Behavior-based Classification of Botnet Malware

Peter Ekstrand Berg

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Computer Science and Media Technology
Gjøvik University College, 2011

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

Behavior-based Classification of Botnet Malware

Peter Ekstrand Berg

2011-07-01

Abstract

The rapid development of information technology has led to great advances in personal computers. At the same time, it has also brought a lot of threats, where malware (malicious software) is one of the most severe. According to Symantec, there was a 51 % increase in added malware signatures from 2009 to 2010. To make matters even worse, malware developers are becoming more sophisticated, creating hybrid malware with obfuscation and mutation capabilities. These hybrids are often found in botnets, where capabilities like self-propagation, stealth and remote-control are important. This thesis will analyze malware behavior that employs obfuscation techniques in the context of botnets. Through tools for reverse engineering, digital forensics and data mining, malware behavior is analyzed to solve a two-class classification problem.

Sammendrag

Den raske utviklingen innenfor informasjonsteknologi har ledet til store framsteg for personlige datamaskiner. Denne utviklingen har også ledet til mange trusler, hvor ondsinnet programvare (skadevare) er en av de mest alvorlige. I følge Symantec så har økningen av skadevaresignaturer økt 51 % fra 2009 til 2010. For å gjøre saken verre, så har skadevareutviklere blitt mer sofistikerte og de utvikler hybrider med obfuskerings- og mutasjonsegenskaper. Disse hybridene er ofte å finne i botnets, hvor de innehar viktige egenskaper som å operere ubemerket, infisering av nye datamaskiner og fjernstyring. Denne masteroppgaven analyserer skadevare i botnets, og ved å benytte verktøy for "reverse engineering", digital etterforskning og "data mining", blir skadevareoppførsel analysert for å løse et klassifiseringsproblem.

Acknowledgments

I would like to thank my supervisor, Professor Katrin Franke, for her excellent guidance throughout the preliminary project, where the initial scope of the thesis was defined, and throughout the process of writing this thesis. Especially, in the field of data mining and machine learning her mentoring has been valuable. Furthermore, I would also thank my co-supervisor, Hai Thanh Nguyen, who helped me to improve the method's performance with his extensive knowledge in the field of feature selection. Finally, I would like thank my opponent, Philip Clark, which has provided me with valuable comments to further improve the quality of my thesis.

Contents

Abstract . . . iii
Sammendrag . . . v
Acknowledgments . . . vii
Contents . . . ix
List of Abbreviations . . . xiii
1 Introduction . . . **1**
1.1 Topic covered by the Thesis . . . 1
1.2 Keywords . . . 1
1.3 Problem Description . . . 1
1.4 Justification, Motivation and Benefits . . . 2
1.5 Research Questions and Hypotheses . . . 2
1.6 Methodology . . . 2
1.7 Contributions . . . 4
1.8 Thesis Outline . . . 4
2 Malware Detection and Malware Forensic Analysis . . . **5**
2.1 Digital Forensics Overview . . . 5
2.1.1 Branches of Digital Forensics . . . 5
2.1.2 Forensic Methodologies . . . 6
2.2 Malware Detection and Analysis Introduction . . . 6
2.2.1 Malware Types . . . 6
2.2.2 Malware Detection . . . 7
2.2.3 Malware Analysis . . . 7
2.3 Obfuscation Techniques . . . 8
2.3.1 Packers and Cryptors . . . 8
2.3.2 Polymorphism and Metamorphism . . . 9
2.3.3 Defeating Obfuscation . . . 12
2.4 Botnet Malware . . . 12
2.4.1 Botnet Life-Cycle . . . 12
2.4.2 Botnet Architecture . . . 13
2.4.3 Botnet Detection . . . 15
2.5 Static Malware Analysis . . . 16
2.5.1 Static Malware Forensics . . . 16
2.5.2 Windows Portable Executables . . . 17
2.5.3 Application Programming Interface Calls . . . 18
2.6 Dynamic Malware Analysis . . . 20
2.6.1 Dynamic Malware Forensics . . . 20
2.6.2 Application Programming Interface Tracing . . . 22
2.6.3 Virtual/Sandbox Environment . . . 23
3 Machine Learning and Data Mining . . . **25**
3.1 Machine Learning and Data Mining Introduction . . . 25

3.1.1 Machine Learning 25
3.1.2 Data Mining 27
3.1.3 Applications 28
3.1.4 Challenges 28
3.2 Features and Feature Quality 29
3.2.1 Feature Roles and Types 29
3.2.2 Feature-Quality Measures 30
3.3 Data-Preprocessing Methods 32
3.3.1 Feature Discretization 32
3.3.2 Missing/Unknown Feature Values 33
3.3.3 Feature Selection 33
3.4 Classification vs Clustering 34
3.4.1 Classification 34
3.4.2 Clustering 38
3.4.3 Main Differences 39
4 New Computational Method for Static and Dynamic Analysis 41
4.1 deLink Framework 41
4.2 Theoretical Method 42
4.2.1 Preferences and Assumptions 42
4.2.2 System Design 42
4.2.3 Data Acquisition 43
4.2.4 Feature Extraction 44
4.2.5 Preprocessing 47
4.2.6 Classification 48
4.2.7 Evaluation 48
4.3 System Implementation 49
4.3.1 Tools, Libraries and Data Formats 50
4.3.2 Implementation Details 54
4.3.3 Summary 57
4.4 Method Discussions 58
4.4.1 Acquiring Analysis Reports 58
4.4.2 Implementation Challenges 58
5 Experiments 61
5.1 Experimental Environment 61
5.1.1 System Setup 61
5.1.2 Dataset 62
5.2 Experiment Scenarios 62
5.2.1 Proof-of-Concept 63
5.2.2 Complete Feature Sets 63
5.2.3 Reduced Feature Sets 64
5.3 Experiment Results 65
5.3.1 Proof-of-Concept 65
5.3.2 Complete Feature Sets 66
5.3.3 Reduced Feature Sets 67
5.4 Experiment Discussions 70
6 Implications, Discussions and Conclusions 73

6.1 Main Results and Findings 73
6.2 Theoretical Considerations 74
6.3 Practical Implications 75
6.4 Recommendations for Further Research 76
Bibliography 79
A Entity Features 89
A.1 DLL Dependencies 89
A.2 Registry Activities 89
A.3 File Activities 90
A.4 Process and Thread Activities 91
A.5 Network Activity 92
B UML Diagrams 95
B.1 Parser 95
B.2 Feature Extractor 96
B.3 Evaluator 96
C Proof-of-Concept Supplements 97
C.1 Packers Identification 97
C.2 ARFF Viewer 97
C.3 Levenshtein Distance Attempt 98
C.4 DLL Import In Code 98
C.5 Clam AntiVirus Results 98
D Classification Results 103
D.1 Complete Feature Sets 103
D.2 Reduced Feature Sets - CFS 105
D.3 New Combined Feature Set - CFS 108
D.4 Reduced Feature Sets - $\mathsf{GeFS}_{\mathsf{CFS}}$ 109
D.5 New Combined Feature Set - $\mathsf{GeFS}_{\mathsf{CFS}}$ 111
D.6 Complete Feature Sets - Unsupervised Discretization 112
D.7 Complete Feature Sets - Supervised Discretization 115
E Source Code 119
E.1 deLink Integration 119
E.2 Feature Extractor 122
E.3 Evaluator 164

List of Abbreviations

API	Application Programming Interface
ARFF	Attribute-Relation File Format
C&C	Command and Control
CFS	Correlation-based Feature Selection
CPU	Central Processesing Unit
DDoS	Distributed Denial of Service
DLL	Dynamic-Link Library
DNS	Domain Name System
DOM	Document Object Model
FN	False Negative
FP	False Positive
FTP	File Transfer Protocol
GeFS	Generic Feature Selection
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
IRC	Internet Relay Chat
ISP	Internet Service Provider
JPEG	Joint Photographic Experts Group
K-NN	K-Nearest Neighbors
Malware	Malicious Software
MD5	Message Digest Algorithm 5
MDL	Minimum Description Length
NOP	No Operation Performed
OEP	Original Entry Point
P2P	Peer-to-Peer
PE	Portable Executable
RAT	Remote Administration Tool
SHA1	Secure Hash Algorithm Version 1.0
SSL	Secure Sockets Layer
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TN	True Negative
TP	True Positive
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language

1 Introduction

This chapter gives a brief introduction to the increasing threat of botnets and how it is repelled with malware analysis techniques. It emphasizes on the problems involved and states the research questions we are trying to solve. Additionally, the thesis' methodology, contributions and outline are presented before going further into the background theory, in Chapter 2 and 3.

1.1 Topic covered by the Thesis

Over the past decade, the number of malicious software, or malware for short, has grown rapidly. The 2010 Symantec Internet Security Threat Report [1] reported over 5 million malware in circulation on the Internet. Today, malware does not fit into well-defined categories anymore, since they are becoming multifaced and more modular. Botnets are one of these outcomes, and it is a phenomenon where thousands of computers are compromised and remotely controlled as robots, or bots for short. These bots constitute a serious threat as they can be exploited to gain access and/or cripple systems and critical infrastructures worldwide.

One of the greatest challenges is that malware developers constantly find new obfuscation techniques, and new malware variants of the same malware family arise. This implies that virus code will be difficult to detect by commercial anti-virus applications. In short, there are several methods to acquire this alteration. Encrypting parts of the malware code using different encryption for each infection makes the malware look different for each time. Also, instead of encryption, malware developers are employing more advance mutation techniques to actually make the malware code completely different after each infection [2]. Detecting this malware type requires a behavior analysis, either sacrificing a host with appropriate monitoring capabilities or in more controlled environment.

1.2 Keywords

Malware Analysis, Botnets, Digital Forensics, Data Mining, Machine Learning

1.3 Problem Description

Botnets are heavily used for computer crime, where they utilize malware to remotely initiate and control illegal activities. In this context the malware has several roles, such as autonomously expanding the botnet by searching for vulnerabilities in new hosts, and opening backdoors that enables the adversary to gain control over the new bot members. As a result of employing obfuscation techniques, different malware behavior may occur within the same botnet. This complicates the evidence acquisition and analysis for the forensic investigators.

There are two general approaches to malware analysis, namely static analysis that studies the malware without executing it, and dynamic analysis where the behavior of the malware is observed. A digital forensic framework, called *deLink*, can detect static malware traces and link these across several computers [3]. However, this framework lacks the functionality to analyze the behavior of malware. This work will therefore study a combined approach for static and dynamic malware analysis. It will focus on which features are most relevant in order to detect the malware and their behaviors. The findings will be integrated in *deLink* in order to increase the framework's efficiency and effectiveness.

1.4 Justification, Motivation and Benefits

As malware rapidly evolves and new botnets are forming, the need of a liable detection system at the client side is crucial. Botnet malware may include scanning capabilities in such a way that each infected host can further expand the botnet by exploiting unpatched/unknown vulnerabilities in operating systems. Even with an installed and freshly updated anti-virus software, the average user could remain unnoticed since malware employ methods to stay undetected. The challenge with obfuscation techniques was presented in a Black Hat conference [4], where it was stated that detecting these types of malware is very difficult in real-time or post-mortem analysis. Even though if the original malware will be detected by anti-virus application, a different variant will evade the common pattern matching technique because it yields a different pattern. It is furthermore important to gain knowledge about their behavior to develop accurate detection schemes.

1.5 Research Questions and Hypotheses

In order to get a clear understanding of how botnet malware behave on an infected host, and how they can be detected, this thesis is going to investigate the following questions:

1. Which features are adequate for static and dynamic malware analysis?
2. Can we obtain disjunct or overlapping feature spaces using these approaches?
3. In what manner will obfuscation techniques influence the feature sets and the individual feature parameters?
4. What type of analysis approach is better suited for analyzing botnet malware, when comparing static analysis versus dynamic analysis?

1.6 Methodology

In order to find (i) adequate features for static and dynamic malware analysis, (ii) study whether we can obtain disjunct or overlapping feature spaces, (iii) investigate how obfuscation techniques influence the feature sets, (iv) and determine what type of analysis approach is better suited for analyzing botnet malware, previous work in the field of botnet and malware behavior analysis should be retrieved and studied. A starting point for further research has been presented in Chapter 2, where state of the art approaches for static and dynamic malware analysis are revisited. Furthermore, these surveys about malware behavior [5, 6] and botnet detection [7, 8] are starting points for those who are so far unfamiliar in the field of malware behavior.

Mainly there are two methods of collecting malware in a secure and controlled manner, namely (1) using honeypots that simulates vulnerabilities that botnet malware tend to exploit for propagation [9] and (2) manually downloading it from sites such as vxheavens [10], packetstorm [11] or offensivecomputing [12]. In this thesis botnet malware will be downloaded manually, since it is not guaranteed that one will capture botnet malware with honeypots and it is not guaranteed to capture appropriate malware in a limited time-period [9]. Especially, this thesis needs different malware samples from several malware families. Also, a set of harmless (benign) software should be obtained that have similar behavior-characteristics as botnet malware. This can be achieved with portable software such as mail clients, BitTorrent clients, browsers and so on [13].

The reason for choosing several botnet malware variants within a malware family is because they utilize different behaviors on the host, such as file modifications and network activity. Additionally, the applied obfuscation techniques may also differ within the same malware family. The malware set and benign software set will be used in a two-class classification scheme, where the malware set will be noted as $MW = \{MW_1, ..., MW_n\}$ and the benign software will be noted as $BSW = \{BSW_1, ..., BSW_n\}$.

When dealing with malware, it is worthwhile to use an isolated experimental environment, to be certain it does not propagate to other hosts. Since malware used in this experiment is meant for Windows only, the experiment will utilize a Linux workstation where the malware is going to be analyzed. Additionally, *VMware Workstation* [14] can be employed if it is necessary for an analysis in a Windows environment. Even though the virtual machine guest may get infected, it cannot traverse to the host and further propagate. For static malware analysis, we will use tools that can extract the API calls performed by the malware without executing the malicious code, as for example [15, 16, 17]. These are called PE parsers and will only work with portable executable malware. On the other hand, dynamic malware analysis will be done by a sandbox-environment established via *Anubis* [18]. This sandbox requires PE formatted files, also.

After applying the static and dynamic analysis methods, it will be necessary to implement a feature extraction component. This component shall be integrated with the *deLink* framework by Flaglien [19, 3], and here the *deLink* framework will be used to analyze individual files instead of disk images, because the PE analyzer and *Anubis* operates on single files. To accomplish this, the feature extraction component must support ARFF format output (see Section 4.3.1) of the MW and BSW samples. This output is then fed to the preprocessing component of the *deLink* framework by Flaglien.

Scenario	**Description**
S_1	Feature set from malware and benign software
S_2	Reduced feature set from malware and benign software

Table 1: Two classification scenarios (S_1 and S_2). S_1 uses the complete feature set, and S_2 uses a reduced feature set.

In order to answer the previously stated research questions, in Section 1.5, we have planned two experiment scenarios. Table 1 provides the main outline of classification scheme. In the first scenario (S_1) we plan to use all of the extracted features from the malware and benign software. Further, in the second scenario (S_2) only the most “relevant” features are used in order to improve the overall performance of the classification scheme. Additionally, as an analysis of the classification result this thesis will analyze each of the scenarios based on features extracted from static and dynamic malware analysis, as shown in Table 2. For example in the first scenario (S_1), we analyze samples using all of the extracted features. The first part (1) will only use static features FS_a, the second part (2) will only use dynamic features FD_a, and in the third part (3) both static and dynamic features are used.

	Scenario S_1				**Scenario S_2**			
	FS_a	FD_a	FS_r	FD_r	FS_a	FD_a	FS_r	FD_r
1	1	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	1
3	1	1	0	0	0	0	1	1

Table 2: Two scenarios (S_1 and S_2) and types of features are employed. Each of the scenarios will use only static behavior features (1), dynamic behavior features (2), and a combination of both (3). FS_a is all of the static features and FD_a is all of the dynamic features. Accordingly, FS_r and FD_r are the reduced features sets.

1.7 Contributions

As implied in Burji *et al.* [20], more experimentation with obfuscated malware is necessary in order to come up with more powerful behavior analysis approaches, which later can be applied in commercial anti-virus solutions and digital forensics applications. The planned contribution in this thesis will work towards a combined approach of static and dynamic malware analysis. By cross-comparing and studying these approaches, the expected results will yield whether static or dynamic analysis, or a combination of the two is best suited for this kind of detection. Thus, a strong focus on the features will be necessary when doing the analysis. In addition to the cross-comparison, the results will be integrated with the *deLink* framework [3].

1.8 Thesis Outline

This thesis is divided into several chapters, in a top-down approach, by first presenting the theoretical background of the different disciplines involved, followed by the computational method and the obtained results.

- Chapter 2 presents the theoretical background and related work needed to get a basic understanding of malware detection and malware forensic analysis. It will discuss the areas of static and dynamic analysis of malware.
- Chapter 3 presents the theoretical building blocks of data mining and machine learning methods, where it will focus on features, preprocessing and classification.
- Chapter 4 proposes the new computational method for solving a two-class classification problem of malicious and benign executables. It consists of two major parts, one presenting the theoretical method and one to the practical implementation.
- Chapter 5 describes the environment that was utilized for the experiments, where two main scenarios are conducted in order to evaluate the method.
- Chapter 6 concludes the thesis by discussing the experiment results, theoretical considerations and practical implications of the method. At the end, proposals for further research is given.

2 Malware Detection and Malware Forensic Analysis

Malware detection and malware forensic analysis are two closely related topics that concentrate on finding characteristics about malicious software. They differ in the way that malware detection focus on detecting the malware's characteristics on the infected system. Malware forensic analysis employs malware analysis and reverse engineering methods to further study these characteristics, and therefore gaining knowledge about its behavior.

Our discussion about malware detection and malware forensic analysis will be in the context of botnet malware, which often is a hybrid of different malware categories. Also, these new threats employ techniques to stay undetected by anti-virus software. A general introduction to digital forensics, malware detection and analysis is given to build the foundation of methods in static and dynamic analysis later presented in this thesis.

2.1 Digital Forensics Overview

Digital forensics is a topic that involves principles and procedures to figure out what happened, when and how it happened, and who was involved. These principles and procedures need to be complied in order to preserve and present the evidence in a forensic manner, concerning an incident or crime. Moreover, the term digital forensic investigation is defined by Carrier and Spafford [21]:

> "Digital forensic investigation is a process that uses science and technology to examine digital objects and that develops and tests theories, which can be entered into a court of law, to answer questions about events that occurred."

In the rest of this section we will give a short overview in the field of digital forensics, by first presenting the different branches of digital forensics that are interconnected to methods later used in this thesis. Then important aspects with the forensic methodologies are presented to give a clear idea of the challenges in a forensic investigation.

2.1.1 Branches of Digital Forensics

Today several branches of digital forensics exists. This is the result of a more narrow focus into a specific digital domain. A few examples that are relevant to this thesis are listed below:

- *Computer forensics* involves examining and analyzing data from a computer storage media such as hard disks, memory sticks, diskettes, tapes etc [22].
- *Network forensics* relates to monitoring and analysis of network traffic with the intent of information gathering, legal evidence or intrusion detection [23].
- *Computational forensics* involves utilizing computer power to perform and automate forensics analysis tasks. Here, machine learning and data mining plays an important role [24].
- *Malware forensics* is the process of examining and analyzing suspicious or malicious code, where the purpose is to learn the true purpose behind the piece of code [25].

A complete overview of the different sub-domains in digital forensics is given by Kittelsen *et al.* in [26]. Moreover, to analyze malware in an appropriate and controlled manner, we need to focus on methods used in malware forensics. These methods have different approaches, for example static and dynamic analysis, and have their strengths and weaknesses. See Sections 2.5 and 2.6.

2.1.2 Forensic Methodologies

Digital investigators need guidelines to achieve the best outcome. This is vital in order to get an overview over the different events surrounding the incident. Hence, various forensic methodologies exist for each branch in the digital forensics domain. In the case of a malware infection, it is necessary to examine and gain knowledge of the infected system, its network surroundings and the malware itself [25].

There are different aspects related to the chosen methodology that is worthy of notice. *Forensic soundness* is related to the process and documentation of collecting digital evidence. In digital forensics this is challenging, because an investigator may alter important data on a system when he/she collects evidence. This phenomenon is called *order of volatility* and it states that it is impossible to capture all data from a running system [27]. Thus, the investigator needs to specify what data he/she thinks important and acquire evidence in such a way that preserves the integrity.

Another issue, often done by malware or an attacker, is that critical evidence is destroyed by overwriting data, deleting logs or encrypting incriminating information. This type of situation falls under the term *evidence dynamics*, which is any kind of influence to the evidence that makes it challenging to prove the integrity and reliability of the evidence [25].

2.2 Malware Detection and Analysis Introduction

This section will present background material related to the basics of malware detection and analysis. A definition of malware is given by Preda *et al.* [28]:

> "Malware is a program with malicious intent that has the potential to harm the machine on which it executes or the network over which it communicates."

The following sections will present common categories of malware often found in botnets. Then common methods used in malware detection will be discussed and finally basic malware analysis approaches will be given.

2.2.1 Malware Types

Before we start our discussion of malware detection and analysis it is necessary to define the different types of malware lurking in the wild. The following list will cover the most common types of malware that are usually found in botnets as hybrids; a complete list of malware types is given in [2, 29].

- *Viruses* are malware that infects other files and make them perform some unwanted and harmful function. In other words, a virus copies itself into another file. When the file is executed, the virus functions will also be executed.

- *Worms* are self-propagating malware. This category spreads through networks by for example exploiting known vulnerabilities in commonly used operating systems.

- *Trojan horses* are programs with a disguised intent, by concealing a malicious payload. Trojans may emulate the behavior of an arbitrary program such as an authentication through a login shell and retrieve an user's login credentials.

- *Rootkits* are software with the main purpose of staying concealed and undetected by anti-virus software and end-users. This type of malware was originally intended to provide root-account on UNIX-like systems.
- *Backdoors* are malware used to bypass authentication and/or security measures. When a system has been compromised by one of the previous described types of malware, a backdoor can be installed to allow easier access later on.

2.2.2 Malware Detection

This section describes the most common techniques that are applied in anti-virus applications. These methods have evolved to keep up with the more sophisticated malware and their evasion methods. Besides the mentioned detection schemes below, a complete list is given in [30, 2].

- *String scanning* is the most primitive approach to detect malware. It searches for sequences of strings (bytes) that are typical for a specific malware. Anti-virus companies organize these string sequences as signatures in databases and a local anti-virus application must download the latest signature updates to have the latest means for detecting new malware.
- *Wildcards* is a method that allows the scanner to skip bytes or a range of bytes, for example skip bytes represented with the '?' character. Malware with early-generation obfuscation techniques can be detected with wildcards.
- *Algorithmic scanning methods* are techniques used when the standard algorithm (such as string scanning) of the anti-virus cannot deal with a specific malware. Under this category we find filtering techniques that only scans certain files that are more exposed to infections, for example to apply boot virus signatures to boot sectors. Another technique is decryptor detection that focuses on detecting the decryption component in malware that applies encryption (see Section 2.3).
- *Code emulation* uses a virtual machine that simulates a CPU and memory management system in order to execute the malicious executable. This technique mimics the instruction set of the CPU by using virtual registers and flags. Additionally, the functionality of the operating system must be emulated in such a way that it supports system APIs, files etc. To detect malware with this method the emulator analyzes each of the instructions that are run in the virtual machine.
- *Heuristic analysis* is useful when detecting new malware. This technique looks for certain instructions/commands within an executable that are not found in "benign" executables. However, its biggest disadvantage is that they often find false positives.

2.2.3 Malware Analysis

Malware analysis is techniques that enable us to study and obtain information about a malware's behavior [27]. These techniques are also known as reverse engineering of malware. Commonly used approaches are static (code) analysis that studies the malware without executing it, and dynamic (behavioral) analysis which study malware as they execute. Even though both methods may accomplish the same goal of studying how malware works, the tools and skills required are different [31].

Static analysis is done by analyzing the source code of the malware to study how it functions. Typically, static analysis use reverse engineering tools such as disassemblers, debuggers and compilers. After applying these tools on the malware executable, the investigator or malware analyst can study the source code to gain knowledge on how the malware operates. For example how it infects systems and how it propagates. Additionally, further static analysis methods are discussed in Section 2.5.

The easiest way of doing a dynamic analysis is to run the malware and see what happens. Note that this approach is not without problems, since you may end up destroying all information on your system or letting the malware propagate if the sacrificed host is connected to the Internet. A popular technique is to use a sandbox, which is a controlled environment for running software. Moreover, different techniques in applying dynamic analysis of malware is presented in Section 2.6.

2.3 Obfuscation Techniques

Malware in the wild is often protected with obfuscation (or armoring) techniques. These techniques were first intended to protect the intellectual property of software developers, however these techniques are commonly applied to malware code to make the disassembly process more time-consuming. A definition of obfuscation in the context of malware analysis is given by Madou *et al.* [32]:

> "Code obfuscation makes it harder for a security analyst to understand the malicious payload of a program."

Obfuscating malware is not only used to block out the good guys such as virus researchers, malware analysts and other security professionals, but also other malware writers or hackers from examining the code [25]. For example in botnets (see Section 2.4), the adversary wants to hide how he/she controls the infected computers in the botnet. This is necessary to prevent others from hijacking these computers, to build their own botnets or other forms of fraudulent activity such as phishing, spamming and click fraud.

The following section will discuss the most common utilities that obscures/protects their malware code, namely packers and cryptors. Furthermore, two other popular methods to evade anti-virus applications are presented, which are polymorphism and metamorphism.

2.3.1 Packers and Cryptors

Packers are programs designed to compress, and sometimes encrypt, the contents of an executable file [33, 25]. Thus, in some literature, packers are referred to as compressors. This obfuscation technique works by compressing the executable and obfuscating its contents that ends in a new executable. Before the executable is loaded into memory its content will pass through a decompression routine that extracts the program into memory, see Figure 1.

Yan *et al.* [33] presents more specific information on how packers work. Modern executable files in Windows are PE files (see Section 2.5.2) and packers are therefore designed towards this file format. Most PE packers require executables using dynamic linking (see Section 2.5.3), however there are no restrictions to the programming language. This implies that you can use "everything" from C++ to Assembly.

The first operation a packer is performing is parsing the internal structures of the PE

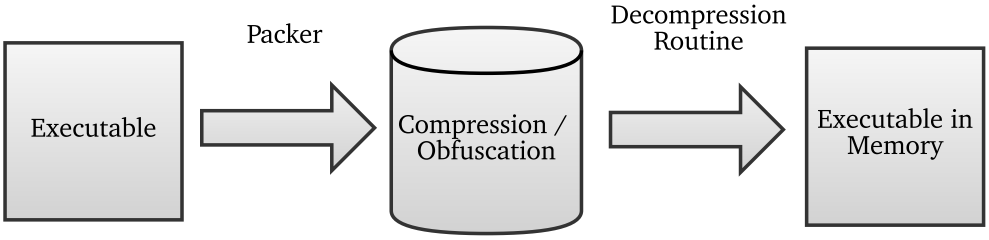


Figure 1: Creation and execution of packed malware

file. Then PE headers, sections, import/export tables are reorganized into new structures. Additionally, it attaches a piece (*stub*) of code that the executable invokes before the original entry point (OEP). When executed, the stub will decompress the original data and locate the OEP. Packers may utilize randomization during packing, which means that it generates different variants every time the executable is packed.

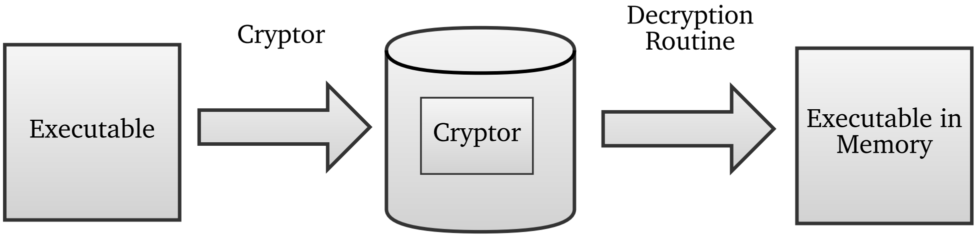


Figure 2: Creation and execution of cryptor protected malware

Cryptors are designed with the same purpose as packers, namely to conceal the content of the binary. This obfuscation technique is also referred to as encryptors or protectors, since it applies an encryption algorithm on the executable, making the content scrambled and undecipherable. As with packers, cryptors have a stub that contains the decryption routine to the encrypted executable, which is loaded when the file is executed, see Figure 2. Also, cryptors may generate different encryption keys which will result in different encrypted files [33].

