being carried out. Section 3.2 provides information about the features extracted in detail. Section 3.3 presents the steps of the proposed method and targeted malware classification process.

3.1 System Overview

Malware classification process basically consists of several tasks including dynamically analyzing binaries, storing the analysis results, extracting features and performing malware classification using machine learning classifiers. There already exits widely-used systems, frameworks or tools for some specific tasks but for custom needs the best way is to implement manually by writing a program, script or even a small piece of code. Therefore an integrated system with all the required components is needed to be implemented, set up and configured properly.

Figure 3.1 illustrates the architecture of the system we devised and used for

targeted malware classification throughout the thesis work. The system is composed of the following components:

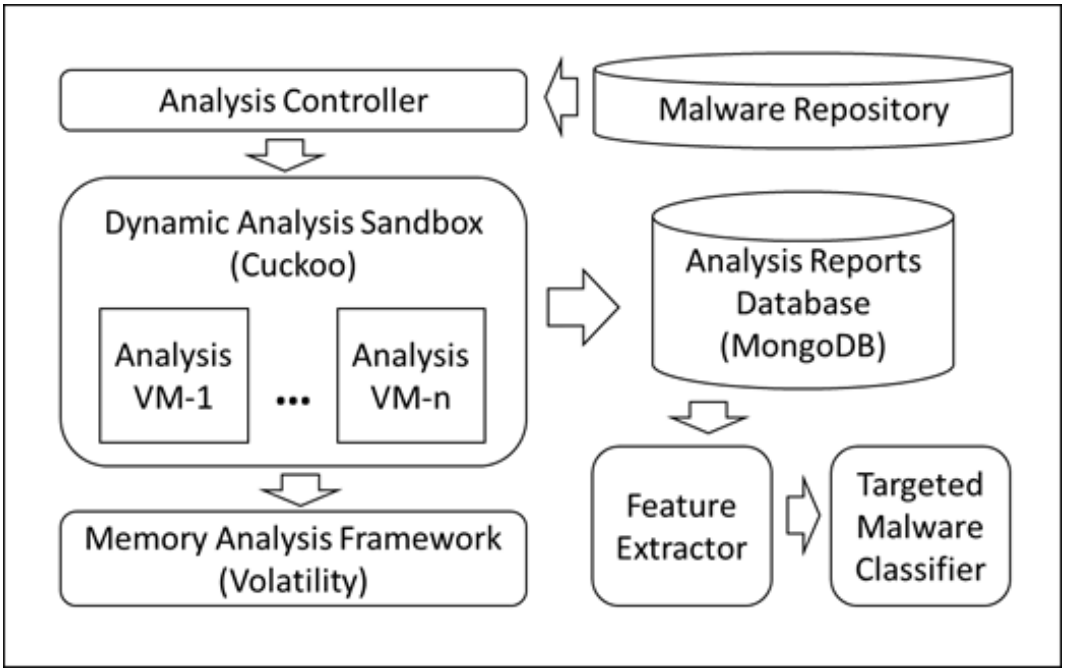

Figure 3.1: System Overview.

Malware repository:

Malware repository stores the malware data set, including targeted and traditional malware samples.

Analysis controller:

Analysis controller fetches malware from the malware repository and submits to the dynamic analysis sandbox. It checks the availability of the sandbox and controls the running analysis tasks. It is written in Linux shell script.

Dynamic analysis sandbox:

Dynamic analysis sandbox analyzes malware samples dynamically and utilizes modified version of Cuckoo Sandbox [3]. It leverages memory analysis framework for conducting memory analysis and sends the analysis reports to the analysis

reports database. Analyses are carried out in the virtual machines running Windows XP service pack 3 with Microsoft Office 2007 installed. An example analysis report in JavaScript Object Notation (JSON) format is provided in Figure 3.2.

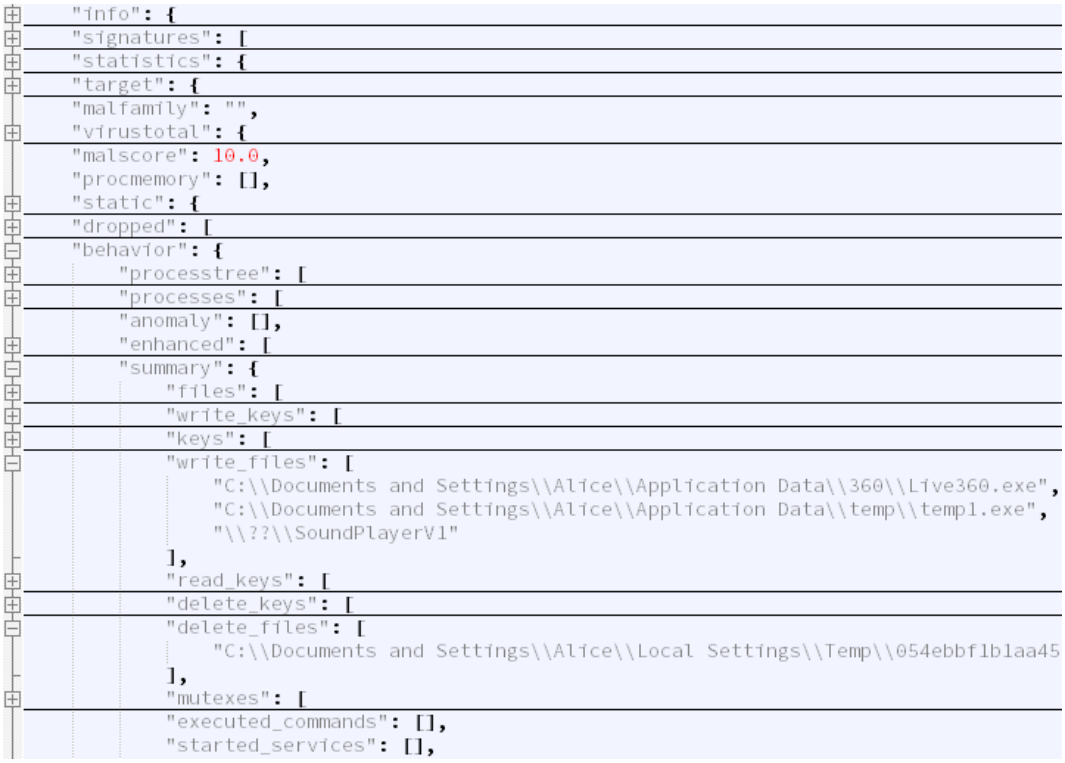

Figure 3.2: Example analysis report in JSON format.

Memory analysis framework:

Memory analysis framework offers wide range of plugins for memory analysis and utilizes Volatility Framework [6]. It is integrated with the dynamic analysis system.

Analysis reports database:

Analysis reports database stores the detailed analysis reports as JavaScript Object Notation (JSON) documents and utilizes MongoDB NoSQL database.

Feature extractor:

Feature extractor retrieves analysis report of each sample from database, extracts features from artifacts contained in the report and constructs the corresponding feature vector. It is implemented in Python.

Targeted malware classifier:

Targeted malware classifier employs supervised learning algorithms on the feature vectors provided by feature extractor. It is written in Python using machine learning library scikit-learn [20].

3.2 Feature Extraction

Wide-range of system level artifacts are captured during the whole analysis process comprised of dynamic and memory analyses and contained in the analysis reports.

We examined the analysis reports in detail to identify features for allowing a classifier to be able to distinguish between targeted and traditional malware. Process of identifying new features is guided by known malware behavior and our expertise on targeted attacks. We defined a novel set of memory features that will help identifying targeted malwares more efficiently. We also made use of several dynamic features already utilized in malware classification works published in the literature. Instead of utilizing high granularity features that represents low level atomic operations [14, 15], we decided to utilize low granularity and mostly quantitative features that could represent malware behavior from a wider aspect [19].

We identified a total of 764 features either by defining new dynamic and memory features or adopting the existing ones [16, 17, 18, 19]. These features are divided into six categories (see Table 3.1) and detailed information about each category is provided in the following sections.

Table 3.1: Feature categories.

Feature Category	Number of Features	Vector Index Range
Network features	27	0 - 26
File system features	174	27- 200
Registry features	4	201- 204
System call features	289	205 - 493
Miscellaneous features	219	494 - 712
Memory features	51	713 - 763
Total	764	764

3.2.1 Network Features

The utilized dynamic analysis sandbox, Cuckoo Sandbox, captures all packets over the network for the analysis period and stores in a pcap file. After the analysis period is completed, another task is initiated to extract network artifacts from the pcap file and found network artifacts are specified in the analysis report. Network features were then extracted from network related artifacts contained in the analysis report and therefore limited to the capability of the dynamic analysis sandbox in terms of network traffic analysis.

Cuckoo sandbox is capable of capturing, analyzing and reporting network protocols and information including, UDP, IRC, HTTP, SMTP, TCP, ICMP, DNS connections, hosts contacted and domain names resolved. From these network artifacts we extracted a total of 27 network features (see Table 3.2).

Most of the features are counts and the name of each feature in Table 3.2 provides sufficient information about the feature itself. However, we also defined new features and contained them in feature vectors including DNS request type frequency, domain level frequency and domain name length frequency, which require further information to be provided. Further information are provided only about these special types of features in this section and likewise in the following sections.

In the course of analyzing all samples, we encountered only the following DNS

Table 3.2: Network features.

#	**Feature Name**
1	Number of UDP connections
2	Number of distinct IP addresses
3	Number of distinct UDP destination ports
4	Number of IRC connections
5	Number of HTTP connections
6	Number of SMTP connections
7	Number of TCP connections
8	Number of ICMP connections
9	Number of hosts
10	Number of hosts with reverse DNS
11	Number of DNS requests
12 - 17	DNS request type frequency *
18	Number of domains
19 - 22	Domain level frequency **
23 - 27	Domain name length frequency (***)

requests types; A, AAAA, MX, SRV, TXT, PTR. Therefore we defined a new feature vector, DNS requests type frequency (*), composed of six features, between 12 and 17 in Table 3.2, each corresponding to the specific DNS request type and representing its count.

Domain name is a hierarchical series of strings separated by a dot, each dot specifying a level. A legitimate domain name consists of at least two levels, top-level and second-level domains, e.g., domain.com, but it can contain more levels with every dot following the second-level domain, each one is another subdomain in this case, e.g., anothersubdomain.subdomain.domain.com. Thus, we defined a feature vector, domain level frequency (**), composed of four features, between 19 and 22 in Table 3.2, each representing the count of domains for the following levels:

- second-domain + top-domain,
- one level subdomain + second-domain + top-domain,
- two level subdomains + second-domain + top-domain,

- three or more level subdomains + second-domain + top-domain.

For example, consider a malware trying to connect three domain names including, domain.com, sub1.domain.com and sub3.sub2.domain.com. Domain level frequency vector for the malware would be 1, 1, 1, 0.

Domain names are varying extremely in length and length alone is not very characterizing. To make it more characterizing, first we picked the following four length ranges for domain name lengths; 0-10, 11-16, 17-20, 20-32 and 32-. Then we defined a feature vector, domain length frequency (***), composed of five features, between 23 and 27 in Table 3.2, each corresponding to the specific length range and representing the count of domains in that range.

While we encountered plenty of artifacts related to UDP traffic, we simply observed no HTTP traffic in the analysis reports. Therefore we defined more features regarding the UDP traffic than HTTP, as it can also be seen in Table 3.2. It could be due to misconfiguration of the sandbox or the system but without detailed HTTP features, we managed to extract enough features regarding the network activity.

3.2.2 File System Features

The utilized dynamic analysis sandbox, Cuckoo Sandbox, is employing several injection and hooking techniques while executing the malware in order to detect and capture the file system operations including, file access, file read, file create, file modify and file delete. From these file system artifacts we extracted total of 174 file system features (see Table 3.3).

We came across a wide range of file extensions of the created, modified or deleted files throughout the analysis process. In order to characterize this behavior, we defined a feature vector, top extension frequency, composed of top 34 extensions varying from “dll”, “exe”, “jpg” to “tmp”, “txt” each corresponding to the specific extension and representing its count (see appendix for full list of

top extensions).

Once malware executes, it creates files under some specific path or location in the file system. This path is usually associated with an environment variable and named as common folder or known folder in Windows operating system [21]. For example, the following paths are well-known paths and also widely used by malware for copying itself under them;

```
C:\Windows\System32 (%SYSTEM%)

C:\DOCUME~1\ <user name>\LOCALS~1\Temp (%TEMP%)

C:\Documents and Settings\All Users (%ALLUSERSPROFILE%)
```

Because path to the file in terms of known folders in Windows environment can be a good indicator for malware behavior, we defined a feature vector, known path frequency, composed of 14 known paths each corresponding to the specific known path and representing its count (see appendix for full list of known paths).

In the course of our analyses, we observed files created in varying sizes from bytes to megabytes. After examining the file sizes in more detail, we decided not to utilize size as a whole but in ranges and we picked the following four size ranges in bytes; 0-64, 65-4096, 4097-262144 and 262144-. We defined a feature vector, file size frequency, composed of four features, each corresponding to the specific size range and representing the count of files in that range.

Another important issue about files are their types and file type could be a good indicator for characterizing malware behavior. Therefore we decided to utilize the types file, directory, driver, pipe, and alternate stream as features for read, accessed and modified files. Thus, we defined a feature vector, file type frequency, composed of five features, each corresponding to the specific file type and representing the count of that file type.

Table 3.3: File system features.

#	**Feature Name**
1	Number of files deleted
2	Number of files modified
3	Number of files deleted in distinct paths
4	Number of files deleted with distinct extensions
5-38	Top extension frequency of deleted files
39-52	Known path frequency of deleted files
53	Number of files modified in distinct paths
54	Number of files modified with distinct extensions
55-88	Top extension frequency of modified files
89-102	Known path frequency of modified files
103-107	File type frequency of modified files
108	Number of files created
109	Number of files created with distinct extension
110	Number of files created in distinct path
111-114	File size frequency
115-148	Top extension frequency of created files
149-162	Known path frequency of created files
163	Number of files read
164-168	File type frequency of read files
169	Number of files accessed
170-174	File type frequency of accessed files

Table 3.4: Registry features.

#	**Feature Name**
1	Number of registry keys deleted
2	Number of registry keys modified
3	Number of registry keys read
4	Number of registry keys accessed

Because during its execution a malware sample reads and accesses extensive number of files which are mostly duplicate or noise because of nested path traversal, we process and filter them before computing their counts. There are five types of file artifacts in the analysis results in terms of access, read, modified, create and delete operations. However, the defined frequency vectors are utilized not for all of them, but only for the ones where it is meaningful to utilize them.