Unfortunately, few packer/cryptor applications have a native unpacking/decryption ability. There are however, scripts that are targeted towards specific versions of packers [25]. Note that these scripts may not behave as promised, either failing in unpacking the packed executable or infect the system with malware. With proper tools for disassembly and debugging you may do this operation manually, since there are forums on the web that specializes in reverse engineering, such as [34].

2.3.2 Polymorphism and Metamorphism

Advanced packers may utilize a polymorphic or metamorphic engine to make the static analysis even more challenging [33]. However, malware may contain this type of engine, in addition to what is implemented in the packer application, with the purpose of changing the appearance of the malware after an infection [35].

Malware that employs polymorphism[1] will take many forms by applying encryption

[1] In Greek *poly* means many and *morhi* means form.

on the malware body and mutate the decryptor from instance to instance. It is the mutation engine's job to generate new decryption and encryption routines during infections [2, 29]. Figure 3 illustrates the process where the malware applies the new decryption routine with the encrypted code onto the targeted file. The malware body is constant from generation to generation, where D is the decryptor, M the malware body, and G the current generation.

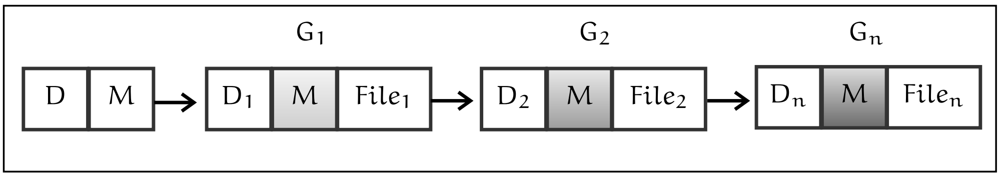


Figure 3: Polymorphic malware instances

Furthermore, metamorphic malware is malware that applies mutation to the malware body and do not use encryption. This will result in instances that never look like its predecessors [36]. A great advantage compared to polymorphic malware is that the malware body is not encrypted, because when the malware body is encrypted it must eventually be decrypted and loaded into memory. Advanced detection methods can wait for the malware to decrypt itself and then detect it [35]. Figure 4 illustrates this, and there are no constant data between the generations.

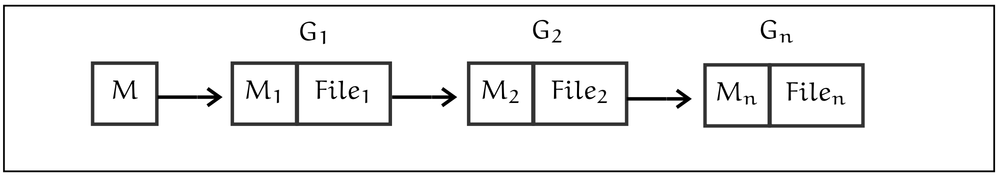


Figure 4: Metamorphic malware instances

Whether the malware is polymorphic or metamorphic its functionality will remain the same [37, 35]. The mutation engine in polymorphic or metamorphic malware applies similar obfuscation techniques, however with polymorphism they are applied on the decryptor and with metamorphism they are applied on the malware body. There exists several malware generation kits on the Web that utilize metamorphic engines such as *Second Generation virus generator* (*G2*), *Next Generation Virus Creation Kit* (*NGVCK*), *Virus Creation Lab for Win32* (*VCL32*) and *Mass Code Generator* (*MPCGEN*) [2].

The next sections will give a brief introduction to the common obfuscation techniques applied in polymorphism and metamorphism. A complete list is given in the technical report by Konstantiou [38].

Dead-Code Insertion

Dead-code insertion (or garbage/junk code) is the simplest form of obfuscation. This is an effective method for changing the malware's appearance by adding ineffective instructions, however, its original behavior will remain the same [39]. A simple example is given by Vinon *et al.* in [36] where we have a malware signature *5150 5B8D 4B38 50E8*

[2] Available at VX Heavens [10].

0000 0000 5B83 C31C. Table 3 shows an assembly code with inserted No Operation Performed (NOP) instructions.

Hex Opcodes	**Assembly**
51	push
90	**nop**
50	push eax
5B	pop ebx
8D 4B 38	lea ecx,[ebx+38h]
50	push eax
90	**nop**
E8 00000000	call 0h
5B	pop ebx
83 C3 1C	add ebx, 1Ch

Table 3: Dead-code insertion example [36]

Thus, the new signature will be *5190 505B 8D4B 3850 90E8 0000 0000 5B83 C31C* and will fool the most primitive signature-based anti-virus applications. NOP instruction will, as the name implies, do nothing, and are easily defeated by modern anti-virus applications since they are designed to remove these instructions before further analysis.

Register Renaming

A different technique is register renaming (or register reassignment) that switches registers from generation to generation [39]. Replacing registers with an equivalent requires that no register dependencies in control flow are affected [36].

Code Transposition

Another technique is code transposition which is done by inserting jump instructions and/or unconditional branches in such a way that the original control flow of the program is maintained. A known malware that uses this technique is Win95/Zperm and is illustrated in Figure 5. This malware inserts and removes jump instructions within its code, where each jump will point to a new instruction of the malware [40]. Detecting this malware with signature-based detection is virtually impossible, since it never generates a constant body anywhere, not even in memory.

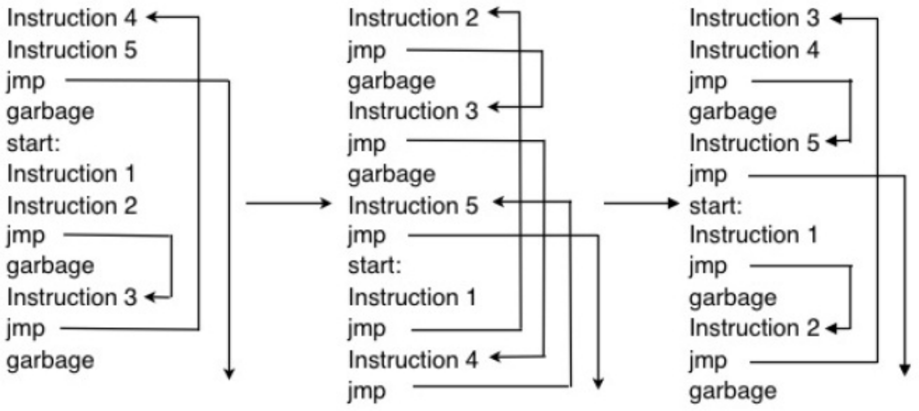


Figure 5: Code transposition example in Zperm [40]

Instruction Substitution

Instruction substitution is a technique that replaces a set of instructions with another set of instructions that are semantically equivalent [36]. To detect malware employing this obfuscation technique it is common to collect different variants of a malware and perform similarity analysis.

2.3.3 Defeating Obfuscation

The digital investigator needs to take care of the applied obfuscation methods to fully explore a suspicious program [25]. To deal with packers, several underground utilities exist, which can only deal with a specific packer. However, these utilities are not guaranteed to work and may not be the best tool for forensic analysis where the findings need to be validated.

For the most skilled malware analysts manual unpacking is the preferred approach [33]. They employ debuggers to analyze the different layers of obfuscation. For example a cryptors encryption and decryption algorithms, where they are able to manually restore the original file. Unfortunately, this is a time-consuming process which requires knowledge in the field of kernel and assembly programming.

Moreover, dumping the process from memory is another approach. This requires that you execute the suspicious file in an isolated environment and employ tools such as *LordPE* [41] or *ProcDump* [42]. Then you need a disassembler to examine the executable. However, not all forms of obfuscation will be defeated with this approach, since there exists “anti-dump” protection for packers [25].

2.4 Botnet Malware

The rise of botnets have become one of the most critical threats to computing assets and infrastructures [43]. A botnet can be exploited for several activities such as distributed denial-of-service (DDoS) attacks, spam, phising and identity theft. The following definition is based on Gu *et al.* [44]:

> “A botnet is a network of compromised computers which is controlled from a central location.”

Moreover, a botnet is the joining of many different threats, since the compromised computers (bots) can propagate their malicious code like worms, hide from anti-virus software like rootkits, initiate attacks and operate as a command and control (C&C) server [7]. A C&C server is the main controlling entity of a botnet, which is operated by the adversary (botmaster).

This thesis will utilize botnet malware in the experiments, which implies that background information is required to give some pointers to suitable features for the analysis. The following sections will present the typical botnet’s life-cycle, architectures and detection methods.

2.4.1 Botnet Life-Cycle

A typical botnet can be created and maintained in five phases [45, 8, 46]; these phases consists of (1) initial infection, (2) secondary injection, (3) connection, (4) malicious command and control, (5) update and maintenance. This five phased life-cycle is shown Figure 6.

During the *initial infection* a computer can be infected by different means, such as

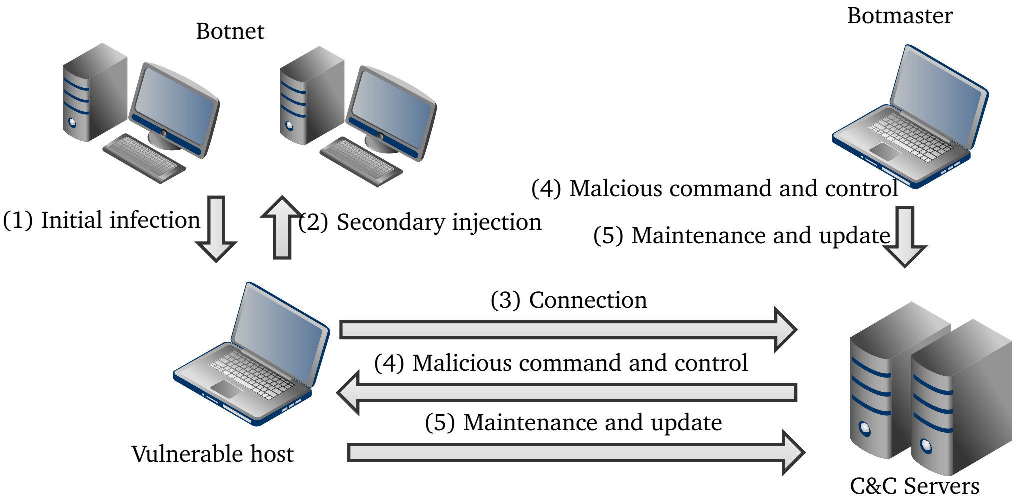

Botnet
Botmaster
(4) Malcious command and control
(1) Initial infection
(2) Secondary injection
(5) Maintenance and update
(3) Connection
(4) Malicious command and control
(5) Maintenance and update
Vulnerable host
C&C Servers

Figure 6: Basic botnet life-cycle

being exploited through vulnerability. This can be done by bots, since most bots include a scanning capability in such a way that each bot can further expand the botnet. One approach to this is to first use scanning tools to check for open ports, then use these acquired ports for a further vulnerability scan. A list of known vulnerabilities that common bots like Agobot and SDBot use, are given in by Schiller *et al.* [45]. Also, there are other methods to provide system access like backdoors left by Trojans, installing malicious software from a web page or from an infected email attachment. During the *secondary injection* phase the infected hosts will execute a script and download the image of the bot binary from a web location (HTTP, FTP or P2P). Furthermore, this bot binary will disable/avoid the system's anti-virus software and open necessary ports so it is able to communicate to the C&C servers.

The next phase, which is the *connection phase*, the bot binary establishes a C&C channel and the host will be connected to a C&C server. It is in this step the host turns into a bot and joins the botmaster's army. This implies that the bot is ready for the *malicious command and control phase* and it will listen to the C&C channel for orders from the botmaster. Thus, the C&C channel enables the botmaster to issue commands remotely to do various malicious activities.

Finally, the *maintenance and update phase* will maintain the bots by for example upgrade the bot binary. Botmasters need to update their bots for several reasons such as to avoid anti-virus software or to add further functionality to the botnet. Server migration is also done when updating the bot binary, which moves the bots to a different C&C server. This method is very useful for the botmasters to keep their botnet alive.

2.4.2 Botnet Architecture

A method to define the characteristics of a botnet is to look at how the bots are communicating with the C&C server. By using these communication channels (C&C channels) the bots can be commanded, maintained and updated [45, 8]. To issue commands to the botnet army, the C&C server(s) can either push commands onto the bots or the bots can pull commands from the C&C server(s) [45, 9]. Common C&C architectures are based on IRC, HTTP and P2P, which are presented below.

IRC-based Architecture

The first botnets were based on an Internet Relay Chat (IRC) architecture, and this is still the common architecture of botnets. The IRC protocol was originally used for online-chat. Thus, it is easy for the botmaster to create IRC servers. The administration of bots can be done effectively, where commands can either be pushed or pulled [47].

When initializing a bot, it tries to contact the IRC server by using an address in the executable binary. Since the possibility of black-listing is high when using IP-addresses the bot master needs to use other methods. Hence, using a DNS name (domain name) instead will allow the bot master to keep hold of the botnet if the current associated IP-address is black-listed.

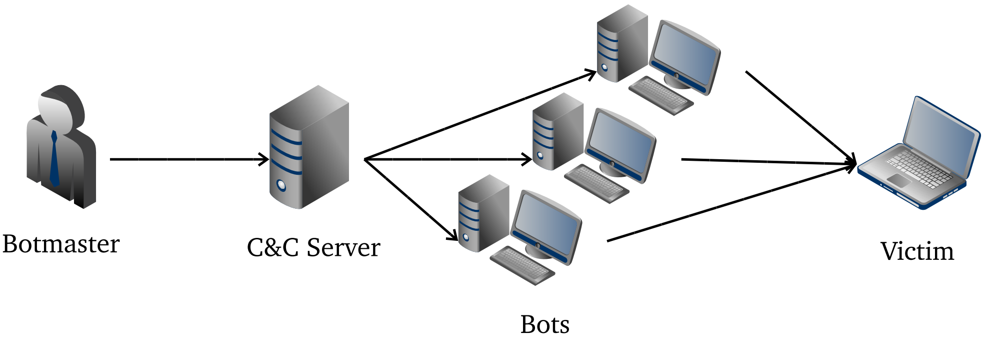


Figure 7: Basic centralized botnet architecture

Web-based Architecture

Web-based architecture uses either HTTP or FTP for C&C channel. These application layer protocols are not as popular as the previously described IRC protocol. However, it does not mean that they are less effective. There are primarily two methods to set up web-based architecture, which are echo-based and command-based [45]. The echo-based technique requires that the bot announces that it exists to the C&C server. On the other hand, command-based works differently in such a way that the botmaster utilizes a Graphical User Interface (GUI) to issue commands to his army of bots.

Furthermore, to increase availability on C&C servers botmasters employ a fast-flux service [46], which associates the DNS with a new IP-address as often as every 3 minutes [48]. Thus, the botmaster assures the availability whether the current IP-address gets blacklisted or not by the ISP.

P2P-based Architecture

The weaknesses in common architectures are that they are centralized (see Figure 7). This means that if the C&C server is taken down, the botnet will be eliminated. In a peer-to-peer (P2P) architecture there is no centralized server, since all nodes act as bot server and client. Thus, if a single node is taken offline the gaps in the network will be unrecognizable and the network continues to function within the control of the botmaster [49].

2.4.3 Botnet Detection

Over the past years common bot malware has been collected and their behavior and characteristics has been analyzed. These findings have been applied in anti-virus software as signatures. However, other detection methods should be considered, since obfuscation techniques will make this type of detection challenging (see Section 2.3). Common characteristics related to bot malware are network activity, because the bot needs some sort of interaction with C&C server(s). Typical characteristics of bot malware is listed in [45]:

- Opens specific ports
- Establishes many unexpected network connections
- Downloads and executes files
- Creates new processes with a familiar name
- Disables anti-virus software

Features (characteristics) that are used for botnet detection varies. Until now research focus on features extracted from network traffic. The survey done by Feily *et al.* [8] describes features used in intrusion detection schemes that detect certain anomalies based on traffic activity such as latency, volume and traffic on unusual ports. This can be further more specific by analyzing parameters from protocols like P2P, IRC, HTTP, DNS, TCP etc. Unfortunately, most of the detection schemes for botnets are best suited for offline analysis. This is necessary since machine learning and data mining approaches are computational demanding methods when dealing with high volume of collected data [50].

Gu *et al.* [51, 52, 44] have developed three systems for botnet detection. *BotHunter* [51] is a system that tracks communication flows from internal to external hosts. Features that are fed into their correlation engine are extracted from outbound scan patterns, which are typically observed when bots search for vulnerabilities to propagate to vulnerable hosts. Outbound connection failure for abnormally high connection rates are interesting to analyze since many IP and DNS addresses may be blacklisted or taken down. Also, payloads are analyzed for anomalies by extracting 1-gram[3] features from the packet payload building a feature vector of 256 bit-values.

BotSniffer [52] and *BotMiner* [44] are two systems that exploit the fact that bots within the same botnet will have the same behavior. The main difference between the approaches is that *BotSniffer* only works for botnets that are IRC-based or HTTP-based. The features extracted are based on message responses between bot and botmaster, and activity responses initialized from the bots when performing a distributed activity. *BotMiner* on the other hand, is architecture independent that clusters features extracted from TCP and UDP flows, such as connection time, IP addresses, ports etc. Additionally, the system employs the anomaly module from *BotHunter* that generates reports based on anomaly activity. By applying hierarchical clustering, they obtained great detection accuracy on different botnets.

Masud *et al.* [53] employed a similar approach as previously described, where they analyzed the correlation between network traffic and execution time of applications. From the obtained data they extracted packet-level and flow-level features used for a

[3] N-gram is a subsequence of length N from a given sequence, e.g., characters within a text.

further classification analysis. A different approach by Strayer *et al.* [54] presented a method suitable for real-time analysis of traffic data. Their detection approach analyzes packet-flow characteristics such as bandwidth, packet timing and burst duration to decide whether this activity belongs to a botnet or not. Seewald and Gansterer [55] employed a passive framework and features are collected from three different levels; by analyzing single packets, network traffic and TCP/IP traffic. By applying Sammon mapping [56], they discovered that communication activity of spambots are similar.

2.5 Static Malware Analysis

This section presents background material on typical techniques that are applied in static malware analysis, specifically what type of malware characteristics or features they utilize. Malin *et al* [25] defined static analysis as:

> “Static analysis is the process of analyzing executable binary code without actually executing the file.”

In the days when we had primitive malware it was an easy task to discover and analyze malware. The major reason for this was that the malware developers were not concerned about stealth and obfuscation techniques. Thus, the malware’s functionality was easily observable and an in-depth analysis of the code would be unnecessary.

Moreover, many static analysis systems have been designed for portable executables and the common approach is to use application programming interface (API) calls to describe the behavior of the malware. Thus, the following sections will give a short introduction to portable executables in Windows and describe approaches using API calls.

2.5.1 Static Malware Forensics

When an investigator tries to gain knowledge about a specific type of malware using static analysis, he/she could use a set of tools to disassemble and debug the sample. According to Malin *et al.* [25], there is a general approach when analyzing a malware or a suspicious executable:

1. First identify and write down the system details where the malicious/suspicious file was obtained. This includes information such as operating system version, installed service pack and patches. Furthermore, the investigators can analyze more in-depth by studying system activities related to network, processes and users, since malware may infect these areas in order to perform their malicious activities.

2. Cryptographic hash values of the executable is a valuable method that creates an unique identifier during the analysis. Malware may remove itself from the current location or change when it is executed. Thus, with the unique identifier you can detect the executable if it moved itself to another location or has changed. Examples of cryptographic hashes are *Message-Digest 5* (*MD5*) and *Secure Hash Algorithm Version 1.0* (*SHA1*).

3. A comparison of the obtained file to other malware is an important step in the file identification process. This will answer whether the executable is benign or malicious. Web pages such as vxheavens [10] and offensivecomputing [12] allow you to search for malware based on *MD5* hashes. However, only one *MD5* or *SHA1* hash may not be appropriate in this situation, because the hash sums will change

with a single bit difference. This problem can be solved by fuzzy hashing or *Context Triggered Piecewise Hashing* [57], that computes a series of checksums for a file.

4. Identification and classification focus on identifying the file type to determine its nature, what operating system and architecture it was meant for. To determine the file type you cannot trust a file's extension. For example an executable may be camouflaged as a JPEG-file. To perform this identification task manually you need to open the file in a hexadecimal viewer/editor and inspect the first 20 bytes of the file, which will reveal the file signature.

5. Scanning and examine the suspicious file with an anti-virus application is the next step. By utilizing several anti-virus applications (locally and online) we can determine whether the file has a known signature. Fortunately, there are free anti-virus applications available such as *Clam AntiVirus*, *Avast Antivirus*, and *Grisoft AVG* [58].

6. Extract and analyze the suspicious file by searching for plain text strings/characters may reveal valuable information. This step may identify program functionality, file names, nicknames, URLs, IP addresses, e-mail addresses etc. Additionally, file metadata and symbolic information are valuable information to investigate in this step. Note that malware developers are aware of this approach and may plant decoys or applying obfuscation techniques to make this challenging. Thus, the type of obfuscation technique(s) applied must be identified.

7. When an executable is linked dynamically it will have dependencies in order to run correctly. To identify these dependencies it is necessary to disassemble the malware. Using tools that dump Dynamic Link Library (DLL) dependencies, such as *pefile* [59], reveals the suspicious file's behavior to a certain degree.

This manual approach is quite common when we are dealing with a single malware instance, however it will be a time-consuming process if we are dealing with a large amount of malware samples.

2.5.2 Windows Portable Executables

Malware directed towards the Windows platform are often using the portable executable (PE) file format. The term portable means in this context that the file is executable on every Windows platform [60, 61], which is an advantage for the malware writers. Furthermore, this format contains a data structure that encapsulates elements such as dynamic library references, API import/export tables, and resource management data.

PE files consists of various sections and headers that describes the section data, import table, export table etc. A PE file starts with a *MS-DOS header* structure. When analyzing this section manually there are two elements worth noticing, namely the DOS executable file signature (*e_magic*) and offset field (*e_lfanew*) to the *PE header*. The second section, *MS-DOS stub* contains mainly a compatibility notification, which implies that the executable will for example print an error message if it is run in a non-Windows environment.

The *PE header* contains the specifics of the PE file, and for the digital investigator there is valuable information stored in this section [25]:

- Target platform/processor
- Time and date the file was created/compiled

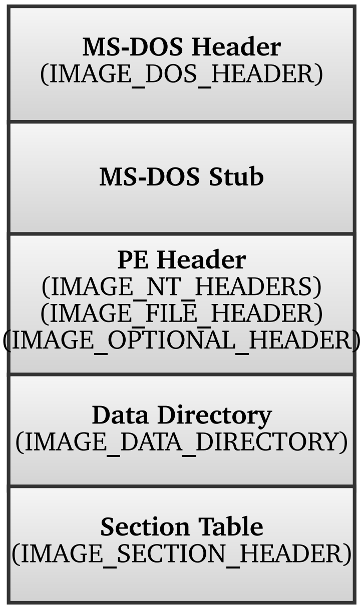


Figure 8: The Portable Executable File Format

- Whether symbols and debugging has been stripped from the file
- File characteristics

Moreover, DLL dependencies can be extracted from the import table in the *data directory*. This table describes required libraries necessary for the file to run successfully. On the other hand, the export table describes functionality that can be exported and utilized from other programs. There are several open source tools to retrieve this type of information from the PE file. This thesis will use *pefile* [59] which is a python module that can access almost all the sections. Thus, it is a powerful tool for static analysis of malware.

A weakness with common PE parsers, like *pefile*, is that they can be fooled by obfuscation techniques. The simplest method of doing this is by compressing (packers), however the report generated by *pefile* will print warnings if the *MS-DOS Header* and *PE Header* are not formatted properly.

The last structure, which is the *section table*, contains different entries or section headers. Here we find the file's *original entry point* (OEP) which is the point where the file execution starts. Additionally, each of the section headers contains name, size, and description of the respective section.

2.5.3 Application Programming Interface Calls

How an executable is linked may be valuable to the investigation [25]. It is the linker's job to assemble any required libraries to the compiled source code (object file) that is required for running the executable, see Figure 9. There are mainly two methods to link an executable; static and dynamic linking. An executable with static linking is self-contained, meaning that the executable contains all necessary libraries and code to run successfully. On the other hand, with dynamic linking, the executable is dependent on shared libraries to run. Typically, these dependencies are often DLLs that are imported from the host operating system when the executable runs.

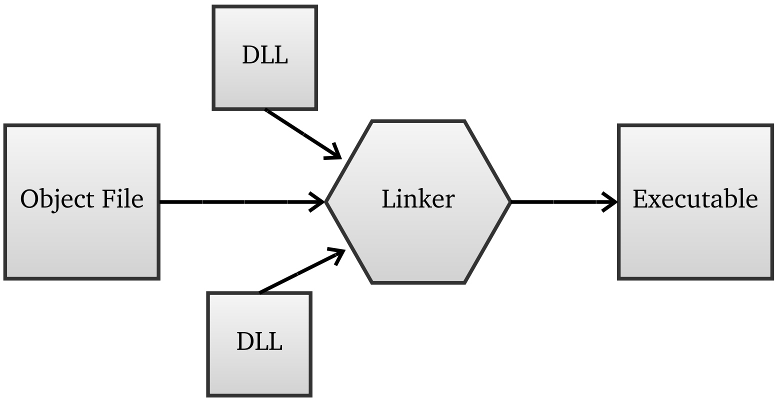


Figure 9: Linking of executables

Windows Application Programming Interface (API) calls are function calls to DLLs that provides functionality you otherwise would have to implement yourself. Thus, programmers (good or bad) use the Windows API to access resources such as processes, network information, registry etc. Also, exploiting functionality in DLLs will make the executable smaller in size, which is an advantage for malware during propagation in the context of required transmission time. Furthermore, API calls can be used to extract information that describes behavior of executables.

Schultz *et al.* [62] developed a framework for detection of new malicious executables. Their framework can automatically find patterns in the dataset to detect new malware, and it supports different methods for feature extraction and different data mining classifiers. They used system resource information, strings and byte sequences that were extracted from the malicious executables using *GNU BIN-Utils* [63]. The first approach used three different types of features. The first feature vector consisted of boolean values describing if the executable used a specific DLL or not. The second feature vector described whether API calls to the different DLLs were called or not. And the third feature vector stored integers of how many API calls were done within each DLL. Furthermore, the two other methods were based on extracting plain-text strings and byte sequences from the data set, where each string and byte sequence was used as a feature. Experimental results showed that detection rate was highest (97 %) with naive-Bayes classification using strings as features.

An approach for detecting obfuscated malware was done by Sung *et al.* [64], where they developed a signature-based detection system (SAVE). SAVE uses PE code as input and feeds the executables to a PE parser which extracts API calling sequences that are mapped to a global 32-bit integer number, where the 16 most significant bits represent a DLL module and the last 16 the specific API calls. This detection system was later used by Xu *et al.* [65]. They assumed that malware contained a sequence of malicious API calls. To test their system against polymorphic variants they manually modified the code by modifying data segment, control flow, and inserting dead code (see Section 2.3). Results showed that the detection scheme was accurate and efficient.

Ye *et al.* [16] developed an Intelligent Malware Detection System (IMDS) for detecting polymorphic and metamorphic malware. This was done by analyzing PE files [60, 61], where each API call represented a feature, and the classification was done by an

Objective-Oriented Association (OOA) mining algorithm [66]. When analyzing unknown malware this approach had a detection rate of 92 %. A similar approach was done by Sami *et al.* [15], where they used every DLL as a feature. Each element of the feature vector corresponded to a DLL library, and the value of each element was equal to the number of API calls. They got the best results using a Random forest [67] classifier, with a detection rate of 99.7 %.

A virus prevention model was presented by Wang *et al.* in [17], where they also extracted features from PE files. Compared to [16, 15], they used a different approach to feature extraction. A tree structure was used to represent the PE file and its DLL call dependencies, and these dependencies were further used as features. Furthermore, a support vector machine (SVM) classifier [68] was used which yielded a detection rate of 99 %. Zou *et al.* [69] also used SVM, however more specific features where chosen such as behavior features on registry, files, memory, processes, network etc. The results showed that the system yielded highest classification accuracy with approximately 1100 API calls.

Wang *et al.* [70] developed a malware detection system that was based on analyzing representative characteristics and systematic description of the suspicious behaviors of malware. They defined suspicious behavior as a sequence of API calls. Different types of suspicious behavior were defined such as searching files to infect, modifying file attributes, modifying registry etc. Bayes algorithm [68] was used to detect the flow of suspicious behavior with a detection precision of 94 %.

2.6 Dynamic Malware Analysis

This section will present background material regarding the techniques that are applied in dynamic malware analysis. As with the section on static analysis we focus on what type of malware characteristics or features the methods utilize in the analysis. Malin *et al.* [25] defined dynamic malware analysis as:

> “Dynamic or behavioral analysis involves executing the code and monitoring its behavior, interaction, and effect on the host system.”

When executing a suspicious executable it is crucial with an isolated environment to make sure of no propagation opportunities if the executable turns out to be malware. The simplest method to achieve this is to sacrifice a host without any network connection and see what happens. With additional tools to monitor system activity it is possible to get a clear understanding of the suspicious executable’s behavior. This analysis approach is also immune to the previous mentioned obfuscation techniques, since the suspicious file is executed [71].

Important aspects when analyzing a suspicious executable with this approach is to identify the nature and purpose of the executable, how it interacts with the host system and network, how and to what extent it compromises the system or network. The following sections will present manual dynamic analysis in more detail. Furthermore, performing analysis in sandbox environments and advanced methods in tracing API calls will be discussed.

2.6.1 Dynamic Malware Forensics

When employing dynamic analysis methods on malware or suspicious executables you have a wide set of tools available. For dynamic analysis the most relevant tools include monitoring capabilities. Malin *et al.* [25] have given the following steps for dynamic analysis of suspicious executables:

1. Sacrificing a physical host for analyzing malware may not be an efficient approach. Instead, the analyst should consider using a virtualized host such as *VMware* [14]. By using *VMware* to create virtual machines you can easily restore the virtual machine to its previous state by using snapshots [72]. Additionally, monitoring tools for file integrity is a valuable technique to study changes to the file system, registry and configuration files[4]. Installation monitors are tools that serve as a loading mechanism that tracks all changes done by a suspicious executable.

2. Learning how malicious executables interacts with a system are crucial in order to identify how damaging the malware is. This step involves setting up your environment for both *passive monitoring* and *active monitoring*. Passive monitoring applies file integrity and installation monitors to compare snapshots before and after an infection. Active monitoring tracks activities in real-time for processes, files, registry, network and API calls. Note that there is a wide variety of tools available and the most common are listed in [25, 30].

3. When the environment is deployed you need to take a snapshot of the current state before executing the malicious file. There are different techniques in executing the malware and the choice may depend on the type of tools that are utilized in the previous step. For example, using an installation monitor to capture changes to the host system when it is executed, or tracing the calls and requests of the malware using an API monitor (see Section 2.6.2). Moreover, it is important to monitor the network activity for investigational purposes. In the context of botnet malware it will connect to the C&C server(s) and may lead the investigators to the botmaster. *Wireshark* [73] is a popular tool for this purpose.

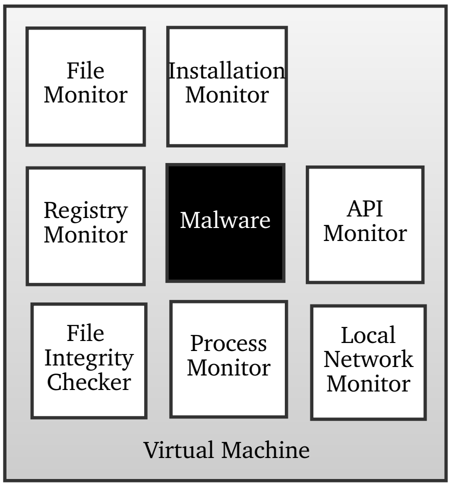


Figure 10: Dynamic analysis system setup

The system setup for dynamic analysis is shown in Figure 10. Note that additional virtual machines can be added to serve as additional network monitors such as a network

[4] Configuration files are typically stored as *.ini files* in Windows.

sniffer, intrusion detection and honeypot. This approach to dynamic analysis is very generic and will suit investigation of suspicious executables dealing with a small amount of malware, however, with a large number of files other analysis approaches are preferable.

2.6.2 Application Programming Interface Tracing

Tracing API calls and/or system calls are an extension to what has been presented in Section 2.5.3, where the main difference here is that the function calls are retrieved while the executable is running, hence a dynamic analysis approach. This feat is achieved by intercepting the called functions from the running executable to the operating system. For example in Linux we have a built-in tool called *strace*. This information will serve well for the analyst related to system/network activity.

Christodorescu *et al.* [74] developed a prototype based on extracting malicious behavior from malware. They built a graph-representation of the malware behavior that was based on system calls and their dependencies. This information was used to compute the difference between malicious and benign software. Hu *et al.* [75] had a similar approach to malware detection, where they employed an API tracer to extract features from six predefined classes of malicious behavior. These classes were related to file, window, process, register, network and service. A 35-dimensional feature vector where used and experiments showed that the number of captured features influenced the detection rate. Six features or more gave the best detection rate. Furthermore, this work was continued by Ding *et al.* [76] where they extended the framework by applying a statistical detection model and a mixture of expert model. Results showed that the statistical detection model yielded the highest detection rate.

Medusa is another system employing API tracing, which was developed by Nair *et al.* [77]. It generates signatures based on entire malware classes (families), where principle component analysis [68] was used to get the critical API calls from each malware family. Then statistical measures were utilized to distinguish the malware families from one another. However, because of a small dataset the classifier was not accurate enough, which implies that samples were assigned to the wrong family. A similar approach was done by Park *et al.* [78], however, they used samples from different malware families instead of using malware generator kits. Their classification scheme was based on intercepting system calls during malware execution using *Ether* [79], and built a graph that represented this behavior. For classification they calculated the similarity between graphs of different malicious and benign software. Experiment results showed that some malware families had two distinct types of behavior.

Ahmed *et al.* [80] also used an API tracer to retrieve statistical features from spatial and temporal information available in the API calls. Furthermore, for the classification task they used a Markov chain [81] and got a detection accuracy of 97 %.

2.6.3 Virtual/Sandbox Environment

Analyzing malware by executing them in a closed and secure environment could yield other behavioral information than that from a static analysis. Sandboxes is a technique, first popularized by *Java* [82], that will set constraints to memory space with low privileges and limit access to services. Furthermore, the main advantage with sandboxes is that the analyst has full control over the memory space and can control the execution step by step [6]. Compared to virtual machines, sandboxes have restricted resources, since virtual machines can emulate any hardware or software resources. A common flaw with both approaches is that they can be detected by an executing malware [83].

Rieck *et al.* [84] used a sandbox environment for automatic classification of malware. A commercial sandbox was used, called *CWSandbox* that was developed by Williems *et al.* [71], that generates a detailed report based on the malware's run-time observations. *CWSandbox* executes the file in a controlled environment and monitors function calls towards the Windows API by using API hooking and DLL injection. API hooking is a technique to access the API functions, and it does this by intercepting the calls to a function. If this is done properly, the malware cannot detect whether the function call has been intercepted or not. Furthermore, DLL code injection is a technique that is used to run code within the address space of a process forcing it to load a DLL, and here load a DLL with the API hooking functionality.

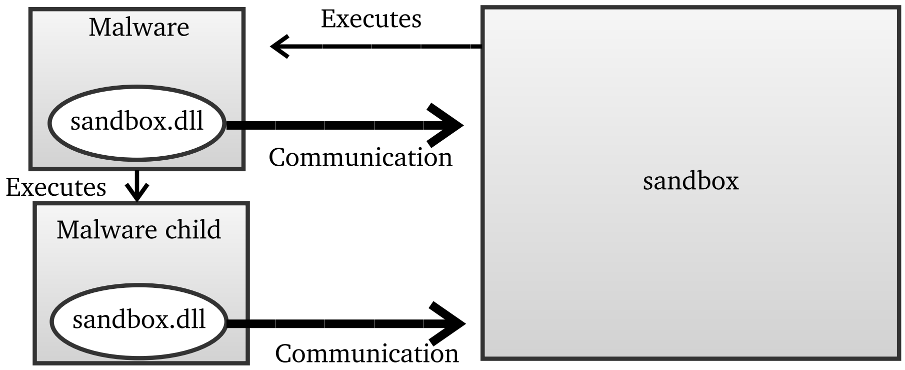


Figure 11: Sandbox with API hooking and DLL injection capabilities

By applying this sandbox for malware analysis it will report on activities related to file creation/modification, changes in registry, DLLs loaded before execution, virtual memory areas accessed, processes created and network information. Rieck *et al.* used operations reported by *CWSandbox* as features, and used these to train a SVM classifier. During an experiment with 3000 previously undetected malware executables the classifier assigned 70 % of the malware with a correct label. A later framework was proposed by Rieck *et al.* in [85], where it was used to automatically identify novel classes of malware by applying hierarchical clustering [68] based on their behavior. By further applying nearest prototype classification [86] of unknown malware, it can assign malware to known classes of behavior.

Another sandbox that has been used for classification and clustering is *Anubis* [87]. The main difference compared to *CWSandbox* is that it utilizes a PC emulator instead of virtual machines. Thus, *Anubis* will simulate a personal computer (processor, graphic card, hard disks etc.) in software and will not execute instructions directly on the real processor as in the case with *CWSandbox*. Bayer *et al.* [88] used *Anubis* to analyze a large collection of malware and applied hierarchical clustering to identify subsets of malware with similar behavior. Features were extracted from behavioral profiles that described the runtime activity of the malware such as system calls and their dependencies and network activities. Firdausi *et al.* [89] also used *Anubis* and they extracted features from the generated XML-reports.

Bailey *et al.* [90] implemented an automated malware classification system that executed the malware in a virtualized environment (*VMware* [14]) and used the state changes of the infection such as file modification, processes creation, and network connections as features. Additionally, they applied single-linkage hierarchical clustering with good results. Another approach for measuring the similarity of malware behavior was done by Apel *et al.* [91]. *CWSandbox* was also applied for this approach, however the focus in this experiment was to apply various distance measures and study their influence in a hierarchical clustering structure. Moreover, they evaluated, among others, a distance measure used for malware classification by Bailey *et al.* [90], the edit distance and normalized compress distance. Nevertheless, the best results were acquired when they applied the manhattan distance.

A recent paper by Burji [20] presented a case study of dynamic malware analysis of the *Nugache* worm, which is often used in botnets. Besides dynamic analysis, they combined static malware analysis tools such as *IDA Pro* [92] for analysis of information such as API calls and DLL file references. This retrieved information was later used as part of a feature set that also included URL references and registry entries. No classification task was documented in this paper only the generation of decision rules using the *BLEM2 machine learning tools* [93].

3 Machine Learning and Data Mining

So far, different methods in machine learning and data mining have been discussed in a malware analysis perspective. These methods prove to be valuable tools when dealing with large datasets that contain both malicious and benign software. Here we want to find some common characteristics in order to distinguish benign from malicious software. In this chapter we will take a closer look at several important aspects in machine learning and data mining such as features and feature quality, preprocessing methods, and the differences between classification and clustering.

3.1 Machine Learning and Data Mining Introduction

Machine learning and data mining are two fields that are closely related. Machine learning is a scientific sub-field of artificial intelligence, with a broad area of applications such as data analysis, knowledge discovery, game playing etc. A common property within these areas is the focus of learning to recognize complex patterns; only then the system can make clever decisions based on the analyzed data [94]. The following definition of machine learning is based on Sergios Theodoridis and Konstantinos Koutroumbas [95]:

> "Machine learning is the scientific discipline whose goal is the classification of objects into a number of categories or classes."

On the other hand, data mining is an interactive and iterative process that applies machine learning methods in order to achieve their goal. Often data mining is associated with knowledge discovery which is concerned with finding and structuring information from large datasets (e.g., databases). Data mining is defined by Hand *et al.* [96] as:

> "Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner."

In the rest of this section we will give a general introduction to machine learning and data mining, by discussing common methods, areas of application and challenges.

3.1.1 Machine Learning

Machine learning is, as mentioned above, part of the artificial intelligence field, where the major concern is to "learn" from a specific type of data. How a machine learns in this context is divided into two categories, namely *supervised learning* and *unsupervised learning*. In supervised learning we have a label associated with each observed object and it is the algorithm's job to assign a label to each object based on some measurement on the object's characteristics. On the other hand, with unsupervised learning we do not have any labels. In this case the algorithm will group objects based on their similarities [94].

Algorithms that are based on supervised or unsupervised learning operates on a dataset which is built from characteristics of observed objects. These object characteristics are represented as features[1] which play an important part when choosing for example the

[1] In some literature referred to as attributes or properties.

type of machine learning algorithm [68]. The process of measuring an object's features is called *feature extraction*. These features which are retrieved by the feature extractor are stored in feature vectors, making up a feature set[2]. In fact, choosing good features which are later represented as a feature set, will have a positive impact on the learning outcome. Furthermore, what is a good feature? This depends of what you want to discover from the learning task. For example, in the context of malware detection you want to extract features that distinguish benign from malicious software. Methods for measuring feature quality is elaborated in Section 3.2.

Classification is part of the supervised learning category, meaning that a supervising variable provides a class label for each classification problem. The classifier is then trained to use the feature vectors from the feature extractor to assign objects to a suitable class. A typical classification problem is done by first comparing the object to all the predefined classes with a distance measure and assigns the object to the nearest class. Thus, it is probable this assignment is incorrect and a similarity measure to a reference object for that class will be done to validate this. Additionally, when classifying an object based on continuous features, the classifier needs to map these continuous features from feature space to class space, where the values are discrete. This mapping function may be given in advance or learned from the data, which is possible since the data consists of training samples which describe previous solved past problems (training data). Common classifiers are decision trees, naive-Bayes classifiers, Bayes networks, decision rules, nearest neighbor classifiers, linear discriminant functions, logistic regression, support vector machines and artificial neural networks [68].

When the learning procedure does not have a supervising variable, meaning that only the object's features are given, we apply unsupervised learning procedures. *Clustering* is one of the methods of unsupervised learning and by far the most popular one. The main task of the clustering algorithm is to form subsets/clusters based on the similarity of object features, and hence aiming to find some structure in a set of unlabeled data. Additionally, the number of clusters can be predefined as background knowledge or can be determined by the learning algorithm. Common learning algorithms for clustering are hierarchical clustering and partitional clustering [95].

Another supervised procedure is *regression*, where the purpose is to find some functional description of data. This often involves predicting values for new input and this is the task for the regressional predictor. In regression we have a set of objects, which have several independent and observable features (which are either discrete or continuous). The regressional predictor maps values with a continuous function from feature space to prediction values. As with classification, this function can be predefined or learned from previously solved problems. The most common regressional predictors are regression trees, linear regression, locally weighted regression, support vector machines (for regression) and artificial neural networks [68].

A typical machine learning process is shown in Figure 12. This process consists of five phases, where the first phase deals with data acquisition, meaning that we collect the data that is going to be analyzed (e.g., using a honeypot). The second and third phase deals with monitoring the data samples and extract relevant features (e.g., using a sandbox). When these phases are done we can utilize the collected features in the learning and classification procedure. Finally, the last phase is the evaluation of the classifier.

[2] In some literature referred to as feature space.

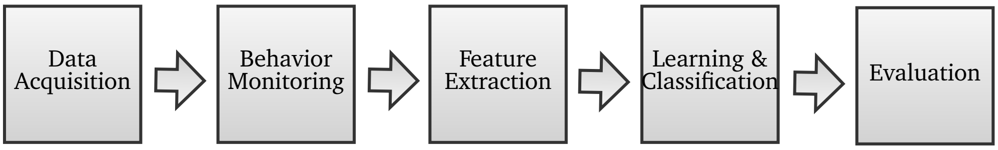


Figure 12: Machine learning process example

3.1.2 Data Mining

Data mining is a discipline which is an intersection of different fields such as statistics, machine learning, data management and databases [96]. Often data mining is associated with knowledge discovery which is an interactive and iterative process used to find and structure information from large data sets [94].

There are two terms in data mining that is worthy of noticing; namely, *descriptive modeling* and *predictive modeling*. A *descriptive model* presents the most important aspects of the data, which is mainly a summary of the data that enables us to gain further knowledge. An example that falls in this category is cluster analysis that groups data objects based on their feature similarities, see Section 3.4. On the other hand, *predictive models* are designed to predict or forecast the outcome of a data mining process based on previously known characteristics of the observed data. Typical examples of predictive modeling are classification algorithms that assign a class label to an observed object based on feature measurements, and regression that predicts values of new input to the algorithm.

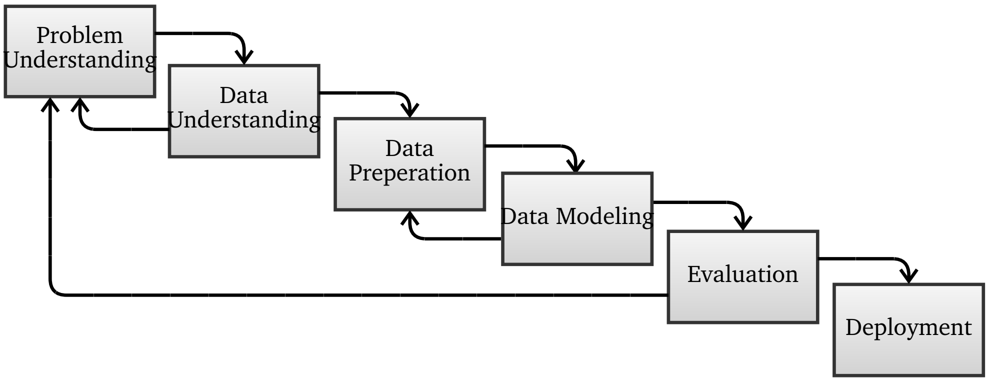


Figure 13: Data mining process example

Figure 13 shows a popular data mining process called Cross Industry Standard Process for Data Mining (CRISP-DM) [97, 94]. The following list will give a short description of the 6 different phases of the process:

- *Problem understanding* focuses on project objectives to further convert this knowledge into a data mining problem.
- *Data understanding* starts with collection of initial data. This is done to gain initial knowledge about the data that is going to be analyzed.
- *Data preparation* is the phase where you construct the dataset from the collected

data. This phase will include aspects such as feature extraction and feature selection.

- *Modeling* starts with selecting various modeling methods. Some methods require certain representation of the data set (e.g. discrete features values). Thus, it may be necessary to take a step back to the data-preprocessing phase.
- *Evaluation* phase focus on evaluating the previous used model. Dependending on the objectives, the different evaluation criteria may be related to performance, accuracy etc.
- *Deployment* is the last phase where the model is implemented and utilized.

Compared to the machine learning example presented in the previous section shows that the approaches are clearly similar. However, as stated by Witten and Frank [98], the process of data mining is a more practical approach. Therefore, simply put, data mining employs learning in a practical manner.

3.1.3 Applications

There are several application areas which use machine learning and data mining in some various form. For example companies or institutions that possess huge volumes of data and want to gain knowledge from this information [98, 94]. The marketing and sales domain is perhaps the most popular area that has been utilizing data mining and focusing on predictive modeling. Data mining can here be used to predict groups of people that are suitable for specific services. For example in the context of cellular phone services, where the companies try to find suitable groups of people that they offer a type of subscription that would be beneficial for both parts.

Another domain for machine learning is when you need to come up with the correct diagnosis. This relates to fields such as medicine, where records from previous patients, that is treated from a similar disease, can be used to induce knowledge [94]. This information can further be used to give a diagnosis to new patients. A different field which employs diagnosis is the preventative maintenance of industrial equipment. This is a field where engineers and technicians have decades of experience, which means that they can give a diagnosis when faults arises, or even better predict when faults may arise due to for example wear rate and how long a certain part has been used.

Furthermore, in the context of malware detection and analysis, data mining and machine learning are valuable tools. As described in Chapter 2, data collected from malware activity such as malicious network traffic and host-related activity can for example be used to analyze whether your system has been infected and/or is under attack by a known or a new threat.

3.1.4 Challenges

When employing machine learning and data mining methods there are different challenges to be aware of. Jain *et al.* [81] presents two common pitfalls which will influence the chosen machine learning algorithm's performance; namely the *curse of dimensionality* and *overfitting*.

The *curse of dimensionality* is a situation where the number of features is too large relative to the number of data samples. Two possible solutions for this problem are to acquire more samples and dimension reduction. Acquiring more samples may not be possible, so the only option is to apply techniques for dimension reduction (see Section 3.3). Be aware that this approach may lead to loss of discrimination and thus may be less accurate. On the other hand, *overfitting*[3] may occur when a classifier is too optimized on the training data[4], by for example describing insignificant relationships of the data (e.g., random errors or noise). A solution for this problem is to always use an independent test set for evaluation. Thus, this dataset should not be part of the classifier's training.

Additionally, when evaluating classifiers there is an aspect worthy of notice. The *no free lunch theorem* [68] states that there are no context-independent or usage-independent cause to favor one classification algorithm over another. This means that if we have the situation where one algorithm seems to outperform another, it is because the specific machine learning problem and not the superior characteristics of the algorithm.

3.2 Features and Feature Quality

Whether you are planning to use machine learning algorithms based on supervised or unsupervised learning you must focus on the characteristics of the observed objects/-samples. These characteristics are represented as features and will influence the outcome of the machine learning task. The following definition is based on Wittend and Frank [98]:

> "The value of a feature for a particular instance is a measurement of the quantity to which the feature refers."

After the feature extraction phase it is necessary to proceed with one of the most crucial tasks in machine learning. This task relates to the quality evaluation of the features. To be able to perform a quality evaluation there exists several measures that estimates the feature's usefulness in the classification task [94].

The following sections will give an introduction to the typical feature types and roles. Later, various feature quality measures will be discussed.

3.2.1 Feature Roles and Types

Features can have different roles in a dataset. A role that has been mentioned previously is *label*. This role is used in supervised learning to describe what group a specific feature belongs with. For example in malware detection and analysis a label distinguishes malicious and benign executables. Thus, labels can be predicted with classifiers for samples that have not yet been assigned with a label. Also, there are roles that are used as an unique identifier (*ID*) for the concerned samples. An example here is hash sum (e.g., MD5) which is often used in malware detection and analysis (see Section 2.5).

Other roles are *weight* (or feature weighting), where a high weight is assigned to the most relevant features and improves the classification scheme, since you can discard features below a specified threshold. Finally, features that do not have a specified role are used to describe the analyzed sample. For these features we can, for example, store activities done by the malware on a system.

Other feature characteristics that are worth noticing is feature types. The number of feature types supported in a classification scheme depends on which type of tools you are using (e.g., Weka [99]). There are in general 3 different feature types that are used:

- *Nominal* which is used for discrete values or for a value set with few options (e.g., yes/no).

[3] In some literature called overtraining.

[4] In some literature referred to as learning data.

- *Numeric* which is used for continuous values.
- *String* which is used to hold a list of characters of arbitrary size.

It is worth noticing that certain classification or clustering algorithms require that the features are converted to a specific type. This is often required since the algorithms are designed for a particular type of input. Converting to another feature type is commonly done in a preprocessing step and are further discussed in Section 3.3.

3.2.2 Feature-Quality Measures

Feature-quality measures are an important step in machine learning. Here we wish to evaluate how useful a feature is, for example, to predict the label of a target sample in the context of classification. It is the learning algorithm's task to search the hypothesis space[5] to evaluate the quality of each feature for the specific learning problem [94]. The three types of quality measures that are going to be presented in this thesis is impurity functions, information gain and minimum description length, which are described below.

Impurity Functions

One of the basic measures for evaluating the feature quality in classification problems are impurity measures. In general, impurity measures define how well classes are separated. Suppose the number of classes C_k is n, so we have $k = 1, 2, ..., n$. Then the impurity measure is a function of the probabilities $P_1, P_2, ..., P_n$. An impurity measure is a function φ which satisfies $P(C_k) \geq 0$ and $\sum_{n} P(C_k) = 1$, with following properties [94]:

1. φ achieves single maximum, when $P(C_k) = 1/n$.
2. φ achieves minimum, when $P(C_k) = 1$, $P(C_l) = 0$, $l = 1, 2, ..., n$, and $l \neq k$.
3. φ is a symmetric function of $P_1, P_2, ..., P_n$.

Given the function φ and feature F_j, the impurity measure $i(f_j)$ is as follows:

$$i(f_j) = \varphi(P(C_1), ..., P(C_n)) \tag{3.1}$$

Furthermore, to estimate the quality of (discrete) features $q(f_j)$, the decrease of impurity is subtracted from the impurity measure:

$$q(f_j) = i(f_j) - \sum_{m=1}^{n} P(V_m)\varphi(P(C_1|V_m), ..., P(C_n|V_m)) \tag{3.2}$$

Here the features have a set of possible values $V_j = v_1, v_2, ..., v_n$. The prior probabilities of classes $P(C)$, feature values $P(V_j)$, and conditional probabilities $P(C|V_j)$ are estimated from the distribution of training samples.

Entropy and Information Gain

In machine learning, most measures are based on information content, which can be used to determine the outcome of the hypothesis space of a classifier [94]. The amount of information[6] (I) used to determine the outcome (X_j) is defined as the negative logarithm of its probability:

$$I(X_j) = -\log_2 P(X_j) \tag{3.3}$$

[5] A set of all possible hypothesis/model outcomes.

[6] In this context amount of bits.

From this equation we can derive the *entropy* (H) by calculating the average amount of information needed to determine the outcome:

$$H(X) = -\sum_{j}^{m} P(X_j) \log_2 P(X_j) \tag{3.4}$$

Information gain (IG) is a basic feature quality measure that uses the entropy as an impurity function. This measure is defined as the amount of information, gained from a feature, used to determine the class. Before we give a definition of the information gain, some notations need to be introduced:

- n is the number of training samples.
- $n(C_k)$ is the number of training samples from class C_k.
- $n(V_j)$ is the number of training samples with j^{th} value of the given feature f.
- $n(C_k, V_j)$ is the number of training samples from class C_k and with j^{th} value of f.

Then the definition of information gain can be defined as:

$$IG(f) = -\sum_{k} \frac{n(C_k)}{n} \log \frac{n(C_k)}{n} - \sum_{j} \frac{n(V_j)}{n} (\sum_{k} \frac{n(C_k, V_j)}{n(V_j)} \log \frac{n(C_k, V_j)}{n(V_j)}) \tag{3.5}$$

Minimum Description Length

The minimum description length, or MDL principle, is based on the idea that any regularity in a dataset may be used to compress it [100, 98]. Here, a compression means that there are a minimal number of symbols used to describe the dataset. Hence, the more regularity found in a dataset, the more we can compress the dataset.

MDL is often described in the context of data transmission through a communication channel [94], where both sender and receiver know the number of values m of a specific feature f and the number of classes m. Furthermore, the sender and receiver also know the values of a feature f for each training sample. However, the sender is the one who knows the correct class of each sample, where the task is to send the shortest possible message with the classes of all samples to the receiver. To send this message with the features and corresponding classes, the feature f is used in such a way that classes are coded separately for each value of the feature. This requires that each value of the feature correspond to a different class distribution. Hence, the receiver can decode the classification from each feature value.

Since we know the class distribution of each feature value, it is now possible to evaluate its significance in the classification task. Henceforth, the different possible distributions of n samples over m is:

$$\binom{n+m-1}{m-1} \tag{3.6}$$

Furthermore, the amount of information (in bits) is given by the logarithm of all possible classifications of n training samples added to the logarithm of possible distributions (Equation 3.6):

$$MDL_{prior} = \log \binom{n}{n(C_1), ..., n(C_m)} + \log \binom{n+m-1}{m-1} \tag{3.7}$$

Then the estimate of information necessary to coding the classes, is given by the values of the feature f, is the sum of all feature values:

$$\mathrm{MDL}_{\mathrm{post}}(f) = \sum_{j} \log \binom{n(V_j)}{n(C_1, V_j), ..., n(C_m, V_j)} + \log \binom{n+m-1}{m-1} \tag{3.8}$$

The quality $\mathrm{MDL}(f)$ of a feature is defined as the compressivity of feature as shown in the equation below:

$$\mathrm{MDL}(f) = \frac{\mathrm{MDL}_{\mathrm{prior}} - \mathrm{MDL}_{\mathrm{post}}(f)}{n} \tag{3.9}$$

3.3 Data-Preprocessing Methods

Different machine learning algorithms requires different input to function optimally. For example some machine learning algorithms only work with binary, discrete or continuous feature values. Others may give an unsatisfactory result when dealing with a high dimensioned feature space. Fortunately, there exist many preprocessing methods that take care of the mentioned challenges. The following definition of data preprocessing is based on Pyle [101] and Kotsiantis *et al.* [102]:

> "Data preprocessing deals with approaches for preparing the features before they are applied to a machine learning algorithm."