3.2.3 Registry Features

The utilized dynamic analysis sandbox, Cuckoo Sandbox, is capable of detecting and capturing the registry operations including, registry key access, registry key read, registry key modify and registry key delete. From these registry key artifacts we extracted total of 4 registry features (see Table 3.4).

Because during its execution a malware sample reads and accesses extensive number of registry keys which are mostly duplicate or noise because of nested registry traversal, we process and filter them before computing their counts.

3.2.4 System Call Features

Our dynamic analysis toolbox, Cuckoo Sandbox, is capable of capturing system or API call traces inside each process and classify them into several categories according to the type of operation carried out, including process, filesystem, browser, crypto, etc. It also provides the name of API file from which the

Table 3.5: System call features.

#	**Feature Name**
1	Number of running processes
2-17	System call category frequency
18-33	System call category percentage frequency
34-289	API function frequency

API call is initiated. From these system call artifacts we extracted a total of 289 system call features (see Table 3.5)

In most cases a malware sample initiates huge number of system calls, but system calls count alone is not very meaningful and not characterizing its behavior properly. However, category of a system call in terms of type of operation carried out is a better indicator to understand the actions taken by the malware. By examining category values specified for each system call in all analysis reports, we identified 16 different category labels tagged by Cuckoo Sandbox (see appendix for full list of system call categories). Thus, we defined a feature vector, system call category frequency, that is composed of 16 features, between 12 and 17 in Table 3.5, each corresponding to a specific system call category and representing the count of the system calls initiated in that category.

Although tracking counts of system calls in individual categories can provide good insights about the malware behavior, some categories have disproportionally high counts of system calls. To capture malware actions at a higher level, we defined a new feature vector that identifies proportion of number of system calls in each individual category to the total number of system calls in all categories. The feature vector, system call category percentage frequency, is composed of 16 features, between 18 and 33 in Table 3.5, each corresponding to a specific system call category and representing the percentage of the system calls count in that category to total system calls count in all categories.

Most of the time, system calls are not directly accessed by programs but via a high-level Application Program Interface (API), e.g., Windows API, which invokes the real system call itself. API is implemented by various API functions

Table 3.6: Miscellaneous features.

#	**Feature Name**
1	Number of resolved API functions
2	Number of distinct API files
3-101	Resolved top API function frequency
102	Number of commands executed
103	Average number of command items
104	Number of services created
105	Number of services started
106	Number of mutex created
107	Average mutex name length
108	Number of signatures matched
109-219	Matched signature frequency

contained in several API library files, typically Dynamic-Link Library (DLL) files. By examining system call traces captured in all analysis reports, we identified 256 different API functions called by malware samples (see appendix for full list of API functions). Therefore, we defined a feature vector, API function frequency, composed of 256 features, between 34 and 289 in Table 3.5, each corresponding to the specific API function and representing the call count of that API function.

3.2.5 Miscellaneous Features

Cuckoo Sandbox is a very powerful and highly customizable open-source dynamic analysis sandbox backed by a large community of researchers and developers. With the active support of its community, it offers wide range of functionality in malware analysis area. Apart from the network, file system, registry and system call artifacts, it generates analysis reports enriched with several other malware related artifacts including API functions resolved, mutexes created, services started, commands executed, etc. From these different types of artifacts we extracted a total of 219 miscellaneous features (see Table 3.6).

Cuckoo Sandbox is a dynamic analysis sandbox, but along with dynamic artifacts, it also contains some static ones in analysis reports including strings,

resolved API files and functions, etc. In order to make use of artifacts related to resolved API functions, we examined resolved API files in all analysis reports and selected top 99 API files to utilize as a feature (see appendix for full list of top API files). We defined a feature vector, resolved top API function frequency, composed of 99 features, between 3 and 101 in Table 3.6, each corresponding to the specific API file and representing the count of resolved functions within that API file.

Many signatures are created for identifying specific type of malware behavior by the Cuckoo community using YARA tool [22] and made available for public use. These signatures provides wide-range of behavioral characteristics when utilized as a feature. By examining matched signatures in all analysis reports, we identified total of 111 different signatures that matched at least once for a malware sample (see appendix for full list of Cuckoo community signatures). We defined a feature vector, matched signature frequency, composed of 111 features, between 109 and 219 in Table 3.6, each corresponding to the specific signature and representing the occurrence of that signature by noting 1, if matched and 0, if not.

3.2.6 Memory Features

As a memory analysis framework, we utilized Volatility Framework [5], well accepted widely used open-source memory analysis framework offering extensive number of tools and plugins for different types of memory analyses. Similar to Cuckoo Sandbox, Volatility Framework has a large user base and very active community that continuously helps extending its functionality by contributing to the core development or by writing custom user plugins for memory-related analyses. Another important fact is that Cuckoo Sandbox offers integration with Volatility and can initiate memory analysis task with selected number of plugins right after the dynamic analysis task is finished. Results of the executed Volatility plugins are contained in the same analysis report with the Cuckoo Sandbox.

Because recent targeted attacks employ increasing number of obfuscation and

Table 3.7: Memory features.

#	**Feature Name**	**Volatility Plugin**
1	Number of mutexes	Mutantscan
2-5	Mutex length frequency	Mutantscan
6	Number of processes exited	Pslist
7	Number of processes running	Psxview
8-14	Process list frequency	Psxview
15-21	Module in common Windows process frequency	Dlllist
22-28	Avg. module per common Windows proc. Frequency	Dlllist
29-31	Injection VAD tag frequency	Malfind
32	Number of injections	Malfind
33	Average number of injections per process	Malfind
34	Number of processes with privileges	Privs
35	Number of processes with Administrator SID	Getsids
36-38	Hidden DLL type frequency	Ldrmodules
39	Number of services	Svcscan
40	Number of driver names	Devicetree
41	Found duplicate driver name (0 or 1)	Devicetree
42	Number of device offsets	Devicetree
43	Number of device names	Devicetree
44	Number of kernel drivers	Modscan
45	Number of timers	Timers
46	Number of distinct timer periods	Timers
47	Number of distinct timer modules	Timers
48	Number of callbacks	Callbacks
49	Number of distinct callback types	Callbacks
50	Number of distinct callback modules	Callbacks
51	Number of distinct callback details	Callbacks

stealth techniques that could not be fully identified without using memory analysis, we investigated technical information about memory indicators corresponding to a malicious activity and memory analysis techniques on various technical resources. We picked commonly used 13 Volatility plugins in terms of efficiency and also performance, because some plugins take huge amount of time to finish analysis and not feasible to perform on thousands of samples. From results of these plugins, we extracted a total of 51 features. In Table 3.7, all extracted memory features are listed along with the Volatility plugin used for the feature. These features are novel set of features that have not been used in any other work before, at least not that we know of because we have not come across any other work that utilizes Volatility or similar framework.

3.3 Proposed Method

Figure 3.3 shows the overall targeted malware classification process and steps of the proposed method in detail.