In the following sections we will discuss the basic feature conversations such as discretization of continuous features, and discrete to continuous features. Also, challenges related to how we deal with missing or unknown feature values will be presented, followed by approaches for reducing the amount of features.

3.3.1 Feature Discretization

There are machine learning algorithms that are designed for only using discrete features as input. Hence, if our dataset consists of continuous features we must employ a discretization scheme either beforehand, or during the training. In short, discretization of continuous features means that the feature values are split into a finite number of intervals that are treated as discrete feature values [98]. The main challenge with discretization of continuous features is information loss for each feature. This situation arises since specific feature values from the same interval cannot be discriminated any more, or values falls between two intervals. Thus, there are two important aspects for the discretization algorithm to handle:

- Optimal numbers of intervals
- Optimal boundaries for each interval

Methods for feature discretization are many, both unsupervised and supervised variants exist. When applying unsupervised discretization, there are three implementations which are often used. These are equal-interval binning, equal-frequency binning and proportional k-interval discretization [98]:

- Equal-interval binning splits the whole range of feature values into intervals of equal size. This may result in very uneven distribution, where some intervals contain many instances, and others none.

- Equal-frequency binning applies intervals that contain equal number of feature values. This variant may cause bad boundaries for example when two classes are related to two separated intervals and we have an uneven set of feature values used to distinguish between two classes.

- Proportional k-interval discretization utilizes equal-frequency binning with naive-Bayes (see Section 3.4), where the number of intervals is chosen in a data-dependent form.

Additionally, supervised discretization takes the classes into account when setting the intervals. Here there are two general variants that are often used, namely buttom-up and top-down methods [94, 102].

- In the bottom-up variant, the discretization algorithm starts with as many intervals as there are training samples, an iteratively merge neighboring intervals based on the most similar class probability distributions. This process stops when the quality $q(f)$ of the discretized features f stops increasing, or we have two intervals left.

- When using the top-down variant the discretization process starts with a single interval, and for each step this is split into smaller intervals. The interval is split at the point where the quality $q(f)$ is maximized. This process stops when the quality $q(f)$ of the discretized features f stops increasing

There are different methods to control the discretization processes of continuous features, where any quality measure can be used (see Section 3.2.2). A good candidate for this task is MDL that do not overestimate features with multiple values [94].

3.3.2 Missing/Unknown Feature Values

Often when we extract features from a raw dataset it is quite common to find missing feature values. In malware analysis context this situation may occur when the malware sample is protected by obfuscation techniques where we are not capable of extracting certain values. The most usual way of dealing with this issue is either to ignore them or to replace them with a specific scheme. Common schemes of replacing missing values are replacing the missing values with zeros or calculate its value based on probability distribution of feature values [95].

3.3.3 Feature Selection

When dealing with a high-dimensional feature set, not all the extracted features may be of relevance for the phenomena of interest, for example classification task. We may end up in common pitfalls, namely the *curse of dimensionality* and *overfitting* (see Section 3.1.4). Feature selection will allow a more efficient machine learning procedure by selecting an optimal subset of features [94]. This is necessary if the feature set consists of too many features that are irrelevant, correlated, or redundant which may reduce the classifiers performance.

There are two general approaches for feature selection. The first is to make an independent analysis based on the characteristics of the data. This approach is called *filtering*, where the feature set is filtered to yield the most promising feature subset. The other approach, *wrapper* method, tries to find the optimal subset using a machine learning algorithm with an internal cross-validation. Here the algorithm is wrapped into the feature selection procedure [98]. Thus, this is a more advanced and slower method compared to the filtering approach.

Correlation-Based Feature Selection

Correlation-based feature selection (*CFS*) is a filtering approach that evaluates subsets of features based on the assumption that "...useful feature subsets contain features that are predictive of the class but uncorrelated with one another" [103].

When a feature set is applied to CFS they will first be discretized if they are continuous. Then a heuristic measure of the "merit" is computed that use a feature subset from from pair-wise feature correlation. Furthermore, heuristic search [95] is applied to traverse the feature (sub)sets, that yields the subset with the highest merit. The computation of the merit is shown in Equation 3.10.

$$\mathrm{Merit}_{S_k} = \frac{k\overline{r_{cf}}}{\sqrt{k + k(k-1)\overline{r_{ff}}}} \tag{3.10}$$

Here the equation gives the merit of the feature subset S that consists of k features. $\overline{r_{cf}}$ is the average value of all feature-classification correlations, and $\overline{r_{ff}}$ is the average value of all feature-feature correlations.

Moreover, the CFS algorithm has been improved in a generic feature-selection algorithm (GeFS) by Nguyen *et al.* [104, 105], by introducing a minimal-redundancy-relevance (mRMR) measure [106]. Here, the CFS optimization problem is transformed into a polynomial mixed 0-1 fractional programming problem. By further introducing additional variables, a new mixed 0-1 linear programming problem is obtained. This includes a number of constraints and variables that are linear in the number of features that can be solved by a branch-and-bound algorithm [94]. Thus, the GeFS algorithm is capable of removing both irrelevant and redundant features.

3.4 Classification vs Clustering

In Section 3.1 a brief introduction of the two terms classification and clustering was given. This thesis will compare several classifiers in different scenarios and therefore this section will emphasize more on the classification part. First, a detailed description of classification with chosen algorithms is discussed. Next, a short introduction to hierarchical and partitional clustering is given. Finally, main differences between classification and clustering are discussed.

3.4.1 Classification

Classification is in the supervised learning category, where we have a label associated with each sample, and it is the classifier's job to assign the correct label to each sample based on the sample's features. An example of the different phases during classification is shown in Figure 14. In general, during the training phase the classifier is fed with training samples that are labeled. Here the classifier learns the discriminative differences between samples with a specific label from samples with another label. This knowledge is used in the classification phase, where the classifier is fed with unlabeled samples. Now the classifier predicts the label to the sample based on some feature measurements. There exists many classification algorithms and the ones applied in the experiments are presented below.

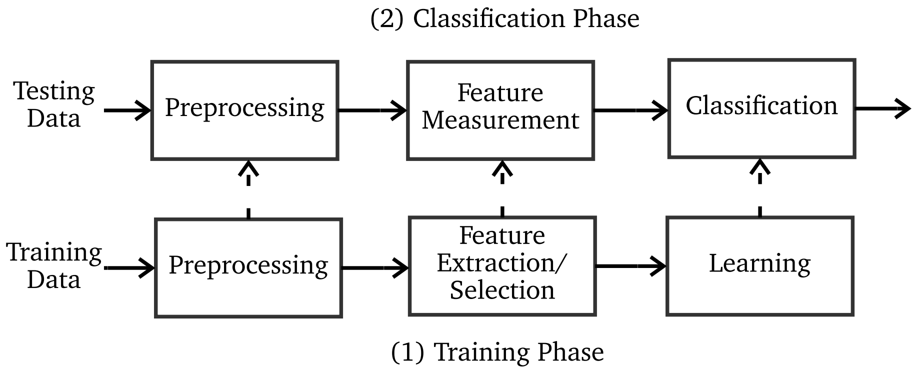


Figure 14: Different phases during classification

Naive-Bayes

The naive-Bayes classifier is based on Bayes decision theory (Bayes formula) that describes how to calculate the inverse probabilities [95]. It states that by knowing the conditional probability of the feature vector V given the class C_k ($P(V|C_k)$) you can calculate the conditional probability of class C_k given the feature vector V ($P(C_k|V)$) if you know prior/unconditional probabilities $P(V)$ and $P(C_k)$.

$$P(C_k|V) = \frac{P(V|C_k)P(C_k)}{P(V)} \tag{3.11}$$

From this equation we can derive the naive-Bayes classifier[7]. This equation is derived with the assumption that there is conditional independence between features with respect to the class. Before providing the naive-Bayes we need to know how to estimate the unconditional probabilities $P(C_k)$ and conditional probabilities $P(C_k|v_i)$:

$$P(C_k) = \frac{n(C_k) + 1}{n + m} \tag{3.12}$$

$$P(C_k|v_i) = \frac{n(C_k, v_i) + \lambda P(C_k)}{n(v_i) + \lambda} \tag{3.13}$$

Both the unconditional probabilities $P(C_k)$ and conditional class probabilities $P(C_k|v_i)$ are estimated by the learning algorithm for each feature value v_i of feature f. Furthermore, $k = 1...m$, $n(C_k)$ is the number of training samples from class C_k, n is the total number of training samples, m the number of classes, and λ is a m-estimator used to estimate the conditional class probabilities [94]. Henceforth, the naive-Bayes is stated as:

$$P(C_k|V) = P(C_k)\prod_{i=1}^{a} \frac{P(C_k|v_i)}{P(C_k)} \tag{3.14}$$

Training in naive-Bayes is therefore reduced to calculating the unconditional and conditional probabilities. Naive-Bayes' strengths is that it performs well where there are no or weak dependencies between features. Thus, it will not give any good results in the case when strong dependencies between features exist. Another strength is that all of its probability estimations are reliable, which means that overfitting will not likely occur.

[7] The derivation of Bayes formula is provided in Kononenko and Kukar [94].

K-Nearest Neighbors

K-nearest neighbors (K-NN) algorithm is the simplest classifier in the nearest neighbors category, where it stores a part or all of the training samples which are later used to predict the class label when a new sample is presented to the algorithm [95]. Thus, given an unknown feature vector and distance measure:

- Identify the k nearest neighbors from the n training samples. This operation is done regardless of the class label. In general, an odd number is chosen for k when dealing with a two class problem.
- Identify the number of feature vectors k_i using the k samples that belongs to class $C = C_1, ..., C_m$, where $k = \sum_i k_i$.
- Assign the feature vector to the class C with the maximum number k_i of samples.

An example of K-NN is given in Figure 15, where the previously applied training samples are a two class problem where the green dots symbolizes benign executables and the red dots malware. The blue dot is the unknown sample. If $k = 3$, which is shown as the first circle, the unknown sample is classified as a malware because this type of training sample has more training samples within the specified distance. However, a different classification arises when $k = 5$ (dotted circle). Now the unknown sample will be classified as a benign executable since this type is overrepresented.

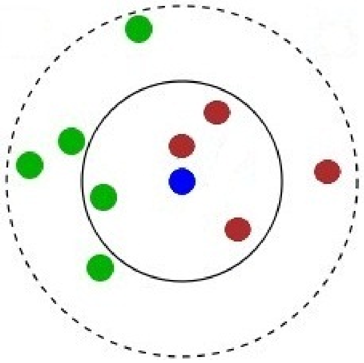


Figure 15: K-NN example

A distance measure needs to be specified in order to decide whether a training sample falls within a specified k neighbor. The *Euclidean distance* [94] is a common distance measure used in a K-NN classifier. Let us consider two samples where each is characterized by a feature vector $u = u_1, ..., u_j$ for the first sample, and $v = v_1, ..., v_j$ for the second sample. Then the Euclidean distance can be defined as:

$$d(u, v) = |u - v| = \sqrt{(u_1 - v_1)^2 + ... + (u_j - v_j)^2} \tag{3.15}$$

This classifier is quite effective if we have a large amount of training samples. This however, leads us to one of its weaknesses which are the calculation complexity, because the distance measure needs to be calculated between the unknown sample and all of the training samples. Its other weakness is setting the k parameter, which often requires a bit of tweaking before an optimal result is acquired [94].

C4.5

The C4.5 algorithm is in the category of decision trees. First of all, a decision tree consists of nodes that reflects the sample's features, where its edges are represented as feature values and its leaves correspond to the class labels. For example in a malware detection scheme we have two leaves that correspond to malicious and benign software.

One issue with decision trees is selecting the best features for representing nodes, which is often called splitting criteria. This is often done by a feature quality measure (see Section 3.2.2). The feature with highest quality will be selected as the root node and its feature values will be used as edges to connect with the next level. This is repeated until a stopping criterion will be met, for example no "good" features left [98].

C4.5 classifier is the successor of ID3, and uses the *gain ratio* as a splitting criteria. The gain ratio (GR) is defined as the information gain (IG) that is normalized with the feature entropy:

$$GR(f) = \frac{IG(f)}{-\sum_{j} \frac{n(V_j)}{n} \log \frac{n(V_j)}{n}} \tag{3.16}$$

Here, n is the number of training samples, $n(V_j)$ is the number of training samples with the j^{th} value of the given feature f. The C4.5 employs *pruning*, which means that the tree is reduced in size because of often unreliable lower levels [94]. This characteristic is one of the C4.5 strengths that avoids overfitting, since "unrealiable" features are left out. Also, because of the splitting criterion, it performs well when the dataset consists of a few highly relevant features. On the other hand, the gain ratio evaluates the quality independently of the features. Hence, with many dependencies between the features the C4.5 will not perform well [107, 108].

Support Vector Machines

Support vector machines (SVM) are one of the most successful approaches for solving classification tasks. A reason for its success is that it utilizes all of the available features, regardless of their relevance [68]. Thus, it is suitable for large datasets which contains a large amount of features, which may be of less importance.

The following example will describe the SVM for a linearly separable two-class problem, see Figure 16. Our goal is to first design an optimal hyperplane which is equally distant from the nearest examples from both classes. Furthermore, the training samples that are nearest the hyperplane are called support vectors, and the distance between the hyperplane and its support vectors is called the margin. Thus, the optimal hyperplane is selected based on the maximum margin.

Now let x_i, $i = 1, ..., n$ be the feature vectors of the training set, X. These features belong to either of the red or green class. Additionally, w is the direction of the hyperplane, w_0 is its exact position in space, and $J(w, w_0)$ the cost function. For every feature x_i we denote its corresponding class label by y_i (+1 for red and -1 for green). When the classes are clearly separable, the classification problem can be stated as:

$$minimize \quad J(w, w_0) \equiv \frac{1}{2}||w||^2 \tag{3.17}$$

$$subject\ to \quad y_i(w^T x_i + w_0) \geq 1, \quad i = 1, ..., n \tag{3.18}$$

It is worth noticing that the main strength of the SVM classifier is that it is capable of transforming the feature space to a more complex one. This is useful if classes cannot be

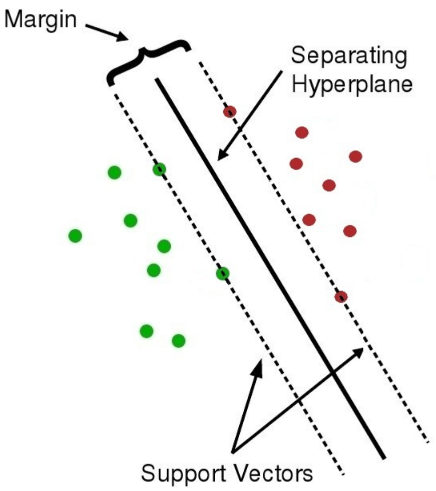


Figure 16: SVM example

separated with a simple linear hyperplane. *Kernel functions* [68] are utilized for the implicit feature space transformation and there exists several to choose from. However, there are no practical method for choosing the best kernel function for a specific dataset [68]. Also, employing a SVM classifier is computational intensive process.

Bayes Network

A Bayes network, is a generalization of naive-Bayes. It is a directed acyclic graph, where the nodes correspond to the features or class labels. In this approach each node is associated with a set of conditional probabilities with respect to the values of the parent features. This means that a node is conditional independent of its non-successors. The conditional probability of each node can be denoted as $P(V_i|A_i)$. Here V_i is the feature associated with the specific node, and A_i is the node's set of parents [95].

When the Bayes network topology is not predefined by an expert that can provide knowledge about the dependencies, it is necessary to build it based on an optimization criterion. The underlying principle in the optimization criterion is the minimum description length (see Section 3.2). Thus, the task of the learning algorithm is to find a topology that minimizes the MDL for a given dataset. When the topology has been decided, the conditional probabilities are estimated from the available training samples [94].

Classifiers based on Bayes network have several strengths. First, they work well on small and incomplete datasets, since it takes into account all available data. Also, when dealing with a lot of features it avoids the curse of dimensionality when estimating the conditional probabilities by only considering the node's parents. This means that the estimations are done in iterations top to bottom. The main weakness of Bayes network is that it cannot handle continuous features, so they must be discretized first [109].

3.4.2 Clustering

Clustering falls under the category of unsupervised learning, where the learning algorithm's task is to group samples into clusters based on unlabeled features. This allows us to find similarities and differences among samples and to derive useful conclusions about them [94, 95]. Similar to classification, there are specific steps when applying a clustering scheme. Here, six basic steps are needed:

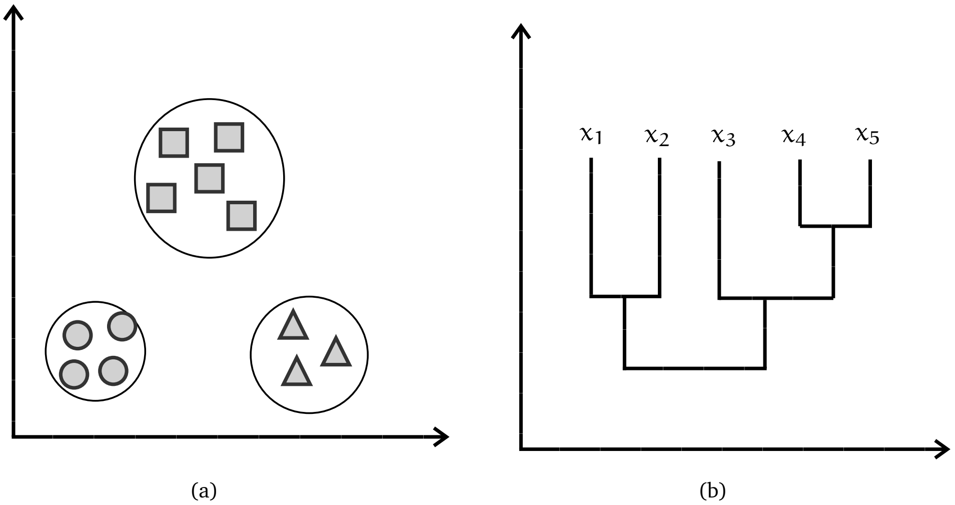
In order to prepare the features for the clustering algorithm we need to perform some (1) *preprocessing*, for example, converting to a supported feature type. Also, by applying (2) *feature selection* techniques it will ensure that the chosen features are highly relevant for the task of interest. A (3) *proximity measure* is applied in order to decide how similar (or dissimilar) two feature vectors are. Additionally, defining what type of clusters that are most sensible for the underlying dataset is stated by a (4) *clustering criterion*. Next step involves applying the (5) *clustering algorithm*, for example, hierarchical or partitional clustering algorithm. Finally, the clustering outcome is (6) *validated* and evaluated to verify its correctness.

Hierarchical Clustering

There are two types of hierarchical algorithms, namely, bottom-up (agglomerative) and top-down (divisive). A standard representation of hierarchical clustering is with a dendrogram (Figure 17 (b)), which is a binary tree where the leaves correspond to each individual sample and the root corresponds to all the samples. When a button-up algorithm is executed it starts with the individual samples and iteratively merges the most similar ones. On the other hand, top-down algorithms starts with all samples in one cluster and iterative splits them into smaller clusters [68].

Partitional Clustering

In partitional clustering, samples are grouped into partitions, see Figure 17 (a). Depending on the specific algorithm, for example, K-means clustering [98], the number of clusters needs to be specified beforehand. This is the general disadvantage with partitional clustering, since the number of clusters will influence the clustering outcome. Thus, it may be necessary to apply the same clustering algorithm several times and evaluate the results to find the best number of clusters that suit the specific task best.



(a) (b)

Figure 17: (a) Partitional clustering, (b) Hierarchical clustering

3.4.3 Main Differences

The main differences between classification and clustering, is the availability of background information. In classification, the labels are known in advance, which is not the case with clustering. Also, the goal of classification is to build a model, which is based on training samples, that predicts the class labels of the test samples only given its features. Clustering does not have a training phase, and the goal is to group samples based on a clustering criterion into different clusters.

4 New Computational Method for Static and Dynamic Analysis

Up to now, we have discussed methods regarding malware detection and analysis in Chapter 2, where we emphasized on static and dynamic malware analysis approaches. Here we studied the different behavior-characteristics that were utilized in order to analyze and detect malware in an effective manner. Furthermore, the theoretical building blocks of data mining and machine learning techniques were presented in Chapter 3.

This chapter presents the system components of the new computational method for static and dynamic analysis. The method will deploy the system as an extension of *deLink* framework by implementing components that also can be utilized independently of the framework. Furthermore, this chapter includes a brief introduction to an overall system description of *deLink*, followed by the theoretical building blocks and practical system implementation of the method.

4.1 deLink Framework

The *deLink* framework by Flaglien *et al.* [19, 3] aims to identify malware evidence across multiple computers. It uses clustering to identify common patterns and correlating data. The main components in *deLink* are; (i) a data collection component that collects information from a computer's file system, (ii) an examination component for processing the acquired data and extracting a set of features based on important characteristics of malware, and (iii) the link mining component that use unsupervised clustering to group files with similar characteristics.

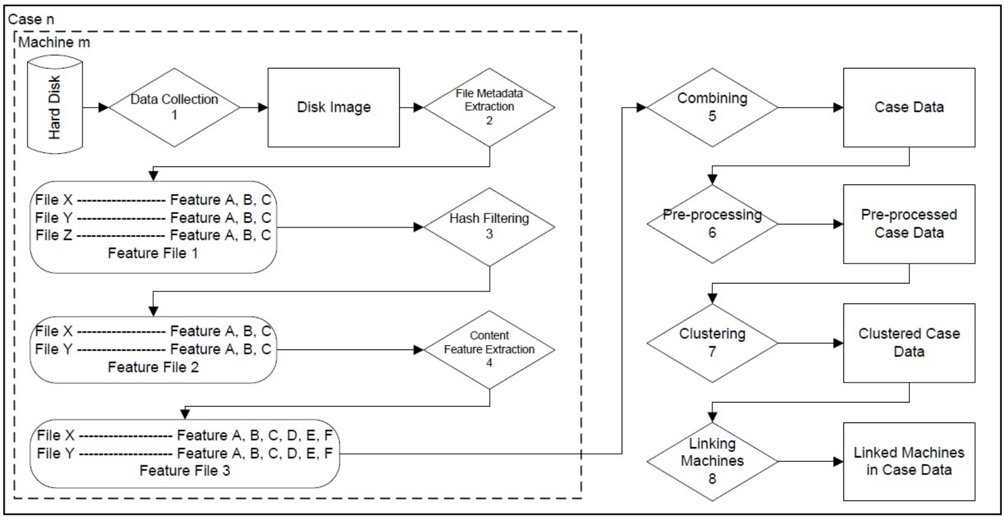

Case n
Machine m
Hard Disk
Data Collection 1
Disk Image
File Metadata Extraction 2
File X ------------------ Feature A, B, C
File Y ------------------ Feature A, B, C
File Z ------------------ Feature A, B, C
Feature File 1
Hash Filtering 3
File X ------------------ Feature A, B, C
File Y ------------------ Feature A, B, C
Feature File 2
Content Feature Extraction 4
File X ------------------ Feature A, B, C, D, E, F
File Y ------------------ Feature A, B, C, D, E, F
Feature File 3
Combining 5
Case Data
Pre-processing 6
Pre-processed Case Data
Clustering 7
Clustered Case Data
Linking Machines 8
Linked Machines in Case Data

Figure 18: Processing steps in *deLink* [3]

Figure 18 shows *deLink's* main processing steps. From each machine a disk image is acquired (1) and the file metadata is extracted (2). This information is used to build a feature file. However, since a disk image will contain a portion of uninteresting files, for example, known and unaltered system/program files, a hash filtering scheme (3) is applied for data reduction. Additional features are extracted (4) by using string searches, such as IP addresses, e-mail addresses and URLs. The final steps (5-8) are necessary to combine all the created feature files, preprocessing the features and analyze the result with link mining.

There are, however, some weaknesses with Flaglien's approach. Using string searches to collect IP addresses, e-mail addresses and URLs from executables will not work in situations where these features are not represented as strings, or where obfuscation techniques are applied, for example packers and encryption. In general, extracting strings from an executable are not very robust as features because they can be easily changed [62]. Also, you must be aware that hackers and malware developers often embed decoy strings in their code to throw digital investigators off track. Instances of false e-mail addresses and domain names are common [25]. Additionally, no method for feature selection was applied to select the most relevant features. Thus, we do not know the weight of significance and the quality of the features.

4.2 Theoretical Method

The theoretical method consists of the basic principles and main elements of the computational method. This method follows a typical machine learning process, where it includes malware forensic approaches for malware analysis. Therefore, it gives an overall understanding of the applied principles and possibilities of improvements in future research.

4.2.1 Preferences and Assumptions

Based on the theoretical background, related to malware detection, malware analysis and machine learning techniques, the following preferences and assumptions are stated:

1. The collected executables must be of portable executable (PE) format and able to run on a Windows-based system.
2. The collected executables must be deployed in such a way that it can be analyzed using the chosen dynamic and static analysis methods.
3. Dynamic and static analysis of the dataset must, in worst case, yield parts of the executables behavior if obfuscation techniques are applied.

4.2.2 System Design

The main system components are designed in such a way that it fits in a general machine learning process. Figure 19 shows the main components of the method, and is further discussed below:

- *Data Acquisition* applies static and dynamic analysis tools in order to gain knowledge about the suspicious dataset. These tools generate reports that are stored before further processing.
- *Feature Extraction* utilizes the generated static and dynamic reports by extracting appropriate features and build a static and dynamic feature set. Also, a combination of these two is built.

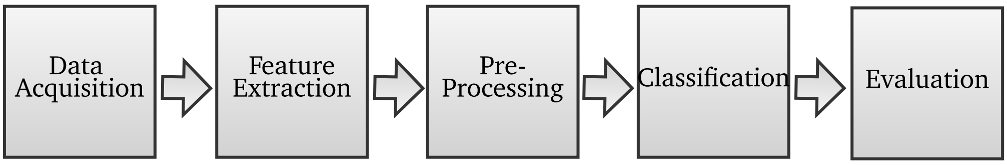


Figure 19: Method's system design

- *Preprocessing* involves feature preparations by ensuring that the feature type is supported by the classifier and removing "unimportant" features to improve the classifier's performance.
- *Classification* will use the preprocessed feature sets by applying different classification algorithms. The reason for applying several classifiers is to find what type of classifier that is most suitable for the extracted feature sets.
- *Evaluation* involves validation of the classification results. This is done by comparing the applied classifiers with measures such as detection rate and accuracy.

4.2.3 Data Acquisition

The data acquisition consists of two steps. First, malware and benign software need to be download and stored as our dataset. In order to acquire an appropriate dataset this thesis will manually download malicious and benign samples. This is done to be sure that the analyzed malware is utilized in a known botnet and that we are able to acquire a suitable dataset in a limited time. Furthermore, to appropriately test the classifiers, benign executables should share some similar behavior when they are analyzed with static and dynamic analysis tools, for example some network-related activities.

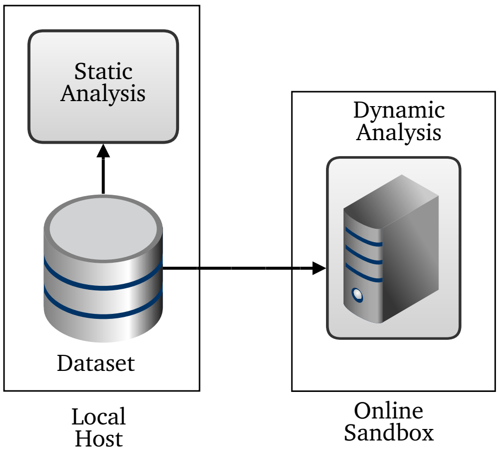


Figure 20: Static and dynamic analysis design

The second step involves analyzing the executables in our dataset, and for this task static and dynamic analysis tools need to be utilized. The static analysis tool will analyze the dataset without executing the files and therefore avoids to infect the system, see Section 2.5. Here the PE format need to be exploited by using a PE parser (e.g., *pefile*), that enables us to retrieve information from the different sections and store this in a static report. The reason for choosing this static analysis approach is that behavior-based characteristics of the files are available without going to a lower abstraction level that requires an analysis of the assembly code. Hence, this approach will not be vulnerable to obfuscation techniques that are applied to the code to confuse the investigator or the disassembler, and have given good results when applied in malware detectors in recent research (see Section 2.5.3).

Additionally, a sandbox (e.g., *Anubis*) is applied to run the executables in a controlled manner utilizing an isolated environment, see Section 2.6. This ensures that executables, with malicious intent, do not have any propagation opportunities. The sandbox needs to emulate a computer instead of a virtual machine that operates by executing instructions directly on the real processor. This will ensure that the malware cannot escape from the emulated environment, and the malware cannot detect its presence as easily as with for example *VMWare* [5, 14]. The purpose of utilizing a sandbox is that the malware will be loaded into memory and executed, to retrieve its system interactions in a controlled manner. Thus, in addition to defeat polymorphism and metamorphism, it will unpack and decrypt malware that is protected by packers and cryptors when it is loaded into memory before execution. Recent research shows that analyzing malware with sandboxes have given satisfying results (see Section 2.6.3).

When utilizing a sandbox the executable's actions need to be monitored and logged, to further produce a dynamic report. All actions stored in the dynamic report will reflect the executable's behavior when it is run on an actual system. The reason for looking at the behavior-based characteristics is to study the behavior of malware and check whether the behavior of malware and benign software can be used to solve a two-class classification scheme. By using benign software with similar system dependencies related to files, registry, and network activities; the chance for a high false positive rate should be lower compared to only utilizing simple benign executables (e.g., *Notepad*). Especially, when the classifier is presented with executables such as browsers, mail clients, network tools etc.

4.2.4 Feature Extraction

During feature extraction, the features describing the executables' behavior need to be extracted from the static and dynamic reports. Moreover, three feature sets need to be built in order to study whether static, dynamic, or a combination of the two, is best suited for a two-class classification problem. Tables 4 and 5 summarize the extracted features from both static and dynamic analysis. It is important to notice that dynamic features are categorized by entities, where each entity consists of a set of features.

Feature	**Value**	**Description**
DLL import	Boolean(s)	Reflects imported DLL in the PE-header of an executable
Funtion calls	String(s)	Reflects all functions called within a DLL

Table 4: Static features used in method

The static feature set needs to be built to handle several DLL names and function call names in order to represent the behavior of each sample.

$$feature\_set_{static} = \begin{cases} dll\_name_1, ..., dll\_name_n \\ dll\_funtion\_name_1, ..., dll\_function\_name_n \end{cases}$$

Here the dll_name represents the imported DLL used in a sample and the $dll\_function\_name$ represent the names of the functions called within a DLL. This implies that the $dll\_function\_name$ holds several feature values:

$$dll\_function\_name = \left(\ function_1, ..., function_n \ \right)$$

Similarly, a dynamic feature set needs to be constructed from the extracted entities in the dynamic reports. Here each sample is represented with multiple entities:

$$feature\_set_{dynamic} = \left(\ entity_1, ..., entity_n \ \right)$$

Here each entity need to hold a set of features:

$$entity = \begin{cases} feature_1 \\ feature_2 \\ \vdots \\ feature_n \end{cases}$$

Since dynamic reports may contain multiple entities of similar type, for example *created file*, it is needed to store multiple feature values for each feature:

$$feature = \left(\ value_1, ..., value_n \ \right)$$

The third feature that we need to build is the combined feature set. This is needed to ensure that both the static and dynamic features are combined into one feature set:

$$feature\_set_{combined} = feature\_set_{static} \cup feature\_set_{dynamic}$$

DLL Dependencies

DLL dependencies are the type of system libraries the executable requires in order to execute. Since the executables satisfy the portable executable standard, the required libraries will be available on a Windows-based system. However, the type of libraries applied is dependent on the type of Windows version the executable is designed for. Both the static and dynamic analysis tools are able to retrieve type of libraries the executable is dependent on, where the PE parser also extracts the different function calls that are used within each library.

Registry Activities

Retrieving features reflecting the registry activity is needed to reveal configuration settings applied by the executables. This type of activity is especially relevant for malware that makes sure that they will run after a reboot by adding autorun keys and values to the Windows registry [110]. Additionally, malware can use the registry to make sure that certain services will be disabled/enabled and to open ports. For example disable the antivirus service and enabling remote login. Thus, typical registry mechanisms that malware exploits need to be retrieved by the sandbox, see entities 2-4 in Table 5.

	Entity	Description
1	Loaded DLL	Features extracted from information about loaded DLL dependencies
2	Created registry key	Features holding information about created registry keys
3	Modified registry key	Features related to registry modifications
4	Read registry value	Features extracted from information about read registry values
5	Created file	Features corresponding to name of created files
6	Modified file	Features corresponding to modifications done to existing files
7	Deleted file	Features related to deleted files by the executable
8	Read file	Features describing files read by the executable
9	Memory mapped file	Features describing memory mapped files, e.g. DLL usage
10	Driver communication	Features reflecting communication to a system driver
11	Control communication	Features that correspond to file system control operations
12	Thread status	Features related to creation of threads and their status
13	Remote thread created	Features extracted hold information about processes created by threads
14	Process created	Features that reflect process creation and its purpose
15	Socket	Features extracted from information about network socket connections
16	DNS query	Features corresponding to DNS queries to domain names
17	SMTP conversation	Features extracting from e-mail using SMTP
18	HTTP conversation	Features reflecting properties in a HTTP conversation
19	TCP conversation	Features reflecting properties in a TCP transmission
20	UDP conversation	Features reflecting properties in a UDP transmission

Table 5: Dynamic features used in method, where `Entities` consist of a set of features. A complete list is given in Appendix A.

File Activities

File activities are a common operation done by executables. A typical example is with installers[1] that store temporary installation data when the executable is running. Malware utilizes file activities for many reasons, for example primitive actions to stay undetected by an average user. Malware tends to copy itself to different directories, often system directories, where its filename is changed to something that does not seem suspicious. Also, the malware may hide its tracks by deleting itself from the directory it was launched. See entities 5-9 in Table 5.

Process and Thread Activities

A process represents an instance of the executable when it is running, and each process can be made up of several executing threads that perform specific tasks. Retrieving features from these activities are important, because it is from these components all communication to system drivers and libraries are initiated, see entities 10-14 in Table 5. Events typically observed from malware are that they spawn additional processes with similar or identical names to running system processes.

Network Activities

Features describing network activities related to protocols such as HTTP, SMTP, TCP and UDP, need to be retrieved in order to observe what actions the executable is performing to other hosts on the network or Internet, see entities 15-20 Table 5. Malware that is utilized in a botnet, to expand the botnet, communicate with a C&C server, or launch attacks, may be logged by the sandbox and found in the dynamic reports.

As described in Section 2.4, a newly infected bot will announce that it exists by trying to connect to the C&C server. Since embedded IP-addresses tend to be blacklisted, a series of DNS queries will be performed to resolve the correct IP-address. Furthermore, bot malware may be propagated by searching for hosts with known vulnerabilities. Port scanning is part of this activity, which is often done by exploiting weaknesses in TCP and/or UDP protocol at the victim to reveal open ports [111, 45].

Note that whether these activities are retrieved and later used as features really depends on several aspects. The amount of time the sandbox uses to analyze the executable sets a limit on how much network activity is logged. Also, C&C servers might have been taken offline and therefore not available to the specific bot anymore.

4.2.5 Preprocessing

Preprocessing is needed in order to prepare the constructed feature sets before classification algorithms are applied. The first aspect which needs to be addressed is when the feature sets consist of missing feature values. Commonly these are either ignored or replaced, but this is dependent on implementation details and what type of machine learning algorithm we are dealing with [94]. Therefore it is necessary to study the different classifiers employed in the method, to make sure of how they deal with missing values. For example, in *Weka* [99], the implemented machine learning algorithms expect that missing values is stated as ′?′.

Furthermore, limitations regarding supported feature types need to be addressed, because some classifiers support only discrete features, while others support both continuous and discrete features. In this thesis only *Bayes network* (see Section 3.4) requires

[1] Executable used to install software.

discrete features and therefore discretization of the feature set is needed for that particular algorithm. Perhaps the most important aspect in preprocessing is that the feature sets are high-dimensional, which may lead to a poor or deceptive classification result (see Section 3.3.3). Thus, it is needed to study whether the most appropriate features from the feature sets can be selected without excluding important behavior-related features from the feature set.

4.2.6 Classification

In order to check whether the three prepared feature sets contain appropriate behavior characteristics to solve a two-class classification problem, it is necessary to apply classification algorithms (see Section 3.4.1). Five classification algorithms are compared in this thesis:

- *Naive-Bayes*: calculates the conditional probability of a class given the feature, and assumes there are no conditional dependencies between the features in the feature set.

- *K-nearest neighbors*: assigns a class label to an unknown sample based on the majority of the neighbors' class label. Here the nearest neighbors are predefined before running the algorithm.

- *C4.5*: is a decision tree algorithm, where features are assigned to nodes, and the leaves correspond to the class labels.

- *Support vector machines*: assigns a label to an unlabeled sample based on an optimal created hyperplane, where the training samples nearest to the hyperplane is called support vectors.

- *Bayes network*: is a directed acyclic graph, where it calculates the conditional probability of a class given the feature, which is based on conditional probabilities from the connected nodes in the graph.

4.2.7 Evaluation

In order to compare the different classifiers it is necessary to evaluate their performance. In this thesis we deal with limited data for training and testing. Thus, the general approach of splitting the dataset into two parts, two-thirds for training and one-third for testing, will in general be a bad idea. The main reason for this, is that the samples for training (or testing) might not be representative [98]. Simply put, we may not end up with the right proportion of each class in the training and testing set. The worst case scenario would be if one of the classes is missing from the training set.

An alternate method is *cross-validation*, were it is needed to decide on a fixed number of folds (partitions of the data). Here, the training set is randomly divided into m disjoint sets (folds) of equal size n/m, where n is the total classes in the dataset. Moreover, the classifiers are training m times, and each time a different test set is chosen. The estimated performance will be the mean of these m errors [68]. For example, a *threefold cross-validation*, where two-thirds are applied in training and one-third are applied for testing. This procedure is repeated three times, in such a way that every sample has been used exactly once for testing.

Cross-validation will yield variables that need to be applied to calculate the classifiers performance. The following variables will be needed to calculate the *detection rate* and *accuracy*:

- *True positives (TP)* is the number of executables that are correctly classified as malicious software.
- *True negatives (TN)* is the number of executables that are correctly classified as benign software.
- *False positives (FP)* is the number of benign executables that are classified as malicious software.
- *False negatives (FN)* is the number of malicious executables that are classified as benign software.

Using these variables we can now define the equations for the detection rate and the accuracy. These are defined below:

$$DetectionRate = \frac{TP}{TP + FN} \tag{4.1}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

Using both detection rate and accuracy is necessary since detection rate alone will give an imprecise impression of the performance, which does not include true negatives and false positives. This means in practice that you will get a 100 % detection rate if you classify all malicious and benign software as malware. According to the formula this is correct, however, not an ideal condition in regards to false positives.

4.3 System Implementation

In this section the system implementation of the different steps of the theoretical method will be presented. The system has been mainly implemented in *Java* and *Python*. A simplified flow chart, Figure 21, has been made to present the different components with their belonging operations.

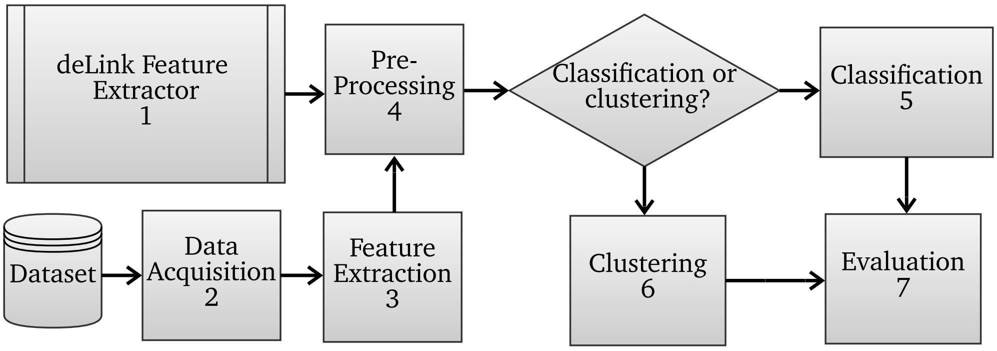


Figure 21: System overview flow chart

From Figure 21, the feature extractor from the *deLink* framework (1) is represented as its own component, mainly because it requires disk images in order to extract file metadata, apply hash filtering, and store an ARFF output (see Section 4.3.1) for further processing. *Data Acquisition* (2) will acquire analysis reports from each of the executables in the dataset. When all analysis reports are retrieved, the features are extracted in the *Feature Extractor* step (3). Next step is *Preprocessing* (4), which is necessary in order to prepare the feature sets for either *Classification* (5) or *Clustering* (6). The classification or clustering step depends on whether the *deLink Feature Extractor* is used or not, since it is designed for clustering file metadata from a set of disk images. Finally, the evaluation of the result is done in the *Evaluation* step (7).

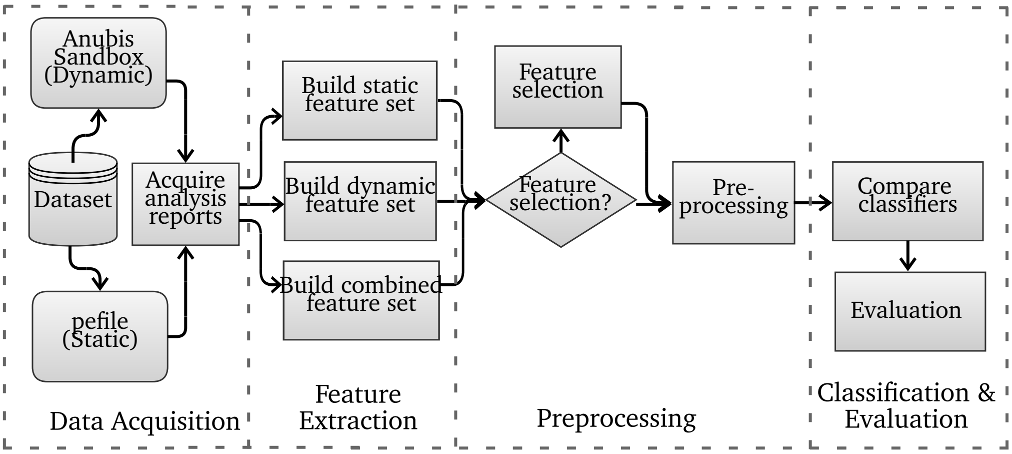


Figure 22: Static and dynamic analysis flow chart

In Figure 22 a more detailed flow chart of the static and dynamic analysis extension is shown. It is divided into four processing steps, which is included in the overall system that was previously presented above. A more detailed view of the *Data Acquisition* step shows that the executables are analyzed by the sandbox *Anubis* and the PE parser *pefile*. The analysis reports are then used to build three different feature sets in the *Feature Extraction* step. The first feature set consists of static features from *pefile*, and the second, dynamic features from *Anubis*. Static and dynamic features are then combined to establish the third feature set. Further, in the *Preprocessing* step, the features can optionally be reduced by applying feature selection. Finally, in the *Classification* and *Evaluation*, five different classifiers are applied and their results are compared and evaluated.

The rest of this section starts with a description of the different tools and libraries necessary to implement the system. Then the main system components are described which include implementation details about analysis report acquisition, feature extraction and building of feature sets, preprocessing, classification and evaluation. Finally, challenges and benefits of the method and implementation are discussed in the last section.

4.3.1 Tools, Libraries and Data Formats

In order to implement a system capable of static and dynamic malware analysis, there are tools and libraries that play a central role for the system. Hence, it is important to present them, with their advantages and limitations. These tools are related to the components shown in Figure 21, especially *Data Acquisition*, *Preprocessing*, *Classification* and *Evaluation*.

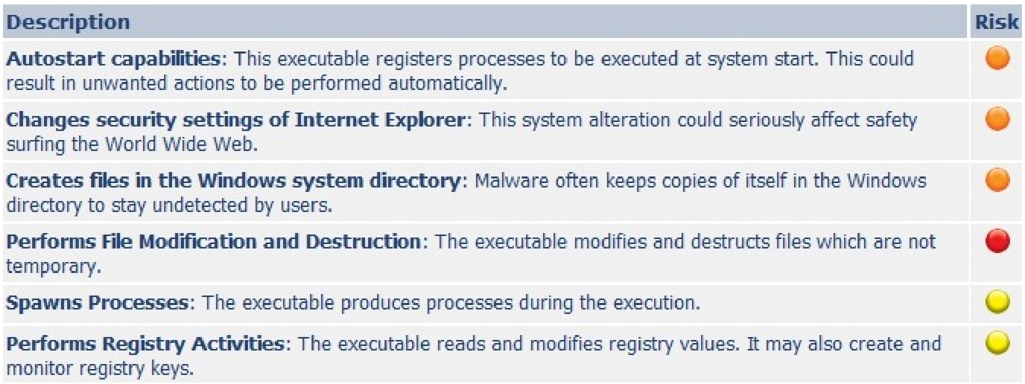
Anubis

Anubis is an online sandbox for analyzing the behavior of PE executables, especially analysis of malware [87, 88]. When an analysis is finished, the sandbox generates a report file that contains the performed actions from the executable. Thus, it is capable of capturing the dynamic behavior of executables. The generated dynamic report is available in different formats, such as HTML, PDF and XML. In this thesis the reports generated in XML have been utilized, since this simplifies the text-parsing during the feature extraction (see Section 4.3.2).

Moreover, the generated reports contain detailed information about:

- Modifications made to the Windows registry.
- Modifications made to the file system.
- Spawned processes by the executable.
- Interactions made to services/processes.
- Logs of all generated network traffic.

Anubis runs the uploaded executables in an emulated environment. This means that it simulates a personal computer in software and will not execute instructions directly on the real processor. Additionally, the sandbox focuses on the security aspects of the executables behavior, which makes it a great tool for malware analysis. An example of an *Anubis* analysis summary is given in Figure 23. Here the different behavior categories are given a risk in order to state whether the executable constitutes a threat.



Description	Risk
Autostart capabilities: This executable registers processes to be executed at system start. This could result in unwanted actions to be performed automatically.	🟠
Changes security settings of Internet Explorer: This system alteration could seriously affect safety surfing the World Wide Web.	🟠
Creates files in the Windows system directory: Malware often keeps copies of itself in the Windows directory to stay undetected by users.	🟠
Performs File Modification and Destruction: The executable modifies and destructs files which are not temporary.	🔴
Spawns Processes: The executable produces processes during the execution.	🟡
Performs Registry Activities: The executable reads and modifies registry values. It may also create and monitor registry keys.	🟡

Figure 23: *Anubis* summery of a SpyBot variant.

Furthermore, this thesis utilizes a python script which is provided on the *Anubis* website[2] in order to upload the malware samples automatically instead of manually uploading one by one via the website. However, this script lacks the capability of automatically retrieving the dynamic analysis reports.

pefile

pefile is a Python module that is designed to work with PE files [59]. The module can access almost all the information in the PE header, including all the sections' information and data (see Section 2.5.2). Some of the modules capabilities are listed below:

[2] `http://anubis.iseclab.org/Resources/submit_to_anubis.py`

- Modifying and writing back to the PE executable.
- PE header inspection.
- Analysis of the different sections.
- Retrieving data.
- Print warnings for suspicious/malformed values, e.g., packed executables.

This module is used in the implementation for static analysis of the executables, where it stores static analysis reports for all the PE executables. An example of some of the information retrieved by *pefile* is shown in Figure 24.

```
[IMAGE_IMPORT_DESCRIPTOR]
0x9414     0x0   OriginalFirstThunk:          0x0
0x9414     0x0   Characteristics:             0x0
0x9418     0x4   TimeDateStamp:               0x0
0x941C     0x8   ForwarderChain:              0x0
0x9420     0xC   Name:                        0x15CA8
0x9424     0x10  FirstThunk:                  0x15428

ADVAPI32.DLL.GetUserNameA Hint[0]
ADVAPI32.DLL.RegDeleteValueA Hint[0]
ADVAPI32.DLL.RegCreateKeyA Hint[0]
ADVAPI32.DLL.RegCreateKeyExA Hint[0]
ADVAPI32.DLL.RegCloseKey Hint[0]
ADVAPI32.DLL.RegOpenKeyA Hint[0]
ADVAPI32.DLL.RegQueryValueExA Hint[0]
ADVAPI32.DLL.RegSetValueExA Hint[0]
```

Figure 24: *pefile* output example

Here, parts of the information from the *data directory* is shown, where imported API calls for the *ADVAPI32* DLL is printed. This DLL provides functionality to the registry, shutdown/restart of the system, spawning processes, and managing the user accounts [112]. *Hint* is a table that suffices for the entire import section [113]. Hence, this information is highly relevant in order to analyze behavior initiated by malware on a system.

ARFF

Before describing the data mining tools and libraries, we need to present the file format which is used to build the different feature sets and later used in the preprocessing and classification steps. Attribute-Relation File Format (ARFF) is a file format, stored in ASCII text, that describes a list of instances that share a set of features [99]. In this context the instances represent each of the executables after extracting the features from the analysis reports generated by *Anubis* and *pefile*. An example is given in Figure 25.

An ARFF-file is divided into two parts, namely the header, which includes the name of the *@relation* a listing of *attribute* statements. The second part is the *@data* section where all of the instances are declared. The different attribute formats supported are:

- *Numeric* - continuous or integer values
- *Nominal* - discrete or a set of predefined values

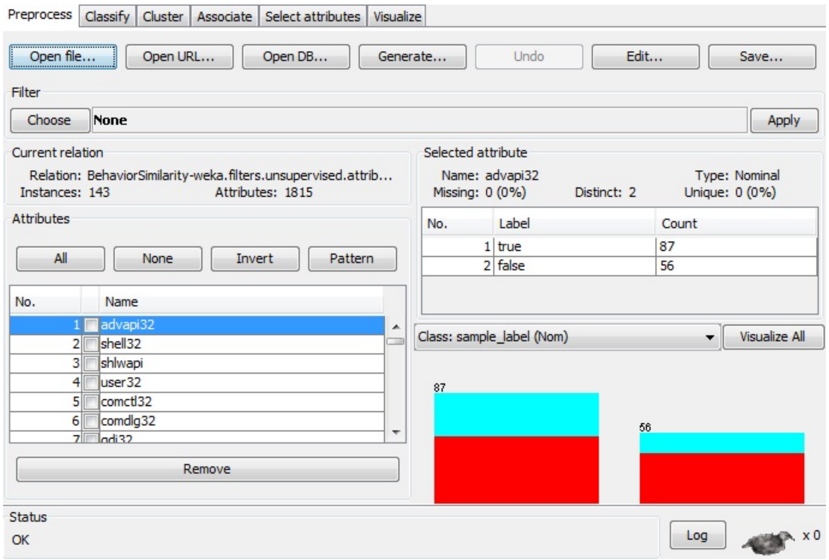
```
%Example of feature set from static analysis
@RELATION static

@ATTRIBUTE ADVAPI32 \{true,false\}
@ATTRIBUTE ADVAPI32_count NUMERIC
@ATTRIBUTE ADVAPI32_functions STRING
@ATTRIBUTE class \{malware,benign\}

@DATA
false,0,?,benign
true,1,‘RegCloseKey’,malware
true,2,‘ ‘‘RegQueryValueExA’’ ‘‘RegSetValueExA’’ ’,benign
```

Figure 25: ARFF example

- *String* - text representation
- *Date* - combined date and time “yyyy-MM-dd’T’HH:mm:ss”

Weka

The Waikato Environment for Knowledge Analysis (*Weka*) is an open source machine learning toolkit, that includes both algorithms for classification, clustering and various preprocessing techniques [99, 98]. Furthermore, *Weka* enables us to quickly try out different machine learning algorithms on the feature sets that were generated from the feature extraction step. *Weka* consists of four user interfaces, however, we will only utilize the *Explorer* for initial prototype testing, see Figure 26.



Filter: None

Current relation
Relation: BehaviorSimilarity-weka.filters.unsupervised.attrib...
Instances: 143 Attributes: 1815

Selected attribute
Name: advapi32 Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count
1	true	87
2	false	56

Attributes

No.		Name
1	□	advapi32
2	□	shell32
3	□	shlwapi
4	□	user32
5	□	comctl32
6	□	comdlg32
7	□	gdi32

Class: sample_label (Nom)

Status
OK

Figure 26: Weka Explorer example

Otherwise, we import the necessary libraries from *weka.jar*[3] for preprocessing, classification and evaluation. This approach avoids manually using the *Explorer*. Furthermore,

[3] Available in the install folder of *Weka*.

the different preprocessing, classification and evaluation methods and their parameters are described in Section 4.3.2.

4.3.2 Implementation Details

The main aim of this section is to present further implementation details in order to describe how each step is performed. It will be presented as shown in Figure 22, describing each step with a simple pseudocode. When dealing with machine learning tools it is often necessary to choose certain parameters before running the algorithms. These parameters will also be described in this section. Furthermore, the complete source code of the method can be found in Appendix E.

Acquire Analysis Reports

In order to acquire analysis reports existing tools will be employed for both *Anubis* and *pefile*. For *Anubis* an uploading tool in python exists. This enables us to upload samples to the sandbox without using the interface given at the website. However, these tools are not fully automated. In order to automate the acquisition of analysis reports, several operations are necessary, as shown in Code 1.

Code 1 Acquire Analysis Reports

```
for all samples in dataset do
  upload samples to Anubis
  store sample identifiers
end for
for all sample identifiers do
  map sample identifier to dynamic analysis report
  while dynamic analysis report is not ready do
    sleep for 5 seconds
  end while
  store dynamic analysis report
end for
for all samples in dataset do
  analyze samples with pefile
  store static analysis reports
end for
```

This was implemented in python, where all samples are uploaded first to *Anubis* and for each uploaded sample an identifier is stored. This identifier is later used to retrieve the dynamic analysis report. Otherwise, you must do this manually from the website. The last operation done in this step is done by *pefile*, which is utilized to retrieve static analysis reports.

Feature Extraction

The feature extraction step required a great deal of development from scratch. This step is shown in Code 2 and is implemented in Java. The first operation starts with loading the static and dynamic analysis reports into two arrays. Then each static analysis report is parsed with a text-parser, that store all encountered DLL names and their function calls. These are found in the static report describing the *image import descriptor*, which lies in the *image data directory* of a PE executable, see Figure 27. Note that an executable will have several *image import descriptors* for each DLL that is imported by the executable. These features are stored in a static feature set, which use an identifier based on

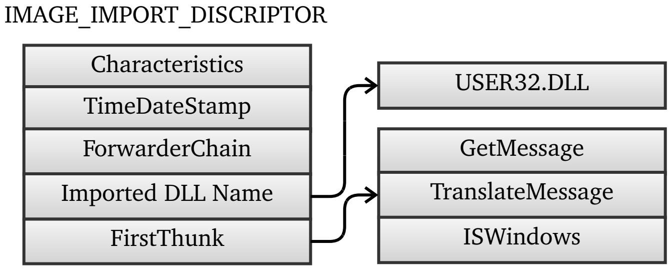


Figure 27: Image data directory of a PE executable

the report name. In situations where multiple function calls are present for a retrieved DLL, they are concatenated and stored in a single string for each DLL, where they are separated by a quotation mark (“).

Code 2 Feature Extraction

```
load static analysis reports
load dynamic analysis reports
for all static analysis reports do
  parse reports with a text-parser
  while parsing do
    store DLL names and their function calls
  end while
  add to static feature set with report name as identifier
end for
for all dynamic analysis reports do
  parse reports with a XML-parser
  for all encountered entities do
    store all variable names and content
  end for
  add to dynamic feature set with report name as identifier
end for
build combined feature set by matching identifiers
store feature sets as ARFF-files
```

A similar operation is done with the dynamic analysis reports. However, here we need to employ a XML-parser in order to retrieve the features. This is solved with the *DOM* (Document Object Model) library that represents the XML in a hierarchy structure. All encountered entities are analyzed to store the variable names and its contents. This information is further used as features and stored in a dynamic feature set. As with the static feature set, we also use an identifier based on the report name for each dynamic report.

Building the dynamic feature set also exploits string concatenation. This is necessary since each sample may be represented by multiple entities of same type (e.g., loaded DLL) that will have multiple feature values representing a specific feature (e.g., DLL name). Therefore, these values are concatenated into a single string where the values are separated by quotation marks (“).

In order to build a combined feature set of both static and dynamic features, we match the identifiers from the static and dynamic feature set. Obviously, this operation cannot be initiated before the static and dynamic feature sets have been created. The last operation done in this step is storing these feature sets as three ARFF-files. This is done by employing *ArffSaver* which is a package in the *Weka* library.