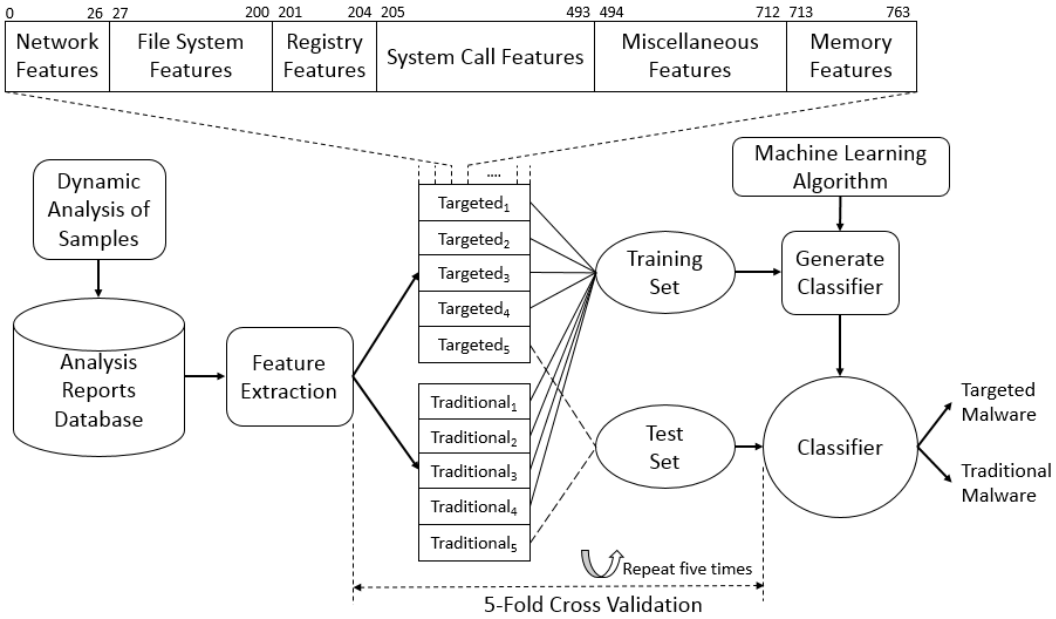

Figure 3.3: Targeted malware classification process.

In our method, we first dynamically analyze all malware samples, targeted

and traditional, and submit the analysis reports to the database. After all the analyses are completed and stored in the database, we extract features defined in the previous section and construct the feature vectors of each samples which composed of 764 features in six different categories. Then we employ 5-fold cross validation and divide targeted and traditional malware samples into five equal size splits. While we use four splits for training and generating a classifier using a selected machine learning algorithm, remaining one split is used for testing the generated classifier and each malware sample in the test set is classified as targeted or traditional malware. We repeat it for five times and average the classification results in terms of measures such as accuracy, precision, etc.

Chapter 4

Evaluation

In the previous chapter we have explained the proposed method and features extracted for utilizing in machine learning schemes for detecting targeted malware. In this section we provide detailed information about the experiment results on targeted malware classification.

The chapter is organized as follows. In section 4.1, we first describe the data set used for classification. Section 4.2 presents the experiment results on various measures. Section 4.3 evaluates features using various feature selection methods.

4.1 Data Set

One of the most important part of targeted malware classification is obtaining the true targeted malware samples. While there are several malware repositories, we dont know much about the provided malware samples there, e.g., which malware family they belong to or if they are used in a targeted attack. There are cyber security firms dealing mostly with targeted attacks, but they do not share the targeted malware samples because they are their intellectual properties.

When we contacted Symantec about the targeted and traditional malware

samples needed for our thesis work, they provided us not the malware samples but list of hashes both for targeted and traditional malware. We then searched for these hashes in Virustotal [23] and downloaded the found malware samples.

Final dataset is composed of 2112 samples, 478 targeted and 1634 traditional malware samples. We used different number of samples for each malware class to reflect the real world distribution where number of targeted malware is a small fraction of total number of traditional malware.

4.2 Evaluation Measures

We have utilized a wide variety of evaluation measures to measure the classification performance of several supervised learning algorithms employed in experiments.

Accuracy: proportion of the correctly classified samples to all samples.

$$\frac{TP+TN}{N}$$

Precision: proportion of the correctly positive classified samples to all positive classified samples.

$$\frac{TP}{TP+FP}$$

Recall: proportion of the correctly positive classified samples to all positive samples.

$$\frac{TP}{TP+FN}$$

F1-score: harmonic mean of precision and recall.

$$2x\frac{PrecisionxRecall}{Precision+Recall}$$

False-positive rate: proportion of the incorrectly positive classified samples to all negative samples.

$$\frac{FP}{FP+TN}$$

Area under ROC curve: ROC curves are typically used in binary classification and constructed by plotting the True Positive Rate (TPR) on the Y-axis against the False Positive Rate (FPR) on the X-axis. Performance of the classifier is measured by the area under the ROC curve where 1.0 represents the perfect prediction and 0.5 represents the random prediction.

4.3 Experiment Results

To evaluate the proposed method and discriminative power of the features extracted in terms of targeted malware classification, we conducted experiments using the following supervised learning algorithms: Support Vector Machine (SVM), Logistic Regression, k-Nearest-Neighbor (kNN) and Decision Tree. In all our experiments, we used 5-fold cross validation where the dataset is divided into five equally sized partitions with four partitions used to train the classifier and the remaining partition used for validation. This process repeated five times and resulting scores were averaged. Because we have extracted a wide-variety of features each lying in differing ranges of value, we employed feature scaling and map the feature values between 0 and 1 to avoid biasing toward any feature resulting misclassification.

We have utilized scikit-learn [20] library for Python to perform machine learning on extracted features. Experiment results are presented in Table 4.1.

From the experiment results, it can be deduced that all supervised learning algorithms performed well on accuracy measure achieving above 88%. While SVM algorithm with RBF kernel achieved best results on measures accuracy, precision and f-1, k-Nearest Neighbor algorithm yielded best recall and lowest

Table 4.1: Targeted malware classification results.

Algorithm	**Accuracy**	**Precision**	**Recall**	**F1**	**False-Pos.**
SVM RBF	92.43 %	87.50 %	78.57 %	82.80 %	6.27 %
SVM Linear	89.60 %	80.68 %	72.45 %	76.34 %	8.06 %
SVM - Poly	88.89 %	83.12 %	65.31 %	73.14 %	9.83 %
K-NN	89.13 %	74.53 %	80.61 %	77.45 %	5.99 %
Log. Regression	88.42 %	78.82 %	68.37 %	73.22 %	9.17 %
Decision Tree	88.42 %	81.82 %	64.29 %	72.00 %	10.12 %

Table 4.2: Area under ROC curve results.

Algorithm	**AUC**
SVM RBF	0.9158
SVM Linear	0.8759
SVM - Poly	0.8623
K-NN	0.8615
Log. Regression	0.8899
Decision Tree	0.8432

false-positive rate.

In order to get more insight on performance of the algorithms in detecting targeted malware, we computed area under the ROC curve for all algorithms experimented and plotted the corresponding ROC curves. Experiment results on area under the ROC curve measure is presented in Table 4.2 and ROC curves are shown in Figure 4.1.

ROC area represents overall performance of a classifier. ROC areas presented in Table 4.2 clearly shows that SVM algorithm with RBF kernel outperformed others in terms of ROC area. Suprisingly, Logistic Regression achieved the second best performance. Although k-Nearest Neighbor algorithm yielded the lowest false-positive rate, it achieves second worst performance in terms of area under the ROC curve measure after Decision Tree algorithm.