Feature Selection

Feature selection will be an optional step in this computational method. The applied feature selection algorithms chosen are correlation-based feature selection and the improved version by Nguyen *et al.* [104, 105]. They will be applied in order to remove features with no relationship to the class label. Before applying this step it is necessary to utilize the *StringToWordVector* package as described below.

Preprocessing

In the next step, preprocessing techniques are applied to prepare the data to improve the result in the later classification step. This is also implemented in Java, where we use two packages found in the *Weka* library, namely *StringToWordVector* to separate multiple features stored in a single string, and *Discretization* to convert the features into discrete values if the *Bayes network* is utilized for classification. These operations are shown in Code 3.

Code 3 Preprocessing

```
for all feature sets do
    apply StringToWordVector
    if classifier is Bayes network then
        apply Discretization
    end if
end for
```

Parameter	**Description**
-R first-last	Specifies to act on the whole range of features
-W 10000	Keeps up to 10000 features
-tokenizer “WordTokenizer -delimiters ””	Specifies to split a string based on words that are separated by ””

Table 6: Parameters for *StringToWordVector*

Moreover, when using these packages, different parameters can be set. These may influence the classification result if they are not set carefully. In Table 6, the adjusted parameters for *StringToWordVector* is given, otherwise default parameters were chosen for both of the packages. It is worth noticing that *StringToWordVector* separates the concatenated strings and converts all the retrieved values to nominal features. This is done by counting the presence of the value in each entity.

Classification and Evaluation

In the final step we applied five different classification methods in order to find out which type of algorithm is most suitable for the different feature sets. To evaluate the different algorithms 10-fold *cross-validation* (see Section 4.2.7) is being computed. This step is implemented and automated in Java by using these packages for classification:

- *NaiveBayes* - Naive-Bayes classifier
- *IBk* - K-NN classifier
- *J48* - C4.5 classifier
- *LibSVM* [114] - SVM classifier
- *BayesNet* - Bayes network classifier

Code 4 Classification and Evaluation

```
for all classifiers do
    prepare feature sets with preprocessing
    train classifier
    cross-validation of classifier
    calculate detection rate and accuracy
end for
```

Here, Code 4 shows the pseudocode for this approach. Each classifier is trained, with default parameters, after the feature set is prepared by the preprocessing step. Then they are evaluated by 10-fold cross-validation. The *confusion matrix (CM)* is computed during the evaluation and is used to retrieve *true positives (TP)*, *false negatives (FN)*, *false positives (FP)*, and *true negatives (TN)*.

$$CM = \begin{pmatrix} TP & FN \\ FP & TN \end{pmatrix}$$

Using these values we now can calculate the *detection rate* and *accuracy* for each classifier.

4.3.3 Summary

The system implementation of the computational method includes several system components, libraries and tools. The following items have its base from the main components shown in Figure 19 and the enumerated operations stem from the system implementation in Figure 22.

- Data Acquisition
  1. Acquire analysis reports
- Feature Extraction
  2. Build static feature set
  3. Build dynamic feature set
  4. Build combined feature set
- Preprocessing
  5. Feature selection
  6. Preprocessing
- Classification
  7. Run classification algorithms
- Evaluation
  8. Compare classifiers
  9. Evaluation

4.4 Method Discussions

The proposed computational method and the system implementation, presented in this chapter, have its strengths and limitations. It is therefore important to discuss the different aspects of the method and the current implementation.

4.4.1 Acquiring Analysis Reports

The method employs *pefile* which is used to dump information from the different sections of the PE header and store this in a static report. Also, *Anubis* sandbox is applied, which is an online sandbox that analyzes the behavior of the uploaded executables. When the sandbox has completed the analysis, due to terminated processes or a time limit, a dynamic report is generated and made available for download. *pefile* will not execute the files during analysis, and therefore, if acting carefully, a malware infection is avoided. Similarly, a malware infection is avoided when employing *Anubis*, since you hand over the analysis problem to a different system, and can therefore avoid sacrificing your own host, or spending time configuring a virtual machine that needs to be re-initialized for each analyzed sample.

The greatest limitation with *pefile*, and PE parsers in general, is that they do not provide as advanced functionality as disassemblers and debuggers (e.g., *IDA Pro* [92]). Thus, analysis of control-flow will not be possible. However, disassemblers and debuggers are vulnerable to obfuscation techniques, such as polymorphism and metamorphism. This issue is avoided by *pefile*, since it does not analyze the assembly code. However, it is vulnerable to packers and cryptors that compresses and/or encrypts parts of the PE header. Another important aspect is that *pefile* will not detect loaded DLL's in code. In the example in Appendix C.4, the DLL is loaded with *LoadLibrary* function. However, *Kernel32.dll* which has implemented the *LoadLibrary* function is detected by *pefile*.

One of the limitations with *Anubis*, and other sandboxes, is that they only analyze one state of the executable. This implies that only certain behavior will be logged in the dynamic report. Another factor that limits the logged behavior is time constraints on *Anubis*. By default, *Anubis* only executes an unknown executable in approximately four minutes. Furthermore, privacy is an important issue for companies that deals with analysis of suspicious files and zero-day malware[4], since they are often bound by non-disclosure agreements to other parties. Although *Anubis* supports secure uploading through SSL[5] and do not reveal any sensitive information in the published analysis reports, we cannot be sure what type of information is stored for *Anubis*' operators. Thus, if privacy is an issue, a local sandbox should be preferable compared to the online sandbox utilized in the proposed proof-of-concept.

4.4.2 Implementation Challenges

When *Anubis* is utilized for dynamic analysis, the system will place uploaded samples in a queuing system where the samples uploaded through the web-interface has higher priority. Also, there are other and more prioritized users that are able to upload large datasets using the upload code from the website. Since the regular user is passed in the queue, the analysis can take several days if you are unlucky. Thus, testing the report retrieval script was a time-consuming process. However, as a backup solution it is possible to retrieve the results at a later stage by registering on the website and check if all the

[4] No signature currently exists.

[5] Secure Sockets Layer

samples are analyzed. Then it is possible to download the reports manually or re-run the upload script, since no analysis is required when identical samples have been analyzed before[6].

By employing preprocessing techniques from the *Weka* libraries we encountered several issues. The first issue was related to the separation of the concatenated strings that contained multiple features. Here, the default setting included too many splitting criteria and was limited to 1000 features. This setting was modified to split on quotation marks and changing the limit to 10000 features. Moreover, an issue was detected when adding the class label as the first attribute in *Weka*, because there is a known bug[7] when dealing with string attributes. String and nominal values are stored as numbers, which are employed as indexes to an array of possible attribute values. However, the first string value is assigned to index 0 and internally stored as 0. The problem is that the ARFF reader of *Weka* interprets internal value 0 as no output. This implies, in practice, that the first class label (e.g., malicious) is not visible in the generated ARFF file. In our case this was solved by putting the class labels as the last attribute in the ARFF file.

One important aspect when either using the *Weka's* graphical user interface or using its libraries in your own implementation, is that the program tends to crash due to lack of memory heap space. Especially when utilizing large feature sets is this relevant. It is worth noticing that less memory heap space was required when importing the classification algorithms in the Java implementation. This is the case because the feature sets are not loaded entirely into memory compared to when you are using *Weka's* graphical user interface.

[6] Comparing MD5 hashes.

[7] `http://old.nabble.com/Sparse-Instances---String-attributes-td26310509.html`

5 Experiments

The previous chapter described the design and implementation of the computational method, whereas this chapter is devoted to the experiments that were conducted using the presented method. First the experimental environment and the applied dataset are given, followed by a description of how the different scenarios were executed to test the two-class classification scheme. Finally, the main experiment results are provided and discussed in order to study what type of analysis approach is best suited for a two-class problem.

5.1 Experimental Environment

For the experiments, we must use methods to ensure an isolated environment. This is necessary since we are dealing with malware and do not want the malware to propagate to other hosts. To deal with this issue we utilized a Linux workstation and analyzed malware meant for Windows only. Additionally, a virtualized environment was installed in case for a further analysis in a Windows environment was needed.

5.1.1 System Setup

The system utilities were installed and configured in the Linux environment where the computational method was implemented. As for the Linux distribution, we installed *Ubuntu* [115] which is based on the stable and popular *Debian* Linux distribution. Also, for running virtual machines (VM) we utilized *VMware* [14]. The complete system setup is given below:

- Processor: 2 cores @ 2,5 GHz
- Memory: 2048 MB
- Hard disk: 160 GB
- Operating system: Ubuntu 10.10 Desktop
- VM environment: VMware Workstation 7.1.0 build-261024
- VM host: Windows XP SP2

Furthermore, the following tools, libraries and integrated development environment were utilized in order to develop the computational method complying with Chapter 4:

1. pefile (1.2.10) [59]
2. Anubis sandbox [87]
3. Weka and Weka API (3.6.4 Stable) [99]
4. Python (2.6.6) [116]
5. Eclipse IDE (Helios build-20100617-1415) [117]

Using these tools are straightforward, since *pefile* and *Anubis* sandbox have scripts that can be imported in order to make the static and dynamic analysis of the whole dataset an automated procedure. Furthermore, the *Weka API* can be imported as a jar-file in the development environment *Eclipse*. By downloading a python extension for *Eclipse* it is possible to do all the implementation using a single development environment.

5.1.2 Dataset

For the experiment an appropriate dataset is required. The botnet malware was manually downloaded from websites such as vxheavens [10], packetstorm [11] and offensivecomputing [12]. Furthermore, benign software was downloaded from websites such as pendriveapps [118]. A total of 143 samples were used in the experiments, whereas 93 were labeled as malicious and 50 were labeled benign.

Malware

The malware samples used in the experiments are various versions from three large malware families. These are found in botnets where one of their tasks is to establish and maintain a connection with the botmaster's C&C server and await further commands. The reason for choosing samples from three malware families is that we wish to study whether we can implement a two-class classification scheme based on the datasets' behavior. Also, malware versions within a family will utilize different behavior on the host and the applied obfuscation techniques may differ.

- *SpyBot* [119] is a worm that propagates through P2P-sharing and IRC. This worm can also infect hosts that are already compromised by common backdoors. In botnets, various versions of SpyBot have been used for C&C related activities and launching DDoS attacks.

- *Torpig* [120] also known as *Sinowal* or *Anserin*, is a Trojan that logs keystrokes and activity to certain bank websites. The Trojan employs domain flux in order to communicate with its main C&C servers [121].

- *SdBot* [122] also known as *Randex* or *Agent*, is a backdoor that connects to an IRC channel using its own IRC client. From here an adversary can remotely control the infected host to for example perform DDoS attacks against a third party and try to infect other users connected to other IRC channels.

Benign Software

A set of benign samples with "similar" behavior-characteristics as botnet malware have been used in the experiments. Here, different portable software such as mail clients, browsers, network tools, instant message clients and BitTorrent clients [118] have been used.

5.2 Experiment Scenarios

In this section the experiment scenarios will be presented. There will be two main classification scenarios; where the first scenario (1) applies the complete feature sets for classification, and the second scenario (2) applies feature selection methods before classification to select the most appropriate features based on some quality measure.

Furthermore, during the experiments we assume that (i) the dataset only contains portable executables, (ii) the dataset can be analyzed with the chosen static and dynamic analysis tools, and (iii) analysis of the dataset must yield parts of the executables behavior-characteristics if obfuscation methods are utilized. The next sections describe the different scenarios in more detail, including a proof-of-concept scenario.

5.2.1 Proof-of-Concept

The proof-of-concept experiment was executed to verify whether the implementation of the computational method worked properly and to check if the two-class classification scheme was able to distinguish between malicious and benign software. The scenario was executed in the following manner:

- Before the static and dynamic analysis were initiated, we first manually downloaded the botnet malware and benign software samples used in our experimental dataset. The dataset, consisting of 143 samples, was analyzed locally by *pefile* and online by *Anubis*. The static reports and dynamic reports were retrieved, and stored in separate folders to make the feature extraction component more efficient.

- The features were retrieved by utilizing the implemented text and XML-parser, and structured into two feature sets, a static and dynamic feature set (see Section 4.2). A combined feature set was built by matching the report identifiers. These feature sets were stored as ARFF-files.

- Each ARFF-file was opened in the *Weka Explorer* to further prepare the features for the classification task by applying preprocessing techniques. First, the concatenated strings that represent DLL function names and entities were separated by utilizing *StringToWordVector*. Then, unsupervised discretization was applied.

- After preprocessing, Bayes network was executed as the classification algorithm in *Weka Explorer*, and this was utilized with default parameters. The reason for choosing this classifier was that it takes into account all available features and their dependencies. For evaluation, a 10 fold cross-validation was utilized to ensure a reliable result.

5.2.2 Complete Feature Sets

This experiment was based on a further extension of the computational method, shown in Appendix B. We have avoided using *Weka Explorer* for preprocessing and classification. Instead, *Weka API* was imported in the Java implementation. This automates the process of executing several classifiers and requires less memory heap space. This experiment was performed in the following steps:

- We employed the already retrieved static and dynamic reports from the proof-of-concept experiment (see Section 5.2.1).

- Static, dynamic and a combined feature set were built of the extracted features from the static and dynamic reports. *StringToWordVector* was applied before the feature sets were stored as ARFF-files.

- Five different classification algorithms were executed in order to study the type of classifier appropriate to handle the large amount of features extracted from the static and dynamic reports. The different classifiers that were executed are:

- Naive-Bayes
- K-nearest neighbors (K-NN)
- C4.5
- Support vector machines (SVM)
- Bayes network

Note that Bayes network requires discretized features. Therefore, we applied unsupervised discretization (as recommended by the *Weka* manual [123]) before the classifier was executed. All classifiers were evaluated by utilizing 10 fold cross-validation.

5.2.3 Reduced Feature Sets

The final experiment scenario was based on applying feature selection techniques in order to choose the most appropriate features to improve the classification results. This also reduced the computational resources required for classification. The scenario was executed in the following manner:

- We employed the already retrieved static and dynamic reports from the proof-of-concept experiment (see Section 5.2.1).
- The already created ARFF-files, representing the static, dynamic and combined feature set, were reused from the complete feature sets experiment (see Section 5.2.2).
- Two feature selection algorithms were applied, where their procedure included different steps.
    - Corrolation-based feature selection (CFS) was executed via *Weka Explorer* using default parameters and the result stored as new ARFF-files.
    - Generic feature selection ($\mathsf{GeFS}_{\mathsf{CFS}}$) was executed by including these steps:
        - Computed the correlation coefficient (merit, see Section 3.3.3) for all the features.
        - Features with no or minimal relationship to the class label were removed from the ARFF-files.
        - The $\mathsf{GeFS}_{\mathsf{CFS}}$ algorithm was executed and the algorithm's output yielded the selected features.
        - ARFF-files were manually modified and stored.
- Five different classification algorithms were executed in order to study whether the feature selection improved the classification results. The different classifiers that were executed are:
    - Naive-Bayes
    - K-nearest neighbors (K-NN)
    - C4.5
    - Support vector machines (SVM)
    - Bayes network

Similar to the previous experiment, we applied unsupervised discretization to Bayes network only. All classifiers were evaluated by utilizing 10 fold cross-validation.

5.3 Experiment Results

Next, the experiment results will be presented. It will look at the results given by the proof-of-concept, complete feature sets and reduced feature sets scenarios to evaluate what type of feature set is suitable for analysis of malware behavior. The feature sets are studied by comparing the results from a set of classifiers.

5.3.1 Proof-of-Concept

An initial observation was done on the retrieved analysis reports during the data acquisition steps. In the static reports, from *pefile*, we observed that a few variants within one malware family did not import as many DLLs compared to the others. By using *PEiD*[1] we found out that packers were employed on these executables, see example of the *SpyBot* variants in Appendix C, where variants have been packed with *Themida*[2].

Similarly, by studying the dynamic reports, from *Anubis* sandbox, we discovered that variants within same malware family yielded different behavior. For example, not all malware samples produced any network-related behavior, even though this is a common type of activity when botnet malware is executing. The causes of this result may be many. Different preconditions may not have been met when executing the malware, for example that it tries to connect to a C&C server at a specific time, either time of day or spent executing time (e.g., one hour). This is a limitation with *Anubis* since it only executes the malware in a limited time for behavioral analysis.

After the feature extraction step we acquired three ARFF-files, and by opening them in *Weka Explorer* we could observe, before applying any preprocessing methods, that the static feature set had 86 features, the dynamic feature set had 75 features, and the combined feature set had 118. Furthermore, the concatenated strings were separated by *StringToWordVector*, which resulted with:

- Static feature set of 1814 features
- Dynamic feature set of 5494 features
- Combined feature set of 7347 features

The reason for the combined features set is smaller than the sum of static and dynamic features is because of DLL name conflicts, meaning that features with identical names were created from the dynamic and static reports[3]. This was solved by integrating the DLL name into the function name in the static feature set (e.g., Kernel32.CreateFileA).

Feature Set	**Detection Rate**	**Accuracy**
Static	98.92 %	87.41 %
Dynamic	94.62 %	88.11 %
Combined	97.85 %	86.71 %

Table 7: Proof-of-Concept with Bayes network

Discretization (unsupervised) was applied through *Weka Explorer*, before the Bayes network classifier was trained and executed. The results are given in Table 7, and here

[1] Detects common packers, cryptors and compilers in PE files `http://www.peid.info/`.

[2] `http://www.oreans.com/themida.php`

[3] *Weka* do not support features with identical names.

we can see that the best detection rate was given by the static feature set, followed by the combined and dynamic feature set.

The static feature set is the least complex of the two since it does not have as strong feature dependencies compared to the other two. The dynamic feature set is built up with entities, where each entity contains from one to multiple features defining a specific behavior. On the other hand, the dynamic feature set gave the best accuracy score, mostly because it did not yield as many false positives as the others.

Furthermore, the combined feature set gained higher detection rate by including both static and dynamic features compared to the dynamic feature set. However, this feature set gave the highest false positive rate and false negative rate compared to the others, which reflect its accuracy score.

5.3.2 Complete Feature Sets

Here we avoided utilizing *Weka Explorer* for preprocessing by importing the necessary libraries in the Java code. We verified that we came up with the same number of features by comparing with the feature sets in the proof-of-concept scenario. Additionally, we imported the necessary libraries for the five classifiers in the Java code. To ensure that the same parameters were used we verified the results by running the same classifiers with the same feature sets in *Weka Explorer*. The rest of this section discusses the results achieved from the different feature sets. See Appendix D for raw output from the implementation, that contain additional information such as the amount of true positives, true negatives, false positives and false negatives.

Static Feature Set

The results from the static feature set are given in Table 8, where C4.5, SVM and Bayes network achieved the best detection rate of 98.92 %. The C4.5 got the highest accuracy score compared to the others, due to the lowest false postive rate. Since the static feature set do not contain many dependent features, only a dependency between DLL name and its function calls, it is suitable for the C4.5 algorithm's feature-quality measure (gain ratio) that operates independently of the features (see Section 3.4.1).

Classifier	Detection Rate	Accuracy
Naive-Bayes	95.70 %	87.13 %
K-NN	96.77 %	90.21 %
C4.5	98.92 %	90.91 %
SVM	98.92 %	84.63 %
Bayes Network	98.92 %	87.41 %

Table 8: Classification results from complete feature set using static features

Dynamic Feature Set

The dynamic feature set is, as mentioned previously, complex with respect to the amount of features and their dependencies. This is reflected in the Table 9, where Bayes network classifier got the best detection rate of 94.62 %, followed by the SVM classifier of 93.55 %. Both classifiers are able to handle feature sets with many dependencies. This is the case, since Bayes network calculates the conditional propabilities of the dependent features to decide the sample class. SVM, on the other hand, transforms the feature set in such a way that the samples can be separated by a hyperplane. However, the SVM classifier got the weakest accuracy score of all classifiers, because of high false negative and false positive rate (see Appendix D).

Classifier	Detection Rate	Accuracy
Naive-Bayes	83.87 %	86.71 %
K-NN	86.02 %	87.41 %
C4.5	84.94 %	84.62 %
SVM	93.55 %	83.21 %
Bayes Network	94.62 %	88.11 %

Table 9: Classification results from complete feature set using dynamic features

Combined Feature Set

The results from the combined feature set is shown in Table 10, and as with the dynamic feature set, the Bayes network and SVM classifier got the highest detection rate of 97.85 %. By combining static and dynamic features the detection rate was improved from only using dynamic features, however, the accuracy score dropped for the Bayes network and remained the same for the SVM classifier. Moreover, the best accuracy score was achieved by the naive-Bayes of 91.61 %, followed by the K-NN classifier of 89.51 %.

Classifier	Detection Rate	Accuracy
Naive-Bayes	89.24 %	91.61 %
K-NN	95.70 %	89.51 %
C4.5	88.17 %	86.01 %
SVM	97.85 %	83.21 %
Bayes Network	97.85 %	86.71 %

Table 10: Classification results from complete feature set using combined features

Summary

When averaging the results using the complete feature sets we can see that the Bayes network and SVM classifier yielded the overall best detection rate, see Figure 28. On the other hand, these also suffered from the poorest accuracy score. The last three classifiers also performed well, where the K-NN achieved the highest accuracy score, followed by the naive-Bayes.

5.3.3 Reduced Feature Sets

Here, two feature selection algorithms were applied, namely, CFS and $\mathsf{GeFS}_{\mathsf{CFS}}$. *Weka Explorer* was utilized in order to retrieve the reduced feature set using CFS, while the $\mathsf{GeFS}_{\mathsf{CFS}}$ needed additional implementation steps. The features' merit (correlation coefficient) were computed by using the merit calculation available in the *Weka* source code[4].

[4] weka-src.jar

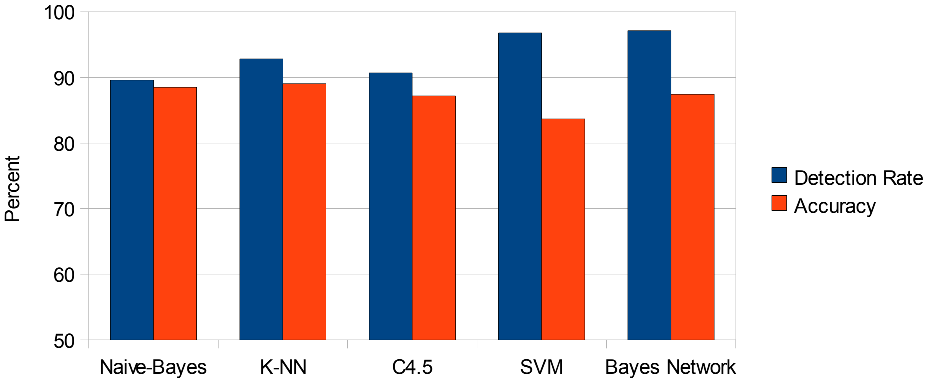


Figure 28: Averaged result from complete feature sets

Before running the GeFS_{CFS}, we removed the features with no relationship[5] to the class label in the static feature set, and we removed features with minimal relationship[6] to the class label for the dynamic and combined feature set. This was done because of the vast amount of features and to reduce the computational complexity when running GeFS_{CFS}.

Static Feature Set

When employing the CFS algorithm the static feature set was reduced from 1814 to 11 features, while the GeFS_{CFS} reduced the feature set to 7 features. By studying the selected features we noticed that the feature selection algorithms chose three shared features, while the rest were different. The classification results using the reduced static feature set, from both feature selection algorithms, is shown in Table 11. All classifiers using the reduced feature set retrieved from the CFS got a detection rate of 97.85 %, where the naive-Bayes classifier got the best accuracy score of 94.4 %. The classification results from the GeFS_{CFS} algorithm also gave good results. Here, the C4.5 got the best detection rate of 98.98 %.

Classifier	CFS		GeFS_{CFS}	
	Detection Rate	**Accuracy**	**Detection Rate**	**Accuracy**
Naive-Bayes	97.85 %	94.40 %	96.77 %	91.61 %
K-NN	97.85 %	93.71 %	96.77 %	92.31 %
C4.5	97.85 %	93.31 %	98.92 %	90.91 %
SVM	97.85 %	91.61 %	96.77 %	90.91 %
Bayes Network	97.85 %	91.61 %	96.77 %	90.91 %

Table 11: Classification results from reduced feature set using static features

Dynamic Feature Set

In this scenario, the CFS reduced the dynamic feature set from 5494 to 19 features, and the GeFS_{CFS} reduced the feature set to 5 features. As with the static feature set, we found

[5] Correlation coefficient higher than 0.
[6] Correlation coefficient higher than 0.1.

that all the five features from GeFS_{CFS} were shared in the feature set retrieved from CFS. The best detection rate was given by the K-NN classifier of 91.4 % by using the reduced feature set from the CFS (see Table 12). However, we achieved the highest accuracy score using Bayes network and naive-Bayes of 91.61 %, with the reduced feature set from GeFS_{CFS}. Compared to the complete feature set, we achieved a more even detection rate and accuracy since we reduced the feature set into a less complex one with absent/weak feature dependencies.

Classifier	**CFS**		**GeFS$_{\text{CFS}}$**	
	Detection Rate	**Accuracy**	**Detection Rate**	**Accuracy**
Naive-Bayes	88.17 %	91.61 %	88.17 %	91.61 %
K-NN	91.40 %	90.91 %	88.17 %	87.41 %
C4.5	86.02 %	90.21 %	87.10 %	87.41 %
SVM	86.02 %	90.21 %	87.10 %	88.81 %
Bayes Network	87.10 %	90.91 %	88.17 %	91.61 %

Table 12: Classification results from reduced feature set using dynamic features

Combined Feature Set

Here, the CFS algorithm reduced the combined feature set from 7347 features to 29 features, while the GeFS_{CFS} algorithm reduced the feature set to 48 features, and these reduced feature sets shared 17 features. In Table 13 the classification results are given, and similar to the results from the reduced feature set of the static and dynamic feature set, we obtained a more even detection rate and accuracy score from the tested classifiers. The highest detection rate of 97.85 % was achieved from the reduced feature set from GeFS_{CFS} using the K-NN classifier. Moreover, the highest accuracy score of 95.10 % was achieved from both reduced feature sets. This was the naive-Bayes classifier that used the reduced feature set from the CFS, and Bayes network classifier from the GeFS_{CFS}.

Classifier	**CFS**		**GeFS$_{\text{CFS}}$**	
	Detection Rate	**Accuracy**	**Detection Rate**	**Accuracy**
Naive-Bayes	92.47 %	95.10 %	91.40 %	94.41 %
K-NN	97.85 %	94.51 %	97.85 %	95.10 %
C4.5	92.47 %	92.01 %	91.40 %	90.91 %
SVM	89.24 %	92.31 %	91.40 %	93.71 %
Bayes Network	91.40 %	94.41 %	92.47 %	95.10 %

Table 13: Classification results from reduced feature set using combined features

Another experiment was conducted using combined feature sets; instead of applying the feature selection algorithms on the complete feature set, we merged the reduced feature sets using static and dynamic features. This implies that the combined feature set from CFS consists of 30 features, and the combined feature set from GeFS_{CFS} consists of 12 features. Furthermore, the results (see Table 14) shows that the overall results from the GeFS$_{CFS}$ are better compared to CFS. The highest detection rate (96.77 %) and accuracy (94.41 %) is achieved by the GeFS$_{CFS}$ utilizing Bayes network classifier.

Classifier	**CFS**		**GeFS$_{CFS}$**	
	Detection Rate	**Accuracy**	**Detection Rate**	**Accuracy**
Naive-Bayes	91.40 %	94.41 %	93.55 %	92.31 %
K-NN	96.77 %	92.31 %	96.77 %	92.31 %
C4.5	94.62 %	90.21 %	95.70 %	93.71 %
SVM	93.55 %	90.21 %	96.77 %	91.71 %
Bayes Network	92.47 %	95.10 %	96.77 %	94.41 %

Table 14: Classification results from reduced feature set by merging the reduced feature set from static and dynamic features

Summary

Figure 29 shows the averaged results from the reduced feature sets. Compared to the averaged results from the complete feature sets, we achieved a more even detection rate and accuracy score between the tested classifiers. The main reason for this is due to the greatly reduced feature sets that made the feature dependencies less complex. Overall, the feature sets retrieved from CFS gave best results using the K-NN classifiers and C4.5. The retrieved feature sets from GeFS$_{CFS}$ gave best results using the C4.5, SVM and Bayes network classifiers.