From results on area under the ROC curve measures, it is confirmed that SVM with RBF algorithm achieves best performance in detecting targeted malware.

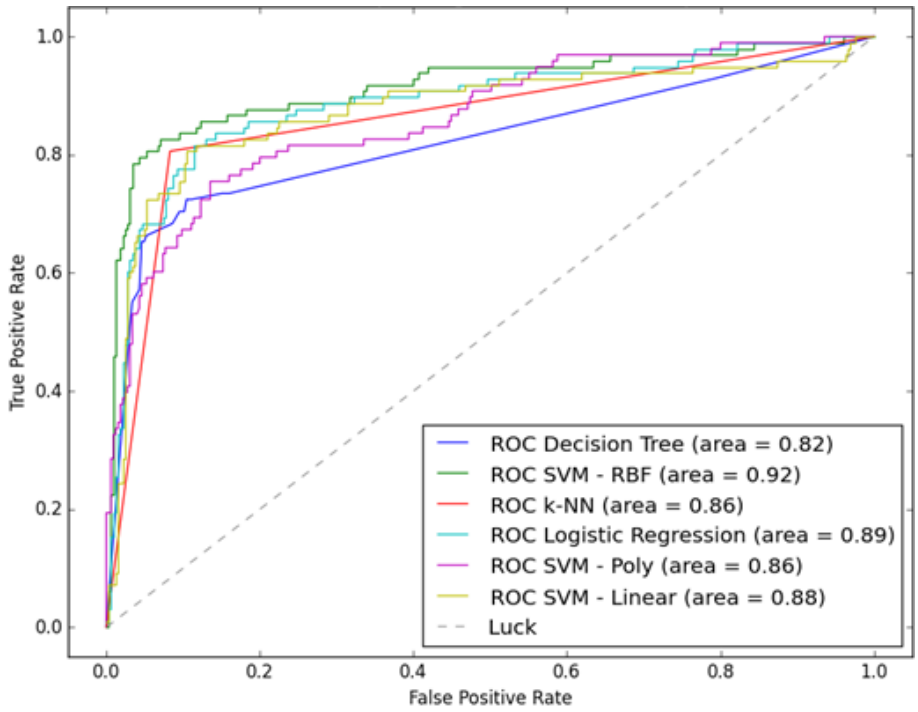

Figure 4.1: ROC curves for targeted malware classification.

Therefore we determined that SVM with RBF achieves best performance overall.

It can be concluded from experiment results on accuracy, precision, recall, f1-score, false-positive rate measures presented in Table 4.1 and also from area under the ROC curve measure presented Table 4.2 that, our proposed method can successfully detect targeted malware and classify targeted malware from traditional malware with high accuracy and precision yielding low false-positive rate.

4.4 Feature Evaluation

Although we utilized 764 number of features in order to detect targeted malware, not all of them was equally important in discriminating the targeted malware behavior from the traditional malware. In order to determine contribution of the features to the accuracy with the increasing number of features selected in

classification of targeted malware task, we performed widely-used univariate feature selection method of ANOVA in conjunction with our best performing SVM algorithm with RBF kernel. Figure 4.2 shows that SVM RBF classifier achieves better performance in terms of accuracy with the increasing number of features.

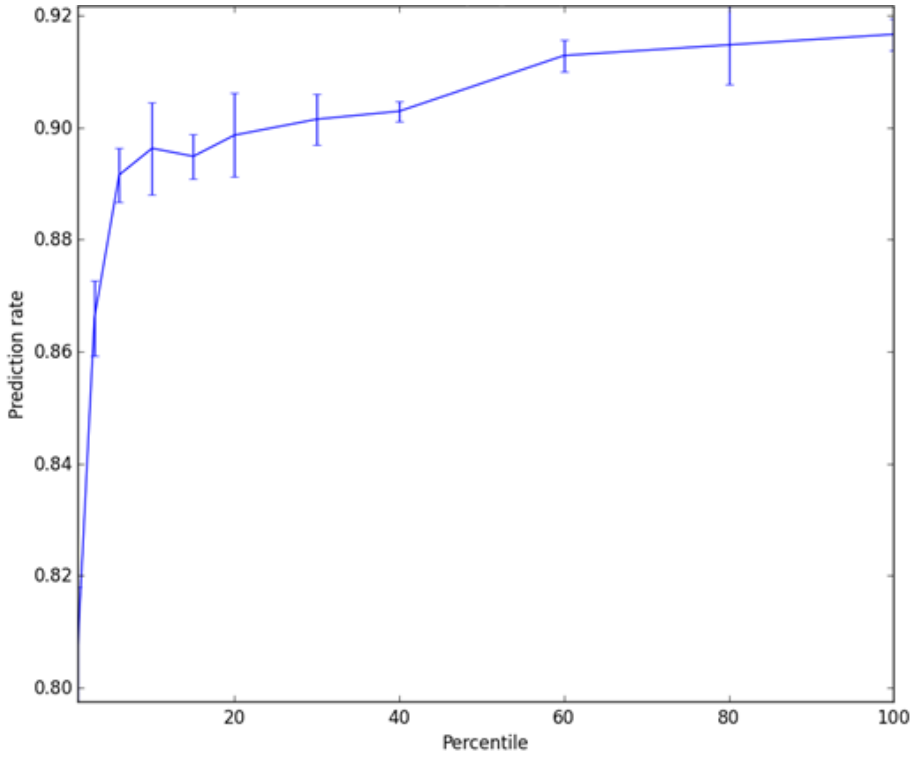

Figure 4.2: Univariate feature selection with ANOVA and SVM.

Then we computed corresponding ANOVA F-values and selected top features based on their F-values. In order to evaluate feature categories presented in the previous chapter, we selected top features and then calculated the distribution of top features to their feature categories. Table 4.3 shows feature categories and number of features among top features belonging to the corresponding category. Top three categories holding highest number of top features are miscellaneous, system call and file system categories, respectively. Memory features comes fourth after file system and possess 6 features in top 100 features.

In order to compute optimal number of features, we used Recursive Feature Elimination (RFE) method which automatically tunes number of features selected with cross validation (see Figure 4.3). However, because RFE method requires a

Table 4.3: Top features distribution.

Feature Category	**Top 10**	**Top 20**	**Top 50**	**Top 100**	**Top 200**
Network features	0	0	0	1	8
File system features	0	2	6	13	21
Registry features	0	0	0	0	0
System call features	3	5	13	32	77
Miscellaneous features	7	13	30	48	79
Memory features	0	0	1	6	15

linear classifier, we performed method with SVM Linear, second best performing classifier in terms of accuracy (see Table 4.1) and optimal number of features are computed as 353. Results of the RFE methods are similar to the ANOVA-SVM (see Figure 4.2) where most of the features are informative and high number of features are required to achieve the best accuracy.

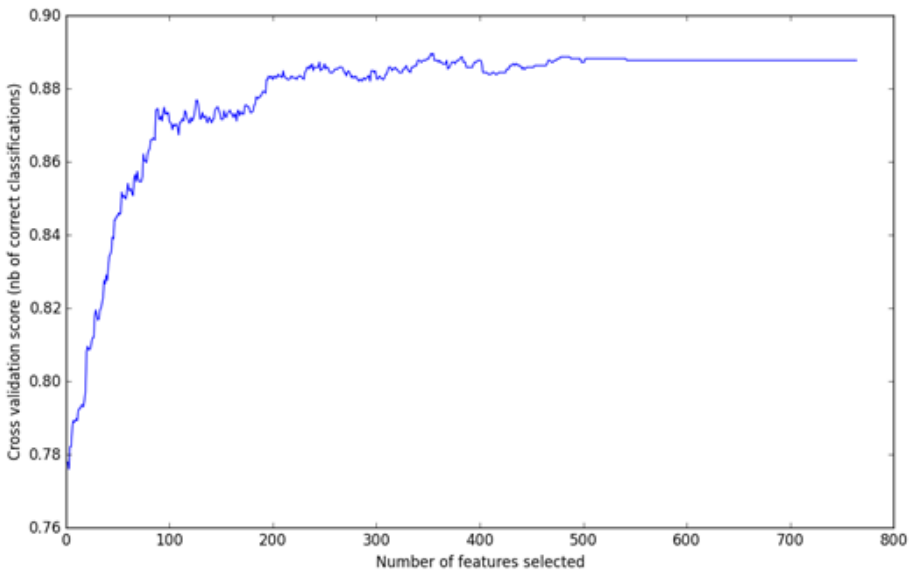

Figure 4.3: Optimal number of features for SVM with linear kernel.

Chapter 5

Conclusion

Targeted attacks become top threats for national cyber security over the last couple years. Targeted attacks evade security defenses and maintain long-time presence in victim environments and operate slow and stealthy to avoid detection. Therefore, targeted malware employed in such attacks is expected to behave differently on systems than the traditional malware because of its stealthy nature. By capturing this characteristic behavior, we thought that it is possible to distinguish targeted malware from traditional malware by performing machine learning.

In this thesis, we propose a method to automatically detect and classify targeted malware using behavioral and memory features extracted from dynamic and memory analysis results via machine learning. We defined a novel set of memory features that will help identifying targeted malwares more efficiently and also made use of several dynamic features already utilized in malware classification works published in the literature. We utilized wide range of features divided into six categories and provided detailed explanation.

We then evaluated the method on a data set comprised of targeted and traditional malware over several supervised learning algorithms and presented the result. Results suggested that method can detect and classify targeted malware

efficiently with 92% accuracy and 0.92 area under ROC curve.

Further work is required to identify the discriminative power of each feature defined in this thesis. Features with low discriminative power are needed to be extracted from the feature set before incorporating new ones. Memory features could be enriched by employing more Volatility plugins even the ones that requires extensive amount of time.

Bibliography

[1] Virvilis, Nikos, Dimitris Gritzalis, and Theodoros Apostolopoulos. "Trusted Computing vs. Advanced Persistent Threats: Can a defender win this game?", *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, 2013.

[2] Tankard, Colin. "Advanced Persistent threats and how to monitor and deter them." *Network security* 2011.8 (2011): 16-19.

[3] Official website of Cuckoo Sandbox, http://www.cuckoosandbox.org

[4] Teller, Tomer, and Adi Hayon. "Enhancing Automated Malware Analysis Machines with Memory Analysis." *USA: Black Hat*, 2014.

[5] Official website of the Volatility Foundation, http://www.volatilityfoundation.org

[6] Schultz, Matthew G., et al. "Data mining methods for detection of new malicious executables." *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on.* IEEE, 2001.

[7] Kolter, Jeremy Z., and Marcus A. Maloof. "Learning to detect malicious executables in the wild." *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2004.

[8] Ye, Yanfang, et al. "An intelligent PE-malware detection system based on association mining." *Journal in computer virology* 4.4 (2008): 323-334.

[9] Tian, Ronghua, Lynn Margaret Batten, and S. C. Versteeg. "Function length as a tool for malware classification." *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*. IEEE, 2008.

[10] Moser, Andreas, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection." *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 2007.

[11] Willems, Carsten, Thorsten Holz, and Felix Freiling. "CWSandbox: Towards automated dynamic binary analysis." *IEEE Security and Privacy* 5.2 (2007): 32-39.

[12] Bayer, Ulrich, Christopher Kruegel, and Engin Kirda. "TTAnalyze: A tool for analyzing malware.", 2006.

[13] Bailey, Michael, et al. "Automated classification and analysis of internet malware." *Recent advances in intrusion detection*. Springer Berlin Heidelberg, 2007.

[14] Bayer, Ulrich, et al. "Scalable, Behavior-Based Malware Clustering." *NDSS*. Vol. 9. 2009.

[15] Rieck, Konrad, et al. "Learning and classification of malware behavior." *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer Berlin Heidelberg, 2008. 108-125.

[16] Tian, Ronghua, et al. "Differentiating malware from cleanware using behavioural analysis." *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. IEEE, 2010.

[17] Islam, Rafiqul, et al. "Classification of malware based on integrated static and dynamic features." *Journal of Network and Computer Applications* 36.2 (2013): 646-656.

[18] Anderson, Blake, Curtis Storlie, and Terran Lane. "Improving malware classification: bridging the static/dynamic gap." *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 2012.

[19] Mohaisen, Aziz, Omar Alrawi, and Manar Mohaisen. "Amal: High-fidelity, behavior-based automated malware analysis and classification." *Computers & Security* (2015).

[20] Official website of scikit-learn, http://scikit-learn.org/stable/index.html

[21] "Common Folder Variables." Microsoft Malware Protection Center. Microsoft. Web. 7 Sept. 2015. ¡https://www.microsoft.com/security/portal/mmpc/shared/variables.aspx¿.

[22] Official website of YARA, https://plusvic.github.io/yara

[23] Official website of Virustotal, https://www.virustotal.com

[24] Bilge, Leyla, et al. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." *NDSS*. 2011.

Appendix A

Top file extensions

‘bak’, ‘bat’, ‘bmp’, ‘cfg’, ‘clb’, ‘com’, ‘com’, ‘dat’, ‘db’,
‘dll’, ‘doc’, ‘docx’, ‘exe’, ‘ico’, ‘ime’, ‘inf’, ‘ini’, ‘jpeg’,
‘jpg’, ‘js’, ‘lnk’, ‘log’, ‘otf’, ‘pdf’, ‘pnf’, ‘png’, ‘reg’,
‘rtf’, ‘sav’, ‘sys’, ‘tmp’, ‘txt’, ‘xls’, ‘xlsx’, ‘xml’

Known paths

c:\documents and settings\all users\application data,
c:\documents and settings\all users\start menu\programlar,
c:\documents and settings\all users,
c:\documents and settings\alice\local settings\temp,
c:\documents and settings\alice\local settings\application data,
c:\documents and settings\alice\start menu\programlar,
c:\documents and settings\alice\application data,
c:\documents and settings\alice,
c:\program files\common files,
c:\program files,
c:\recycler,
c:\windows\system32\drivers,
c:\windows\system32,

```
c:\windows
```

System call categories

```
'system', 'filesystem', 'browser', 'com', 'crypto', 'process',
'synchronization', 'registry', 'misc', 'services', 'windows',
'device', 'network', 'threading', 'hooking', '__notification__'
```