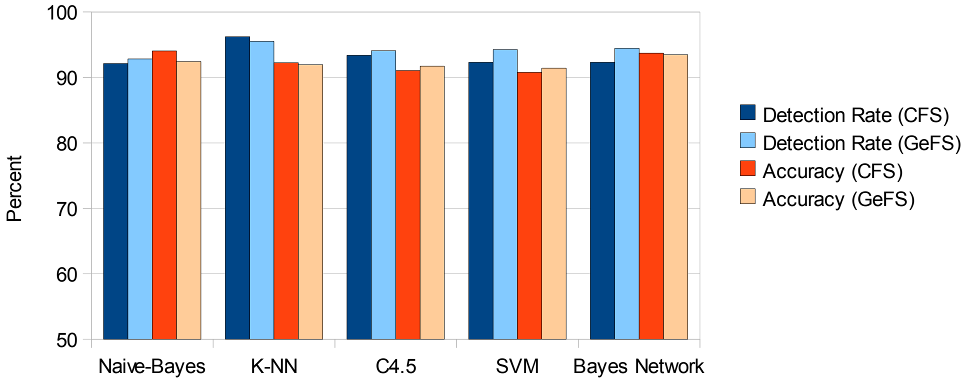


Figure 29: Averaged result from reduced feature sets

Furthermore, the feature selection algorithms yielded different result when choosing the most appropriate features. Table 15 shows the number of features selected by the two feature selection algorithms, where GeFS$_{CFS}$ chose the smallest number of features for the static and dynamic feature set, except for the first combined feature set.

5.4 Experiment Discussions

The experiments conducted in this chapter tested whether an executable's behavior can be used in a two-class classification scheme in order to distinguish between malicious and benign executables. The results clearly shows that the constructed feature sets based

Feature Set	Full-set	CFS	GeFS$_{\text{CFS}}$
Static	1814	11	7
Dynamic	5494	19	5
Combined I	7347	29	48
Combined II	7347	30	12

Table 15: Selected features with CFS and GeFS$_{\text{CFS}}$

on behavior characteristics obtained from static and dynamic analysis methods can be utilized to distinguish malicious software from benign software, even when using a dataset of only 143 samples. From the results where we utilized the complete feature sets, we got the best results using the static feature set, see Table 16. However, it is worth noticing that the result differences between the different feature sets were small, and static analysis only gives a limited view of the expected behavior compared to dynamic analysis. Thus, the combined approach should be best suited for detecting botnet malware.

Feature Set	Average Detection Rate			Average Accuracy		
	Full-set	CFS	GeFS$_{\text{CFS}}$	Full-set	CFS	GeFS$_{\text{CFS}}$
Static	97.84 %	97.85 %	97.20 %	88.05 %	92.92 %	91.33 %
Dynamic	88.60 %	87.74 %	87.74 %	86.01 %	90,77 %	89,37 %
Combined I	93.76 %	92.69 %	92.90 %	87.41 %	93.67 %	93.85 %
Combined II	93.76 %	93.73 %	95.91 %	87.41 %	92.45 %	92.83 %

Table 16: Averaged result using the different feature sets

All malware used in the experiments originate from three malware families, where different variants were collected. Since these are commonly used for botnet related activities, they will optimize the tested classifiers to be able to solve a two-class problem for this particular behavior. This implies that the performance of the trained classifiers will decrease when they are presented with unknown malware that does not yield any botnet type of behavior. Additionally, the benign samples used in the experiments were portable executables that yielded behavior related to network activity, and therefore the classifier will be optimized for this type of benign behavior. However, the same situation as with the collected malware samples will arise, namely that the performance will decrease (more false positives) if benign samples with unknown behavior are presented to the classifier. Thus, by increasing the dataset with additional malware and benign software, the classifier will be trained to handle a wide spectrum of different behavior amongst malware and benign executables.

Other affecting factors for our results were the varying complexity in the different feature sets. Here, the static feature set contained the least amount of features, and had weak dependencies between the features. Contrary to the static feature set, the dynamic feature set had approximately three times more features and a strong feature dependency within each entity (set of features). By joining the static and dynamic features into a combined feature set we got an overall result that was better than only using the dynamic features, however not as good as by only using static features. The averaged classification results showed that Bayes network had the best detection rate, where it showed its strengths with the highest accuracy score in the dynamic feature set. However, the best averaged accuracy score was given by the K-NN.

Although our results were improved by applying feature selection, we observed that the algorithm removed at most 99.99 % of the features from the feature sets. Both CFS and GeFS$_{\mathrm{CFS}}$ try to find the features with the highest correlation to the class label and excluding the rest. The selected static features were related to DLL dependencies such as GUI (*comctl32.dll*) and dialog functions (*comdlg32.dll*), and the selected dynamic features reflected DLL dependencies and registry activities. Similar to the static features, *comctl32.dll* was represented and the registry activites described values holding paths to shell folders and temporary Internet files. Hence, it is inadequate to keep a minimal subset of features if a malware analyst or investigator wants to perform a thorough analysis of an executable, because too much information is left out.

Our experiments showed that CFS gave the best result on the the static and dynamic feature set, whereas the GeFS$_{\mathrm{CFS}}$ proved better on the combined feature sets. One important factor to consider is the reliability of the results from the selected features. Nguyen *et al.* [124] compared the reliability of the selected features of GeFS$_{\mathrm{CFS}}$ and feature selection methods employing heuristic search (such as CFS). The reliability in feature selection was defined using the *steadiness* of the classifier's performance and *consistency* in the search algorithm. In this technical report, the GeFS$_{\mathrm{CFS}}$ proved to be superior by obtaining 100 % consistency in search for relevant features and 99.87 % steadiness for the classifier's performance. Hence, for our results, this means that it is more likely that the results from GeFS$_{\mathrm{CFS}}$ is more reliable compared to CFS.

An important aspect of keeping the behavior-related feature values was proven when we implemented a dimensionality reduction scheme using *Levenshtein distance* [125], see Appendix C.3. Here, we merged multiple entities of the same type into one. This operation was done for each sample using its dynamic features to build a reduced feature space, consisting of 75 features. By transforming string values to integers we lost the behavior-related information, which also was reflected in the poor classification results.

Even by using a small dataset with limited malicious and benign behavior, and further representing each sample with a lot of features, the possibility of overfitting was limited by applying 10-fold cross-validation to evaluate our classifiers. This implies that the system chose a random training set of 90 % and 10 % for testing, where this procedure is repeated 10 times and the results are averaged.

Overall, the computational method showed that it is quite capable of solving a two-class problem, and hence detect botnet malware with satisfying results. It is worth to notice that the method yielded a better detection rate than *Clam AntiVirus* [126]. This popular and free anti-virus gave only a detection rate of 60.22 % on our dataset (see Appendix C.5). An interesting factor we recorded was that the "simpler" classifiers such as naive-Bayes and K-NN gave an averaged accuracy that were better compared to all the other classifiers. This support the *no free lunch theorem*, where we cannot assume that a classifier outperforms another based on the classifier's characteristics. Also, it is important to notice that all the classifiers were utilized with default parameters, which means that we could end up with different results if these were optimized for each particular feature set.

6 Implications, Discussions and Conclusions

In Chapter 4 we presented the new computational method that was integrated with the *deLink* framework to solve a two-class problem of malicious and benign executables. Next, in Chapter 5, experiments were conducted to test the efficiency and effectiveness of the designed method. In this part of the thesis, the main results and findings will be discussed in regard to the stated research questions. Furthermore, theoretical and practical implications are presented, and recommendations for further research are given at the end of this chapter.

6.1 Main Results and Findings

The studies conducted in this thesis have addressed the importance of static and dynamic malware analysis by focusing on the executable's behavior. In particular, we have developed a computational method where we involved several disciplines. Methods for static and dynamic malware analysis were employed to acquire analysis reports from the collected dataset of malicious and benign executables. Data mining techniques and especially machine learning algorithms for preprocessing, feature selection and classification were required to develop a two-class classification scheme. Additionally, as part of the implementation, the method was integrated with *deLink* framework, to make it capable of analyzing behavior of malware.

An important part of this thesis has been the focus of the features used to describe the executable's behavior. In Chapter 1 we stated our research questions, where we first wanted to discover what features were the most adequate for static and dynamic analysis. Our static features were chosen based on previous studies and their results, using a PE parser (*pefile*) to acquire the dynamic-link library (DLL) and its function calls. From this information we get a good indication of the executable's behavior by studying the properties of the imported library and the called functions. Furthermore, all dynamic features where collected from an online sandbox (*Anubis*), where all activities registered were used as features. For example, activities related to registry, files, processes and network traffic were used to build a high-multidimensional feature set.

The second research question studied whether our created feature sets were disjunct or overlapping. By studying our constructed feature sets we found an overlap regarding utilized DLLs. Although, the DLL information found in the static features are more extensive, due to function calls, we complemented the dynamic feature set by constructing a combined feature set using the function calls from the static feature set. Our experiments showed that we got better results by combining the static and dynamic feature sets, compared to only using the dynamic feature set. Nevertheless, the static feature set proved to produce the overall best results.

How obfuscation techniques influenced the extracted features and their individual feature values was analyzed in order to discuss aspects related to the third research question. Obfuscation methods are frequently applied to fool static analysis methods. However, the PE parser utilized in this thesis did not analyze the executables on the assembly code level, and hence avoiding obfuscations such as polymorphism and metamorphism (see Section 2.3). Still, we experienced that packers influenced the amount of DLLs and function calls detected by the PE parser. In cases where packers were applied, the amount of DLLs was reduced to a minimum, and therefore we did not obtain a complete overview of the packed malware's behavior. Furthermore, obfuscation influences were not observed during the dynamic analysis and the malware did not detect the presence of the sandbox.

The fourth research question was related to what type of analysis approach is best suited for analysis of botnet malware. As mentioned, when we compared several classifiers on three different feature sets, we found that the best results were obtained by using the static feature set, followed by the combined and dynamic feature set. This was also the case when we applied feature selection algorithms to reduce the feature set's complexity and keep the features with the strongest relationship to the class label. However, it is important to remember that both analysis approaches have their limitations when obtaining behavior characteristics. Static analysis can easily be deceived by obfuscation techniques, and dynamic analysis will analyze only a single execution trace in a limited time period. In addition, it is known that some malware can detect if it is applied in a dynamic analysis scheme (e.g., virtual/emulated machine). Although static analysis yielded the best result in the experiments, it will give a limited view of on the expected behavior compared to the dynamic analysis. Thus, the combined approach should be best suited for detecting botnet malware.

6.2 Theoretical Considerations

This thesis has combined techniques from the disciplines of malware analysis and machine learning to develop the proposed computational method. It includes several processing steps that include algorithms for data acquisition, feature extraction, preprocessing, classification and evaluation. The current situation for a malware analyst or forensic investigator is to manually analyze a suspicious file with a wide variety of tools in order to study its behavior and decide whether the file is malicious or benign. They can benefit from the computational method to automate several analysis steps, by utilizing a collected set of labeled samples to train a classifier, and decide whether the suspicious executable is malicious or benign by solving a two-class problem based on the suspicious executable's behavior.

In contrary to most anti-virus applications that heavily depend on large databases of signatures, the computational method focused on the behavior related to a malware infection such as application programming interface (API) and system calls. Hence, not as vulnerable to obfuscation techniques compared to signature-based detection. The most primitive signature-based detection will fail to detect a slightly modified malware, since this result in a complete different signature. Therefore, behavior-based detection is a viable approach to detect new malware with similar recorded behavior. It can be concluded from the experiments that malware behavior can be obtained (in an automatic manner) by utilizing static and dynamic analysis techniques, where each sample in the dataset is analyzed to generate a static and dynamic analysis report. Based on the acquired reports, behavior-based features are extracted to build feature sets later used for training the compared classifiers.

Data preprocessing is an important task in order to prepare the features in such a way that they can be interpreted correctly by the classification algorithms. This is a crucial factor that will influence the results, since the classification algorithms may only support certain feature types such as continuous, discrete or string features. Furthermore, the experiments reflected our complex feature sets by studying the classifiers ability to accurately label the malicious and benign executables. Therefore, to increase the accuracy, and reduce the computational complexity, we utilized the features with the strongest relationship to the class label by the means of correlation-based feature selection. For an investigator a reduced feature set will improve the efficiency of the analysis by reducing the overall computational time (overhead). Especially, when dealing with a large collection of suspicious files.

Choosing the most optimal classifier for the dataset and extracted features plays a significant role for the system's outcome. By comparing a set of classifiers, malware analysts or forensic investigators have the opportunity to select the classifier that suits their specific requirements in terms of runtime efficiency, accuracy and detection rate. Therefore, the proposed method utilized a set of classifiers to study their performance when dealing with behavior-based features. After successfully applying naive-Bayes, k-nearest neighbors, C4.5, support vector machines and Bayes network as classifiers for comparison, the experiments demonstrated that Bayes network and support vector machines were preferable when dealing with complex feature sets, since they are able to handle feature dependencies. Still, simpler classifiers such as naive-Bayes, K-NN and C4.5 yielded a satisfying result, and further gave an overall better result after feature selection. This was anticipated since the features were reduced to a minimum and therefore reducing the feature dependencies.

To summarize, the computational method proposed in this thesis shows that it is a valuable tool for analyzing malware, and especially malware found in botnets. We have shown that the behavior of malware and benign software can be obtained, preprocessed and further used to train classifiers in order to reveal the true purpose of a suspicious file. As a complementary component to anti-virus software, this method can improve the system security by detecting malicious behavior with a satisfying false positive rate.

6.3 Practical Implications

The implementation of the computational method does have its practical benefits and limitations. First, the method and experiments were developed and executed based on the computational resources and time available. The performance of the implementation can be improved by two factors; either utilizing a more powerful computer system, or optimize the implementation in terms of programming language and tweaking of algorithms. Other practical issues should also be considered when deploying the method in larger scale such as malware analysis tools and machine learning algorithms that influence the overall system performance.

Conducting experiments using an online sandbox and PE parser also have its advantages and drawbacks. With *Anubis* sandbox the malware analyst or investigator is not required to spend resources on establishing an analysis environment. Instead, all suspicious files can be uploaded to the sandbox with simple automatic means. By comparing a hash sum of the submitted files to previous analyzed files, the sandbox saves computational resources by avoiding analyzing identical files twice. The main drawback encountered when the method employed automatic file uploading and report acquisition, was related to the sandbox's queuing system that prioritizes privileged users. Therefore, if a large dataset is uploaded during an active period, you may end up waiting for the result for many days. Furthermore, utilizing *pefile* avoids common pitfalls in static analysis such as obfuscation methods related to polymorphism and metamorphism by obtaining information from the different sections of the PE header (see Section 2.5.2). However, its main disadvantage is that it is unable to know in what context the system/function call has been used, compared to disassemblers.

The feature set holds comprehensive information about the behavior of each executable, and by applying these to different machine learning algorithms, a certain level of computational power is required. This aspect needs to be kept in mind if the method is going to be deployed in a different environment with for example restricted resources. Lack of memory was an issue during classification and feature selection. However, by using libraries in the implementation (e.g., *Weka API*), we avoided loading the entire feature set into memory before the classification task was initialized.

Utilizing a dataset consisting of only botnet malware in the experiment scenarios may yield a different result compared to using a larger dataset with great malware variety, for example a large variety of viruses, worms, backdoors, Trojans horses and rootkits from different malware families. Similarly, a larger dataset of benign executables may influence the outcome of the method, since those applied in the experiments were mainly network-related software. Hence, a different classification algorithm may prove to be better for a certain feature set, in terms of detection rate and accuracy, than those compared in this thesis.

Moreover, Remote Administration Tools (RATs) [127] share similar characteristics as botnet malware and are used for both good and malicious purposes. These types of tools have not been included in the method's dataset. However, their behavior should be detectable by the method, since the main difference between benign (legal) and malicious RATs is that the benign versions often require user interaction in order to install the RAT-server on the remote computer. The malicious RATs (e.g., *NetBus*, *Back Orifice* etc.) are typically deployed by Trojan horses where the malicious payload is applied with a predefined configuration. This implies that the malicious RATs will yield a different behavior, since it automatically deploys itself and awaits incoming connection from a client.

6.4 Recommendations for Further Research

This section will present possible extensions related to the proposed computational method given in this thesis. First, a larger dataset needs to be acquired in order to increase the system's accuracy and detection rate. By using a larger dataset we have the option to use another approach than cross-validation, such as splitting the dataset into a training, testing and validation set.

The acquired malware should behave differently compared to the botnet malware used in the experiments, for example by acquiring a large dataset that contains viruses, worms, Trojan horses, rootkits, backdoors etc. This can be achieved by utilizing honeypots that simulates common weaknesses found in operating systems and their running services, and store the malicious executable in a secure manner. Furthermore, collecting benign executables would require different acquisition approaches such as manually downloading from arbitrary websites, or obtaining benign executables from system folders of Windows. A third option would be to utilize a web-crawler that is developed to automatically download executables from websites.

All classifiers compared in this thesis were utilized with default parameters, and since the main focus in this thesis was related to behavior-based features, further research should perform additional experiments in order to identify an optimal classifier by adjusting the different parameters. Additionally, The two-class problem could be extended to a multi-class problem by labeling malicious samples based on malware family or functionality. If a honeypot is being used to acquire samples it would be necessary to use anti-virus software in order to label the samples, since the honeypot's function is to collect the malware that tries to infect the system. This functionality could also be automated and integrated with the data acquisition component for acquiring static and dynamic reports.

Alternative methods for feature representation are an interesting area that should be studied further. Especially the complex representation of the dynamic features, where we used 5494 features to represent one sample. Here, another solution could be to apply feature selection in order to remove redundant features, for example, features that describe similar behavior. In the static reports produced by *pefile* there is a lot of information that describes the specific executable. Therefore, further research should look for additional features that can be used to improve the classification results.

Furthermore, choosing an alternative tool for dynamic analysis could be a good idea, since the queuing system for *Anubis* will not prioritize uploaded files through their script if you do not have extra privileges. Hence, obtaining dynamic reports can take several days if you are uploading samples during days with heavy activity. A solution would be to utilize a local sandbox, such as *Truman* [128], that implements a server that simulates the Internet and use virtual machines to obtain the malware behavior. Here it would be possible to tweak the total execution time and analysis properties in such a way that a delayed or triggered activity will be obtained after the analysis.

The computational method is meant to ease the task for a malware analyst or forensic investigator to get an accurate classification whether the executable is malicious or benign. It is therefore necessary in further work to make the method automatic. This implies that the feature selection component needs to be fully integrated with the rest of the system. When this is done and a large dataset is acquired, it would be interesting to test the method against other anti-virus applications.

Bibliography

[1] Fossi, M., Turner, D., Johnson, E., Mack, T., Adams, T., Blackbird, J., Entwisle, S., Graveland, B., McKinney, D., Mulcahy, J., & Wueest, C. Symantec global internet security, threat report, trends for 2009. Technical report, 2010.

[2] Aycock, J. 2006. *Computer Viruses and Malware*. Springer.

[3] Flaglien, A. O., Franke, K., & Aarnes, A. 2011. Cross evidence malware identification in digital forensic with delink. In *Seventh Annual IFIP WG 11.9 International Conference on Digital Forensics*.

[4] Hosmer, C. 2008. Polymorphic & metamorphic malware. *Black Hat*.

[5] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., & Kruegel, C. 2009. A view on current malware behaviors. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'09, 8–8, Berkeley, CA, USA. USENIX Association.

[6] Jacob, G., Debar, H., & Filiol, E. 2008. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4, 251–266. 10.1007/s11416-008-0086-0.

[7] Jahanian, F., Bailey, M., & Cooke, E. 2009. A survey of botnet technology and defenses. *Conference For Homeland Security, Cybersecurity Applications & Technology*, 299–304.

[8] Feily, M., Shahrestani, A., & Ramadass, S. 2009. A survey of botnet and botnet detection. In *SECURWARE '09: Proceedings of the 2009 Third International Conference on Emerging Security Information, Systems and Technologies*, 268–273, Washington, DC, USA. IEEE Computer Society.

[9] Provos, N. & Holz, T. 2007. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison Wesley Professional.

[10] Vx heavens. `http://vx.netlux.org/` Last visited: 01.01.2011.

[11] Packet storm. `http://packetstormsecurity.org/` Last visited: 01.01.2011.

[12] Offensive computing. `http://www.offensivecomputing.net/` Last visited: 01.01.2011.

[13] List of portable software. `http://en.wikipedia.org/wiki/List_of_portable_software` Last visited: 06.01.2011.

[14] Vmware workstation. `http://www.vmware.com/products/workstation/` Last visited: 01.02.2011.

[15] Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., & Hamze, A. 2010. Malware detection based on mining api calls. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, 1020–1025, New York, NY, USA. ACM.

[16] Ye, Y., Wang, D., Li, T., Ye, D., & Jiang, Q. 2008. An intelligent pe-malware detection system based on association mining. *Journal in Computer Virology*, 4, 323–334. 10.1007/s11416-008-0082-4.

[17] Wang, T.-Y., Wu, C.-H., & Hsieh, C.-C. 2008. A virus prevention model based on static analysis and data mining methods. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*, 288 –293.

[18] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., & Kruegel, C. 2009. Insights into current malware behavior. *2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats*.

[19] Flaglien, A. O. Cross-computer malware detection in digital forensics. Master's thesis, Gjøvik University College, 2010.

[20] Burji, S., Liszka, K., & Chan, C. jul. 2010. Malware analysis using reverse engineering and data mining tools. 619 –624.

[21] Carrier, B. D. & Spafford, E. H. 2004. An event-based digital forensic investigation framework. In *In Proceedings of the 2004 Digital Forensic Research Workshop*.

[22] Phillips, A., Nelson, B., Enfinger, F., & Steuart, C. 2010. *Guide To Computer Forensics And Investigations, Third Edition*. Course Technology.

[23] Palmer, G. 2001. A road map for digital forensic research. *DFRWS 2001*, 27–30.

[24] Franke, K. & Srihari, S. 2008. Computational forensics: An overview. In *Computational Forensics*, Srihari, S. & Franke, K., eds, volume 5158 of *Lecture Notes in Computer Science*, 1–10. Springer Berlin / Heidelberg.

[25] Malin, C. H., Casey, E., & Aguilina, J. M. 2008. *Malware Forensics: Investigating and Analyzing Malicious Code*. Syngress.

[26] Kittilsen, J., Franke, K., & Hammerli, B. Digital forensics ontology framework. Technical report, Norwegian Information Security Laboratory, 2010.

[27] Farmer, D. & Venema, W. 2005. *Forensic Discovery*. Addison-Wesley.

[28] Preda, M. D., Christodorescu, M., Jha, S., & Debray, S. September 2008. A semantics-based approach to malware detection. *ACM Trans. Program. Lang. Syst.*, 30, 25:1–25:54.

[29] Skoudis, E. & Zeltser, L. 2003. *Malware: Fighting Malicious Code*. Prentice Hall.

[30] Szor, P. 2005. *The Art of Computer Virus Research and Defense*. Addison Wesley.

[31] Distler, D. 2007. Malware analysis: An introduction. *SANS Institute Reading Room*.

[32] Madou, M., van Put, L., & de Bosschere, K. 0-0 2006. Understanding obfuscated code. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on*, 268 –274.

[33] Yan, W., Zhang, Z., & Ansari, N. 2008. Revealing packed malware. *Security Privacy, IEEE*, 6(5), 65 –69.

[34] Tuts 4 you. `http://tuts4you.com/download.php` Last visited: 06.03.2011.

[35] Konstantinou, E. & Wolthusen, S. 2008. Metamorphic virus: Analysis and detection.

[36] Vinod, P., Laxmi, V., & Gaur, M. S. 2009. Survey on malware detection methods. In *Hack.in 2009*, 74–79.

[37] You, I. & Yim, K. 2010. Malware obfuscation techniques: A brief survey. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, 297 –300.

[38] Konstantinou, E. Metamorphic virus: Analysis and detection. Technical report, RHUL-MA-2008-02, 2008.

[39] Wong, W. & Stamp, M. 2006. Hunting for metamorphic engines. *Journal in Computer Virology*, 2, 211–229. 10.1007/s11416-006-0028-7.

[40] Szor, P. & Ferrie, P. 2001. Hunting for metamorphic. *Virus Bulletin Conference*.

[41] y0da. 2009. Lordpe. `http://www.woodmann.com/collaborative/tools/index.php/LordPE` Last visited: 06.03.2011.

[42] Russinovich, M. 2011. Procdump. `http://technet.microsoft.com/en-us/sysinternals/dd996900` Last visited: 06.03.2011.

[43] Austin, T. 2011. Botnets increased rampantly during 2010, worse to come - damballa report. `http://www.damballa.com/knowledge/Feb2011report.php` Last visited: 24.02.2011.

[44] Gu, G., Perdisci, R., Zhang, J., & Lee, W. 2008. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th conference on Security symposium*, 139–154, Berkeley, CA, USA. USENIX Association.

[45] Schiller, C. A., Binkley, J., Harley, D., Evron, G., Bradlay, T., Willems, C., & Cross, M. 2007. *Botnets - The Killer Web App*. Syngress.

[46] Zhu, Z., Lu, G., Chen, Y., Fu, Z., Roberts, P., & Han, K. 282008-aug.1 2008. Botnet research survey. In *Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International*, 967 –972.

[47] Abu Rajab, M., Zarfoss, J., Monrose, F., & Terzis, A. 2006. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, 41–52, New York, NY, USA. ACM.

[48] Project, T. H. 2007. Know your enemy: Fast-flux service networks. http://www.honeynet.org/book/export/html/130 Last visited: 25.02.2011.

[49] Grizzard, J. B., Sharma, V., Nunnery, C., Kang, B. B., & Dagon, D. 2007. Peer-to-peer botnets: overview and case study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 1–1, Berkeley, CA, USA. USENIX Association.

[50] Wu, S. X. & Banzhaf, W. January 2010. Review: The use of computational intelligence in intrusion detection systems: A review. *Appl. Soft Comput.*, 10, 1–35.

[51] Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. August 2007. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*.

[52] Gu, G., Zhang, J., & Lee, W. February 2008. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*.

[53] Masud, M., Al-khateeb, T., Khan, L., Thuraisingham, B., & Hamlen, K. 2008. Flow-based identification of botnet traffic by mining multiple log files. In *Distributed Framework and Applications, 2008. DFmA 2008. First International Conference on*, 200 –206.

[54] Strayer, W., Lapsely, D., Walsh, R., & Livadas, C. 2008. Botnet detection based on network behavior. In *Botnet Detection*, Lee, W., Wang, C., & Dagon, D., eds, volume 36 of *Advances in Information Security*, 1–24. Springer US.

[55] Seewald, A. K. & Gansterer, W. N. 2010. On the detection and identification of botnets. *Computers & Security*, 29(1), 45 – 58.

[56] Sammon, J.W., J. May 1969. A nonlinear mapping for data structure analysis. *Computers, IEEE Transactions on*, C-18(5), 401 – 409.

[57] Kornblum, J. 2006. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3(Supplement 1), 91 – 97. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[58] Freebyte's guide to free anti-virus software. http://www.freebyte.com/antivirus/ Last visited: 06.03.2011.

[59] Carrera, E. pefile. http://code.google.com/p/pefile/ Last visited: 01.03.2011.

[60] Pietrek, M. 2002. An in-depth look into the win32 portable executable file format. *MSDN Magazine*.

[61] Pietrek, M. 2002. An in-depth look into the win32 portable executable file format, part 2. *MSDN Magazine*.

[62] Schultz, M., Eskin, E., Zadok, F., & Stolfo, S. 2001. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on*, 38 –49.

[63] Gnu bin-utils. `http://www.gnu.org/software/binutils/` Last visited: 05.01.2011.

[64] Sung, A., Xu, J., Chavez, P., & Mukkamala, S. 2004. Static analyzer of vicious executables (save). In *Computer Security Applications Conference, 2004. 20th Annual*, 326 – 334.

[65] Xu, J.-Y., Sung, A., Chavez, P., & Mukkamala, S. 2004. Polymorphic malicious executable scanner by api sequence analysis. In *Hybrid Intelligent Systems, 2004. HIS '04. Fourth International Conference on*, 378 – 383.

[66] Shen, Y.-D., Zhang, Z., & Yang, Q. 2002. Objective-oriented utility-based association mining. In *Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on*, 426 – 433.

[67] Breiman, L. 2001. Random forests. *Machine Learning*, 45, 5–32. 10.1023/A:1010933404324.

[68] Duda, R. O., Hart, P. E., & Stork, D. G. 2001. *Pattern Classification*. Wiley.

[69] song Zou, M., sheng Han, L., wen Liu, Q., & Liu, M. 2009. Behavior-based malicious executables detection by multi-class svm. In *Information, Computing and Telecommunication, 2009. YC-ICT '09. IEEE Youth Conference on*, 331 –334.