API functions

```
'NtOpenSection', 'NtWaitForSingleObject', 'GetAsyncKeyState',
'NtDeleteValueKey', 'WSARecv', 'getaddrinfo',
'InternetGetConnectedState', 'NtCreateEvent',
'GetFileVersionInfoSizeW', 'GetAdaptersAddresses',
'NtMakeTemporaryObject', 'NtRenameKey', 'HttpSendRequestA',
'GetLocalTime', 'NetUserGetLocalGroups', 'FindFirstFileExW',
'CryptRetrieveObjectByUrlW', 'NtReadVirtualMemory',
'HttpAddRequestHeadersA', 'RegOpenKeyExW', 'NtDelayExecution',
'InternetCrackUrlA', 'SetErrorMode', 'ShellExecuteExW',
'RegOpenKeyExA', 'HttpSendRequestW', 'HttpAddRequestHeadersW',
'GetCursorPos', 'JsEval', 'GetUserNameW', 'WinHttpSetTimeouts',
'WaitForDebugEvent', 'FindWindowExA', 'GetUserNameA',
'NtCreateFile', 'TransmitFile', 'GetSystemTimeAsFileTime',
'WinHttpOpen', 'NtLoadDriver', 'GetDiskFreeSpaceA',
'NtCreateProcess', 'NtDeleteKey', 'WinHttpQueryHeaders',
'InternetSetOptionA', 'CryptGenKey', 'recvfrom', 'CryptEncrypt',
'sendto', 'NtSuspendThread', 'NtQueryInformationFile',
'RegCreateKeyExW', 'GetSystemTime', 'DeviceIoControl',
'WSASendTo', 'FindFirstChangeNotificationW', 'NtQueryKey',
'OpenServiceA', 'WriteProcessMemory', 'WSARecvFrom',
'NtSetContextThread', 'HttpEndRequestW', 'RegQueryValueExA',
'RemoveDirectoryW', 'EnumWindows', 'OpenServiceW', 'NtSetValueKey',
```

‘LookupPrivilegeValueW’, ‘NtQueryValueKey’, ‘RegCreateKeyExA’,
‘RemoveDirectoryA’, ‘HttpEndRequestA’, ‘RegQueryValueExW’,
‘WSASocketW’, ‘NetUserGetInfo’, ‘SetWindowsHookExW’,
‘ExitWindowsEx’, ‘WSASend’, ‘WinHttpGetProxyForUrl’,
‘StartServiceA’, ‘NtDeviceIoControlFile’, ‘NtReadFile’,
‘CryptCreateHash’, ‘FindWindowExW’, ‘NtWriteFile’,
‘LdrGetDllHandle’, ‘WinHttpSendRequest’, ‘RtlDecompressBuffer’,
‘NtQuerySystemInformation’, ‘NtEnumerateValueKey’,
‘CreateDirectoryExW’, ‘CreateThread’, ‘NtLoadKey’,
‘SetupDiGetClassDevsA’, ‘SetUnhandledExceptionFilter’,
‘NtQuerySystemTime’, ‘GetVolumeNameForVolumeMountPointW’,
‘DnsQuery_A’, ‘CryptDecrypt’, ‘recv’, ‘SetupDiGetClassDevsW’,
‘NtProtectVirtualMemory’, ‘SHGetFolderPathW’, ‘RegDeleteValueW’,
‘GetDiskFreeSpaceExA’, ‘socket’, ‘RegSetValueExW’, ‘WriteConsoleA’,
‘LdrGetProcedureAddress’, ‘NtOpenThread’, ‘CopyFileA’, ‘CopyFileW’,
‘RegSetValueExA’, ‘GetDiskFreeSpaceExW’, ‘NtEnumerateKey’,
‘NtOpenDirectoryObject’, ‘LdrLoadDll’, ‘NtWriteVirtualMemory’,
‘URLDownloadToFileW’, ‘WriteConsoleW’, ‘CreateToolhelp32Snapshot’,
‘SendNotifyMessageA’, ‘RegCloseKey’, ‘NtOpenEvent’,
‘NtSetInformationFile’, ‘HttpSendRequestExW’, ‘NtCreateKey’,
‘WinHttpConnect’, ‘MoveFileWithProgressW’, ‘ioctlsocket’,
‘WSAStartup’, ‘NtTerminateThread’, ‘DbgUiWaitStateChange’,
‘NtTerminateProcess’, ‘send’, ‘shutdown’, ‘SendNotifyMessageW’,
‘COleScript_ParseScriptText’, ‘HttpSendRequestExA’, ‘select’,
‘NtQueryFullAttributesFile’, ‘CreateRemoteThread’,
‘GetSystemMetrics’, ‘NtQueueApcThread’, ‘WSASocketA’,
‘CreateServiceA’, ‘WinHttpSetOption’, ‘InternetCloseHandle’,
‘DeleteFileA’, ‘NtLoadKey2’, ‘CryptExportKey’,
‘CryptImportPublicKeyInfo’, ‘NtAllocateVirtualMemory’,
‘ReadProcessMemory’, ‘CreateDirectoryW’, ‘DeleteFileW’,
‘VirtualProtectEx’, ‘CreateServiceW’, ‘listen’, ‘NtCreateThread’,
‘GetComputerNameW’, ‘NtResumeThread’, ‘CryptAcquireContextA’,
‘setsockopt’, ‘InternetReadFile’, ‘CoCreateInstance’,

‘RegEnumKeyExW’, ‘FindNextFileW’, ‘ObtainUserAgentString’, ‘CryptAcquireContextW’, ‘DnsQuery_W’, ‘NtCreateNamedPipeFile’, ‘GetComputerNameA’, ‘NtReplaceKey’, ‘RegEnumKeyExA’, ‘closesocket’, ‘NtGetContextThread’, ‘RtlCreateUserThread’, ‘RegEnumValueW’, ‘NtCreateSection’, ‘StartServiceW’, ‘WinHttpGetIEProxyConfigForCurrentUser’, ‘SetWindowsHookExA’, ‘NtOpenMutant’, ‘InternetOpenA’, ‘NtDeleteFile’, ‘NSPStartup’, ‘IsDebuggerPresent’, ‘RegEnumValueA’, ‘WinHttpReceiveResponse’, ‘InternetOpenW’, ‘CreateProcessInternalW’, ‘connect’, ‘RegDeleteKeyA’, ‘NtDuplicateObject’, ‘RegNotifyChangeKeyValue’, ‘NtQueryMultipleValueKey’, ‘HttpOpenRequestA’, ‘OpenSCManagerW’, ‘GetSystemInfo’, ‘NtCreateProcessEx’, ‘accept’, ‘FindWindowW’, ‘ControlService’, ‘NtClose’, ‘RegDeleteKeyW’, ‘CryptHashData’, ‘NtOpenProcess’, ‘FindWindowA’, ‘HttpOpenRequestW’, ‘NtFreeVirtualMemory’, ‘Process32NextW’, ‘GetLastInputInfo’, ‘InternetConnectW’, ‘UnhookWindowsHookEx’, ‘InternetWriteFile’, ‘GetDiskFreeSpaceW’, ‘NtSaveKeyEx’, ‘RegEnumKeyW’, ‘InternetConnectA’, ‘NtSaveKey’, ‘SetWindowLongA’, ‘CDocument_write’, ‘WSAConnect’, ‘RegDeleteValueA’, ‘CopyFileExW’, ‘NtMapViewOfSection’, ‘SetupDiGetDeviceRegistryPropertyW’, ‘Process32FirstW’, ‘DeleteService’, ‘LsaOpenPolicy’, ‘NtOpenFile’, ‘RegQueryInfoKeyW’, ‘NtUnmapViewOfSection’, ‘NtQueryDirectoryFile’, ‘NetGetJoinInformation’, ‘FindFirstFileExA’, ‘gethostbyname’, ‘DecodeImage’, ‘NtQueryAttributesFile’, ‘RegQueryInfoKeyA’, ‘NtCreateMutant’, ‘GetAddrInfoW’, ‘InternetOpenUrlA’, ‘WSAAccept’, ‘bind’, ‘NtOpenKey’, ‘InternetCrackUrlW’, ‘DnsQuery_UTF8’, ‘CoInternetSetFeatureEnabled’, ‘NtResumeProcess’, ‘OpenSCManagerA’, ‘GetFileVersionInfoW’, ‘CryptDecodeObjectEx’, ‘InternetOpenUrlW’, ‘OpenSCManagerA’, ‘WinHttpOpenRequest’, ‘SetupDiGetDeviceRegistryPropertyA’

Top resolved API files

‘msxml3.dll’, ‘winsta.dll’, ‘icm32.dll’, ‘sqlite3.dll’,
‘msls31.dll’, ‘pstorec.dll’, ‘mpr.dll’, ‘iertutil.dll’,
‘crypt32.dll’, ‘clbcatq.dll’, ‘advapi32.dll’, ‘ole32.dll’,
‘ws2_32.dll’, ‘davclnt.dll’, ‘linkinfo.dll’, ‘mlang.dll’,
‘sqlite.dll’, ‘imgutil.dll’, ‘setupapi.dll’, ‘iphlpapi.dll’,
‘mswsock.dll’, ‘avicap32.dll’, ‘nss3.dll’, ‘msvcrt.dll’,
‘rpcrt4.dll’, ‘ieui.dll’, ‘ieproxy.dll’, ‘urlmon.dll’,
‘odbc32.dll’, ‘apphelp.dll’, ‘dnsapi.dll’, ‘msv1_0.dll’,
‘oleaut32.dll’, ‘netapi32.dll’, ‘ntdll.dll’, ‘xpshims.dll’,
‘shdocvw.dll’, ‘mfc42.dll’, ‘ntlanman.dll’, ‘hnetcfg.dll’,
‘acroiehelper.dll’, ‘comdlg32.dll’, ‘rtutils.dll’, ‘usp10.dll’,
‘uxtheme.dll’, ‘winspool.drv’, ‘rasman.dll’, ‘mfc42u.dll’,
‘comctl32.dll’, ‘winrnr.dll’, ‘msctfime.ime’, ‘samlib.dll’,
‘rasapi32.dll’, ‘user32.dll’, ‘gdi32.dll’, ‘ogl.dll’,
‘shlwapi.dll’, ‘msimg32.dll’, ‘ieframe.dll’, ‘mshtml.dll’,
‘mscms.dll’, ‘actxprxy.dll’, ‘msvbvm60.dll’, ‘winmm.dll’,
‘msctf.dll’, ‘dciman32.dll’, ‘wbemsvc.dll’, ‘xmllite.dll’,
‘sensapi.dll’, ‘psapi.dll’, ‘mso.dll’, ‘faultrep.dll’,
‘kernel32.dll’, ‘drprov.dll’, ‘msi.dll’, ‘shell32.dll’,
‘acgenral.dll’, ‘userenv.dll’, ‘gdiplus.dll’, ‘wintrust.dll’,
‘wshtcpip.dll’, ‘imm32.dll’, ‘ntmarta.dll’, ‘olepro32.dll’,
‘rasadhlp.dll’, ‘sqmapi.dll’, ‘scrrun.dll’, ‘winhttp.dll’,
‘fastprox.dll’, ‘version.dll’, ‘wininet.dll’, ‘shfolder.dll’,
‘cscdll.dll’, ‘sxs.dll’, ‘msvfw32.dll’, ‘secur32.dll’,

Cuckoo community signatures

‘recon_beacon’, ‘recon_checkip’, ‘mimics_agent’, ‘antiav_detectreg’,
‘packer_upx’, ‘packer_vmprotect’, ‘packer_armadillo_regkey’,
‘removes_zoneid_ads’, ‘antiemu_wine_func’, ‘network_tor’,
‘browser_helper_object’, ‘disables_wfp’, ‘antivirus_virustotal’,
‘bootkit’, ‘disables_browser_warn’, ‘browser_addon’,
‘antiav_avast_libs’, ‘disables_system_restore’,

‘antivm_generic_disk_setupapi’, ‘antivm_vmware_files’,
‘packer_entropy’, ‘browser_startpage’, ‘recon_fingerprint’,
‘banker_spyeye_mutexes’, ‘disables_uac’, ‘banker_zeus_mutex’,
‘bitcoin_opencl’, ‘modify_uac_prompt’, ‘antivm_vmware_devices’,
‘infostealer_browser’, ‘antisandbox_unhook’, ‘antiav_servicestop’,
‘spoofs_procname’, ‘infostealer_mail’, ‘persistence_ads’,
‘persistence_service’, ‘stealth_file’, ‘sniffer_winpcap’,
‘driver_load’, ‘spreading_autoruninf’, ‘recon_programs’,
‘antiav_detectfile’, ‘rat_xtreme_mutexes’, ‘packer_armadillo_mutex’,
‘deepfreeze_mutex’, ‘injection_createremotethread’,
‘modifies_certs’, ‘antivm_generic_services’,
‘antivm_generic_diskreg’, ‘process_interest’, ‘antivm_generic_bios’,
‘antisandbox_sleep’, ‘network_icmp’, ‘injection_explorer’,
‘darkcomet_regkeys’, ‘antisandbox_suspend’, ‘network_tor_service’,
‘copies_self’, ‘pdf_page’, ‘antianalysis_detectreg’,
‘stealth_hiddenreg’, ‘mimics_filetime’, ‘rat_pcclient’,
‘reads_self’, ‘modify_proxy’, ‘stealth_network’,
‘antisandbox_mouse_hook’, ‘antisandbox_sunbelt_libs’,
‘antisandbox_productid’, ‘network_http’, ‘stealth_hide_notifications’,
‘antisandbox_sboxie_libs’, ‘browser_security’, ‘stealth_window’,
‘ransomware_recyclebin’, ‘deletes_self’, ‘banker_cridex’,
‘banker_zeus_p2p’, ‘stealth_webhistory’, ‘rat_plugx_mutexes’,
‘antidbg_devices’, ‘antivm_generic_scsi’, ‘exec_crash’,
‘antivm_generic_disk’, ‘encrypted_ioc’, ‘network_bind’, ‘dropper’,
‘antivm_generic_cpu’, ‘creates_nullvalue’, ‘injection_rwx’,
‘antidbg_windows’, ‘disables_windowsupdate’, ‘rat_poisonivy_mutexes’,
‘polymorphic’, ‘modify_security_center_warnings’, ‘prevents_safeboot’,
‘infostealer_im’, ‘infostealer_bitcoin’, ‘injection_runpe’,
‘rat_spynet’, ‘virus’, ‘persistence_autorun’, ‘infostealer_keylog’,
‘multiple_useragents’, ‘bypass_firewall’, ‘origin_langid’,
‘process_needed’, ‘infostealer_ftp’, ‘bot_russkill’,
‘rat_fynloski_mutexes’, ‘antiemu_wine_reg’, ‘stealth_timeout’