[70] Wang, C., Pang, J., Zhao, R., Fu, W., & Liu, X. 2009. Malware detection based on suspicious behavior identification. In *Education Technology and Computer Science, 2009. ETCS '09. First International Workshop on*, volume 2, 198 –202.

[71] Willems, C., Holz, T., & Freiling, F. 2007. Toward automated dynamic malware analysis using cwsandbox. *Security Privacy, IEEE*, 5(2), 32 –39.

[72] Kendall, K. & McMillan, C. 2007. Practical malware analysis. *Black Hat*.

[73] Wireshark - go deep. `http://www.wireshark.org/` Last visited: 06.03.2011.

[74] Christodorescu, M., Jha, S., & Kruegel, C. 2007. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ESEC-FSE '07, 5–14, New York, NY, USA. ACM.

[75] Hu, Y., Chen, L., Xu, M., Zheng, N., & Guo, Y. 2008. Unknown malicious executables detection based on run-time behavior. In *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, volume 4, 391 –395.

[76] Ding, J., Jin, J., Bouvry, P., Hu, Y., & Guan, H. May 2009. Behavior-based proactive detection of unknown malicious codes. In *Internet Monitoring and Protection, 2009. ICIMP '09. Fourth International Conference on*, 72 –77.

[77] Nair, V. P., Jain, H., Golecha, Y. K., Gaur, M. S., & Laxmi, V. 2010. Medusa: Metamorphic malware dynamic analysis usingsignature from api. In *Proceedings of the 3rd international conference on Security of information and networks*, SIN '10, 263–269, New York, NY, USA. ACM.

[78] Park, Y., Reeves, D., Mulukutla, V., & Sundaravel, B. 2010. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, 45:1–45:4, New York, NY, USA. ACM.

[79] Dinaburg, A., Royal, P., Sharif, M., & Lee, W. 2008. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, 51–62, New York, NY, USA. ACM.

[80] Ahmed, F., Hameed, H., Shafiq, M. Z., & Farooq, M. 2009. Using spatio-temporal information in api calls with machine learning algorithms for malware detection. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, AISec '09, 55–62, New York, NY, USA. ACM.

[81] Jain, A., Duin, R., & Mao, J. January 2000. Statistical pattern recognition: a review. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1), 4 –37.

[82] Gong, L. 1997. Java security: present and near future. *Micro, IEEE*, 17(3), 14 –19.

[83] Carpenter, M., Liston, T., & Skoudis, E. 2007. Hiding virtualization from attackers and malware. *Security Privacy, IEEE*, 5(3), 62 –65.

[84] Rieck, K., Holz, T., Willems, C., Dussel, P., & Laskov, P. 2008. Learning and classification of malware behavior. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Zamboni, D., ed, volume 5137 of *Lecture Notes in Computer Science*, 108–125. Springer Berlin / Heidelberg.

[85] Rieck, K., Trinius, P., Willems, C., & Holz, T. Automatic analysis of malware behavior using machine learning. Technical report, 2009.

[86] Fernandez, F. & Isasi, P. 2008. Nearest prototype classification of noisy data. *Artificial Intelligence Review*, 30, 53–66. 10.1007/s10462-009-9116-7.

[87] Bayer, U., Moser, A., Kruegel, C., & Kirda, E. 2006. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2, 67–77. 10.1007/s11416-006-0012-2.

[88] Bayer, U., Comapretti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. 2009. Scalable, behavior-based malware clustering. 1–18.

[89] Firdausi, I., Lim, C., Erwin, A., & Nugroho, A. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*, 201 –203.

[90] Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., & Nazario, J. 2007. Automated classification and analysis of internet malware. In *Proceedings of the 10th international conference on Recent advances in intrusion detection*, RAID'07, 178–197, Berlin, Heidelberg. Springer-Verlag.

[91] Apel, M., Bockermann, C., & Meier, M. 2009. Measuring similarity of malware behavior. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, 891 –898.

[92] Ida pro - the multi-processor, multi-os, interactive disassembler, by datarescue. http://www.www.datarescue.com/ Last visited: 09.01.2011.

[93] Chan, C.-C. & Sengottiyan, S. 2003. Blem2: learning bayes' rules from examples using rough sets. In *Fuzzy Information Processing Society, 2003. NAFIPS 2003. 22nd International Conference of the North American*, 187 – 190.

[94] Kononenko, I. & Kukar, M. 2007. *Machine Learning and Data Mining - Introduction to Principles and Algorithms*. Horwood.

[95] Theodoridis, S. & Koutroumbas, K. 2009. *Pattern Recognition - Fourth Edition*. Elsevier.

[96] Hand, D., Mannila, H., & Smyth, P. 2001. *Principles of Data Mining*. MIT Press.

[97] Cross industry standard process for data mining. http://www.crisp-dm.org/Process/index.htm Last visited: 22.03.2011.

[98] Witten, I. H. & Frank, E. 2005. *Data Mining - Practical Machine Learning Tools and Techniques*. Elsevier.

[99] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. November 2009. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11, 10–18.

[100] Ince, K. & Klawonn, F. 2010. Attribute value selection considering the minimum description length approach and feature granularity. In *Proceedings of the Computational intelligence for knowledge-based systems design, and 13th international conference on Information processing and management of uncertainty*, IPMU'10, 250–259, Berlin, Heidelberg. Springer-Verlag.

[101] Pyle, D. 1999. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers.

[102] Kotsiantis, S. B., Kanellopoulos, D., & Pintelas, P. E. 2006. Data preprocessing for supervised learning. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE*, VOLUME 1, 1–7.

[103] Hall, M. A. *Correlation-based Feature Selection for Machine Learning*. PhD thesis, 1999.

[104] Nguyen, H., Franke, K., & Petrovic, S. 2010. Improving effectiveness of intrusion detection by correlation feature selection. *Availability, Reliability and Security, International Conference on*, 0, 17–24.

[105] Nguyen, H. T., Franke, K., & Petrovic, S. 2010. Towards a generic feature-selection measure for intrusion detection. *Pattern Recognition, International Conference on*, 0, 1529–1532.

[106] Peng, H., Long, F., & Ding, C. aug. 2005. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8), 1226 –1238.

[107] Kohavi, R. & Quinlan, J. R. *Data mining tasks and methods: Classification: decision-tree discovery*, 267–276. Oxford University Press, Inc., New York, NY, USA, 2002.

[108] Rokach, L. & Maimon, O. nov. 2005. Top-down induction of decision trees classifiers - a survey. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(4), 476 – 487.

[109] Uusitalo, L. 2007. Advantages and challenges of bayesian networks in environmental modelling. *Ecological Modelling*, 203(3-4), 312 – 318.

[110] Top 10 malware registry launchpoints. `http://www.f-secure.com/weblog/archives/00001207.html` Last visited: 05.05.2011.

[111] Skoudis, E. & Liston, T. 2006. *Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition)*. Prentice Hall.

[112] Appendix b: Standard .exe files and associated dlls. `http://technet.microsoft.com/en-us/library/cc768380.aspx` Last visited: 22.04.2011.

[113] Microsoft portable executable and common object file format specification. `http://msdn.microsoft.com/en-us/windows/hardware/gg463119` Last visited: 04.03.2011.

[114] Chang, C.-C. & Lin, C.-J. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[115] Ubuntu. `http://www.ubuntu.com/` Last visited: 10.05.2011.

[116] Python programming language. `http://www.python.org/` Last visited: 10.05.2011.

[117] The eclipse foundation. `http://www.eclipse.org/` Last visited: 10.05.2011.

[118] pendriveapps. `http://www.pendriveapps.com/` Last visited: 05.05.2011.

[119] Knowles, D. W32.spybot.worm. `http://www.symantec.com/security_response/writeup.jsp?docid=2003-053013-5943-99` Last visited: 09.02.2011.

[120] O'Connor, J. Trojan.anserin. `http://www.symantec.com/security_response/writeup.jsp?docid=2005-112315-0608-99` Last visited: 09.02.2011.

[121] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., & Vigna, G. 2009. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, 635–647, New York, NY, USA. ACM.

[122] Sevcenco, S. Backdoor.sdbot. `http://www.symantec.com/security_response/writeup.jsp?docid=2002-051312-3628-99` Last visited: 09.02.2011.

[123] Bouckaert, R. *Bayesian Network Classifiers in Weka*, 2004.

[124] Nguyen, H. T., Franke, K., & Petrovic, S. Reliability in feature-selection process for intrusion detection. Technical report, Norwegian Information Security Laboratory, 2011.

[125] Navarro, G. March 2001. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33, 31–88.

[126] Clam antivirus. `http://www.clamav.net/lang/en/` Last visited: 15.06.2011.

[127] Chen, Z., Wei, P., & Delis, A. 2008. Catching remote administration trojans (rats). *Software: Practice and Experience*, 38(7), 667–703.

[128] Stewart, J. Behavioural malware analysis using sandnets. *Computer Fraud & Security*, 2006.

A Entity Features

A.1 DLL Dependencies

Feature	**Type**	**Description**
base_address	String	Base addresses of DLLs
base_name	String	DLL name (e.g., kernel32.dll)
full_name	String	Full path of DLL (e.g., C:\WINDOWS \system32\kernel32.dll)
is_load_time_dependency	String	Load-time or run-time dependent
size	String	Size of DLL

Table 17: Features in *loaded_dll* entity

A.2 Registry Activities

Feature	**Type**	**Description**
name	String	Name of created registry key

Table 18: Feature in *reg_key_created* entity

Feature	**Type**	**Description**
count	String	Number of registry keys modified
description	String	Description of key modified (e.g., internet_settings)
key	String	Key modified
value_data	String	Full value data name
value_name	String	Value data name

Table 19: Features in *reg_value_modified* entity

Feature	Type	Description
count	String	Number of registry keys read
key	String	Key read
value_data	String	Full value data name
value_name	String	Value data name

Table 20: Features in *reg_value_read* entity

A.3 File Activities

Feature	Type	Description
name	String	Name of created file

Table 21: Feature in *file_created* entity

Feature	Type	Description
description	String	Description of operation done to file
name	String	Name of modified file

Table 22: Feature in *file_modified* entity

Feature	Type	Description
description	String	Description of operation done to file
name	String	Name of deleted file

Table 23: Feature in *file_deleted* entity

Feature	Type	Description
name	String	Name of read file

Table 24: Feature in *file_read* entity

Feature	Type	Description
file_name	String	Memory mapped file, may reflect DLL usage

Table 25: Feature in *subject_object_created* entity

Feature	Type	Description
control_code	String	Reflects operations done by driver
count	String	Number of times driver was used
file	String	Driver name

Table 26: Feature in *device_control_communication* entity

Feature	Type	Description
control_code	String	Operation done by file system control (e.g., inter-process)
count	String	Number of operations that were done
file	String	File used in operation

Table 27: Feature in *fs_control_communication* entity

A.4 Process and Thread Activities

Feature	Type	Description
number_of_threads	String	Number of threads created
time	String	Time thread was alive

Table 28: Feature in *thread_status* entity

Feature	Type	Description
process	String	Process created by thread

Table 29: Feature in *remote_thread_created* entity

Feature	Type	Description
cmd_line	String	Command for executing process
description	String	Description of executed process (e.g., process_spawn)
exe_name	String	Name of process

Table 30: Feature in *process_created* entity

A.5 Network Activity

Feature	Type	Description
close_time	String	Date and time when socket was closed
create_time	String	Date and time when socket was opened
created_by_thread	String	Describes which thread that created the socket
foreign_ip	String	Foreign IP address
foreign_port	String	Foreign port
is_listening	String	Reflects whether socket waits for incoming connections
local_ip	String	Local IP address
local_port	String	Local port

Table 31: Feature in *socket* entity

Feature	Type	Description
name	String	Name of domain name server (e.g., hig.no)
result	String	IP address of resolved DNS
successful	String	Reflects whether the query was successful
type	String	Type of DNS query (e.g., mail exchanger recoder, MX)

Table 32: Feature in *dns_query* entity

Feature	Type	Description
content	String	Content of conversation
description	String	Description of SMTP activity (e.g., spambot)
dest_ip	String	Destination IP address
dest_port	String	Destination port
recipient	String	Mail address of recipient
sender	String	Mail address of sender
server_reply	String	Server reply
src_ip	String	Source IP address
src_port	String	Source port
subject	String	Subject of mail

Table 33: Feature in *smtp_conversation* entity

Feature	Type	Description
dest_ip	String	Destination IP address
dest_port	String	Destination port
hostname	String	Hostname in conversation (e.g., microsoft.com)
src_ip	String	Source IP address
src_port	String	Source port

Table 34: Feature in *http_conversation* entity

Feature	Type	Description
dest_ip	String	Destination IP address
dest_port	String	Destination port
org_bytes_sent	String	Amount of bytes sent
res_bytes_sent	String	Amount of bytes received
src_ip	String	Source IP address
src_port	String	Source port
state	String	State of TCP connection

Table 35: Feature in *tcp_conversation* entity

Feature	Type	Description
dest_ip	String	Destination IP address
dest_port	String	Destination port
org_bytes_sent	String	Amount of bytes sent
res_bytes_sent	String	Amount of bytes received
src_ip	String	Source IP address
src_port	String	Source port
state	String	State of UDP connection

Table 36: Feature in *udp_conversation* entity

B UML Diagrams

B.1 Parser

Comment: This is the implementation part which is responsible of parsing the XML-files and txt-files that are retrieved from *pefile* and *Anubis*.

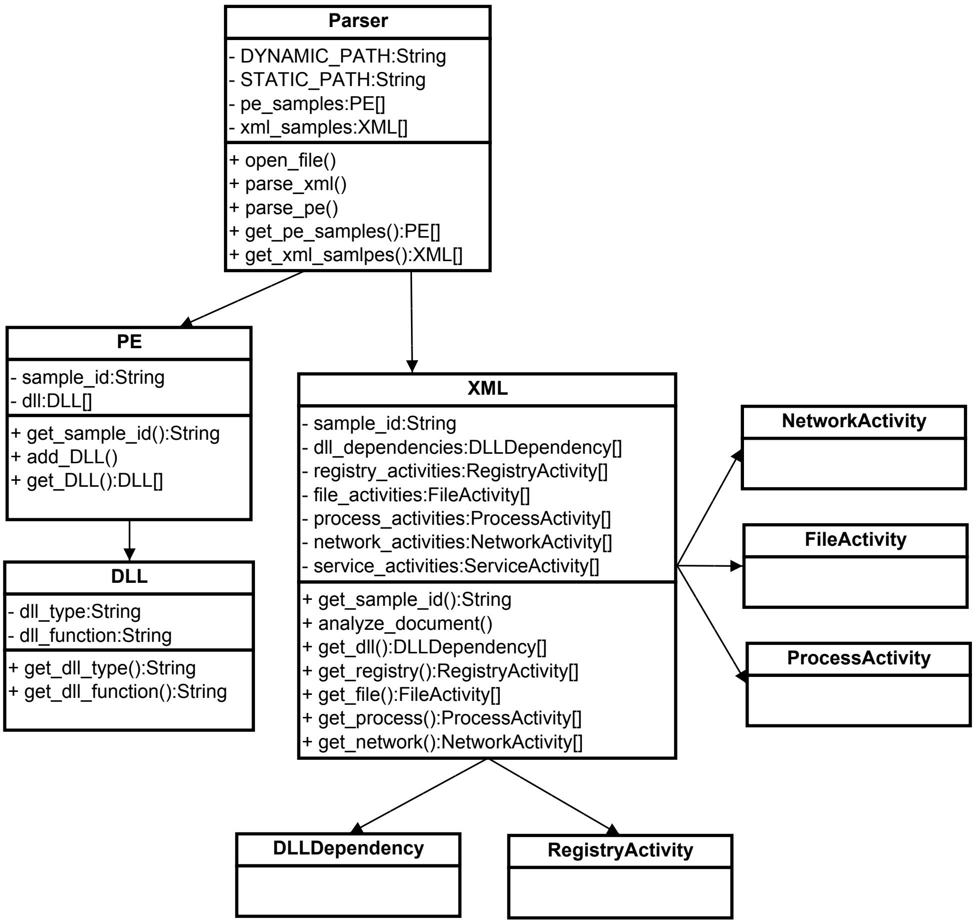

Parser
- DYNAMIC_PATH:String
- STATIC_PATH:String
- pe_samples:PE[]
- xml_samples:XML[]
+ open_file()
+ parse_xml()
+ parse_pe()
+ get_pe_samples():PE[]
+ get_xml_samlpes():XML[]
PE
- sample_id:String
- dll:DLL[]
+ get_sample_id():String
+ add_DLL()
+ get_DLL():DLL[]
DLL
- dll_type:String
- dll_function:String
+ get_dll_type():String
+ get_dll_function():String
XML
- sample_id:String
- dll_dependencies:DLLDependency[]
- registry_activities:RegistryActivity[]
- file_activities:FileActivity[]
- process_activities:ProcessActivity[]
- network_activities:NetworkActivity[]
- service_activities:ServiceActivity[]
+ get_sample_id():String
+ analyze_document()
+ get_dll():DLLDependency[]
+ get_registry():RegistryActivity[]
+ get_file():FileActivity[]
+ get_process():ProcessActivity[]
+ get_network():NetworkActivity[]
NetworkActivity
FileActivity
ProcessActivity
DLLDependency
RegistryActivity

Figure 30: UML diagram of the Parser

B.2 Feature Extractor

Comment: This part is responsible of extracting features and building three feature sets. These are stored as individual ARFF-files.

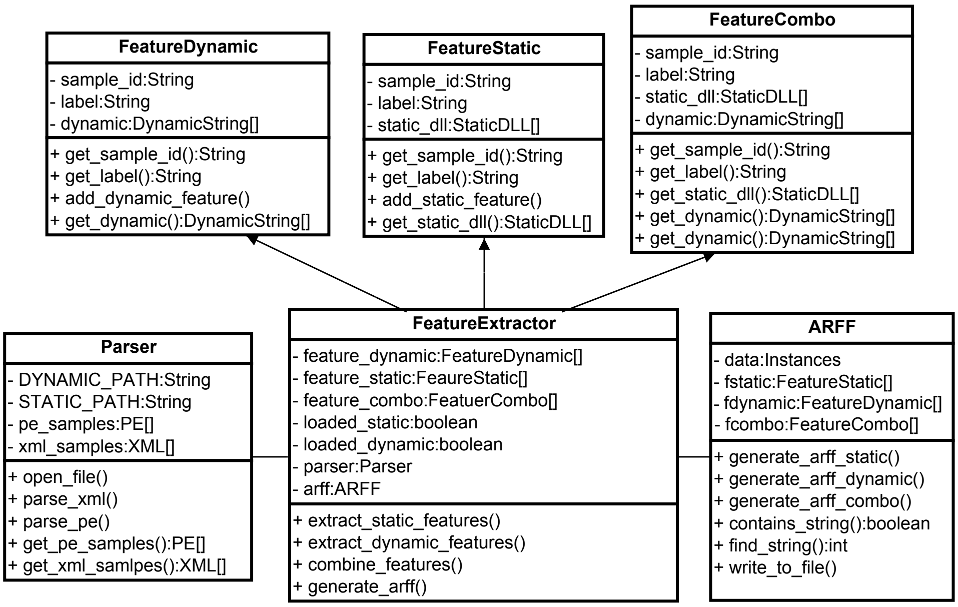


Figure 31: UML diagram of the Feature Extractor

B.3 Evaluator

Comment: The Evaluator is responsible of loading a set of ARFF-files and use these to evaluate five classifiers; Naive-Bayes, K-NN, C4.5, SVM and Bayes Network.

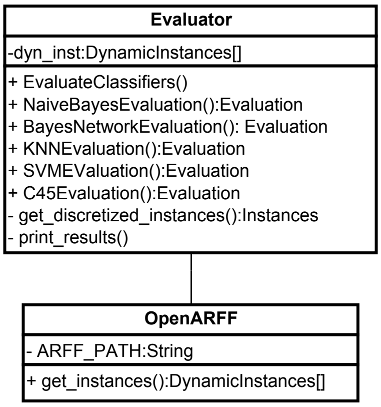
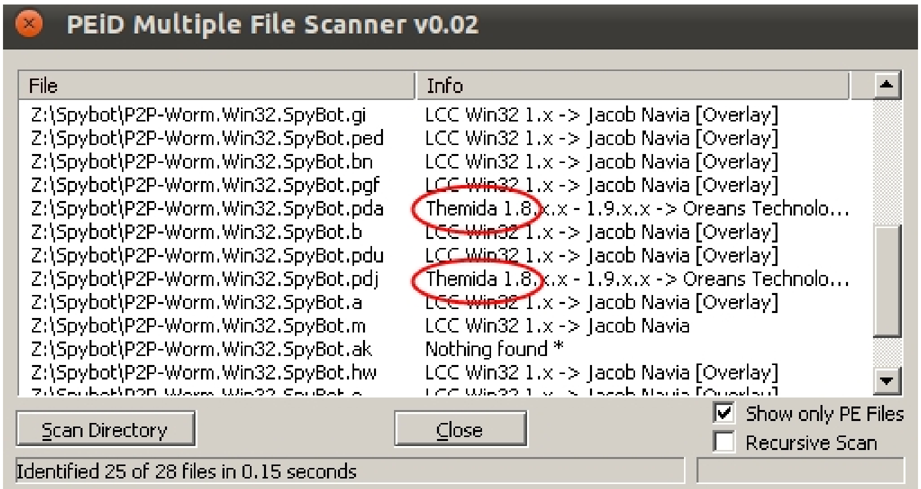

Evaluator
-dyn_inst:DynamicInstances[]
+ EvaluateClassifiers()
+ NaiveBayesEvaluation():Evaluation
+ BayesNetworkEvaluation(): Evaluation
+ KNNEvaluation():Evaluation
+ SVMEValuation():Evaluation
+ C45Evaluation():Evaluation
- get_discretized_instances():Instances
- print_results()
OpenARFF
- ARFF_PATH:String
+ get_instances():DynamicInstances[]

Figure 32: UML diagram of the Evaluator

C Proof-of-Concept Supplements

C.1 Packers Identification

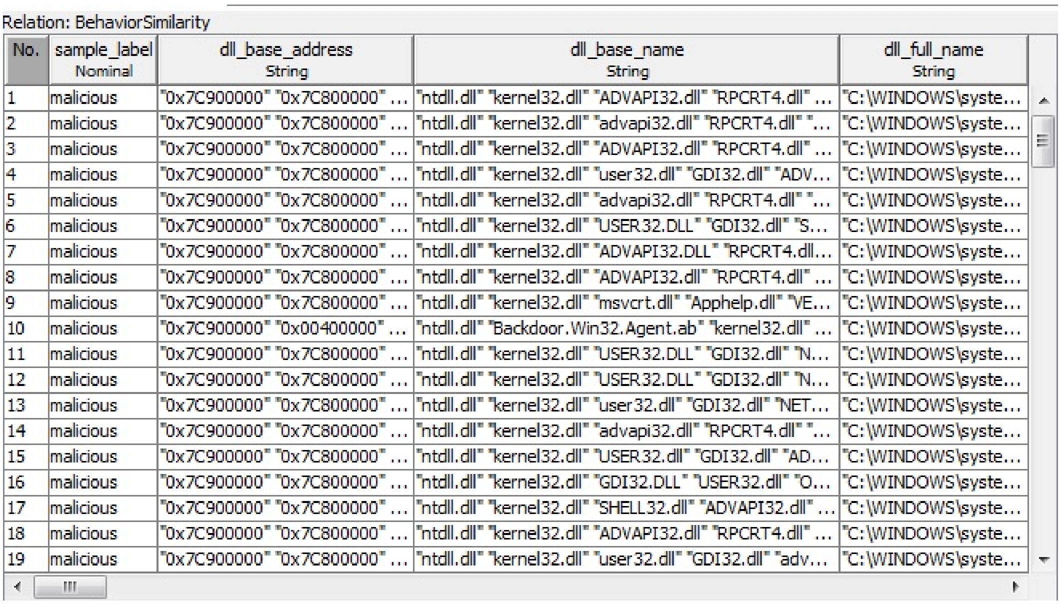


File	Info
Z:\Spybot\P2P-Worm.Win32.SpyBot.gi	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.ped	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.bn	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.pgf	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.pda	Themida 1.8.x.x - 1.9.x.x -> Oreans Technolo...
Z:\Spybot\P2P-Worm.Win32.SpyBot.b	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.pdu	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.pdj	Themida 1.8.x.x - 1.9.x.x -> Oreans Technolo...
Z:\Spybot\P2P-Worm.Win32.SpyBot.a	LCC Win32 1.x -> Jacob Navia [Overlay]
Z:\Spybot\P2P-Worm.Win32.SpyBot.m	LCC Win32 1.x -> Jacob Navia
Z:\Spybot\P2P-Worm.Win32.SpyBot.ak	Nothing found *
Z:\Spybot\P2P-Worm.Win32.SpyBot.hw	LCC Win32 1.x -> Jacob Navia [Overlay]

Identified 25 of 28 files in 0.15 seconds

Figure 33: PEiD results for SpyBot variants

C.2 ARFF Viewer



Relation: BehaviorSimilarity

No.	sample_label Nominal	dll_base_address String	dll_base_name String	dll_full_name String
1	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "ADVAPI32.dll" "RPCRT4.dll" ...	"C:\WINDOWS\syste...
2	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "advapi32.dll" "RPCRT4.dll" "...	"C:\WINDOWS\syste...
3	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "ADVAPI32.dll" "RPCRT4.dll" ...	"C:\WINDOWS\syste...
4	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "user32.dll" "GDI32.dll" "ADV...	"C:\WINDOWS\syste...
5	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "advapi32.dll" "RPCRT4.dll" "...	"C:\WINDOWS\syste...
6	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "USER32.DLL" "GDI32.dll" "S...	"C:\WINDOWS\syste...
7	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "ADVAPI32.DLL" "RPCRT4.dll...	"C:\WINDOWS\syste...
8	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "ADVAPI32.dll" "RPCRT4.dll" ...	"C:\WINDOWS\syste...
9	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "msvcrt.dll" "Apphelp.dll" "VE...	"C:\WINDOWS\syste...
10	malicious	"0x7C900000" "0x00400000" ...	"ntdll.dll" "Backdoor.Win32.Agent.ab" "kernel32.dll" ...	"C:\WINDOWS\syste...
11	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "USER32.DLL" "GDI32.dll" "N...	"C:\WINDOWS\syste...
12	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "USER32.DLL" "GDI32.dll" "N...	"C:\WINDOWS\syste...
13	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "user32.dll" "GDI32.dll" "NET...	"C:\WINDOWS\syste...
14	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "advapi32.dll" "RPCRT4.dll" "...	"C:\WINDOWS\syste...
15	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "USER32.dll" "GDI32.dll" "AD...	"C:\WINDOWS\syste...
16	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "GDI32.DLL" "USER32.dll" "O...	"C:\WINDOWS\syste...
17	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "SHELL32.dll" "ADVAPI32.dll" ...	"C:\WINDOWS\syste...
18	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "ADVAPI32.dll" "RPCRT4.dll" ...	"C:\WINDOWS\syste...
19	malicious	"0x7C900000" "0x7C800000" ...	"ntdll.dll" "kernel32.dll" "user32.dll" "GDI32.dll" "adv...	"C:\WINDOWS\syste...

Figure 34: ARFF viewer displaying concatenated strings in dynamic feature set before preprocessing

C.3 Levenshtein Distance Attempt

The Levenshtein distance was used to merge the entities in this manner:

$$\mathit{Sample} = \begin{pmatrix} \mathit{Entity}_{a_1} = \{\mathit{value}_1, ..., \mathit{value}_n\} \\ \mathit{Entity}_{a_2} = \{\mathit{value}_1, ..., \mathit{value}_n\} \\ \vdots \\ \mathit{Entity}_{a_n} = \{\mathit{value}_1, ..., \mathit{value}_n\} \end{pmatrix}$$

Here we have entities of the same type, namely Entity_a. Applying Levenshtein distance between the two value_1's will return an integer of how similar they are to each other. This information can be used to create an generic entity:

$$\mathit{Entity}_{a_{generic}} = \left(\ \mathit{value}_{1_{generic}}, ..., \mathit{value}_{n_{generic}} \ \right)$$

This will serve as an intra-distance measure, where we store the sum of the Levenshtein Distances for the generic value. Unfortunately the results were poor with Bayes Network and 10-fold cross-validation:

- True positive rate = 68%
- False positive rate = 16%

Note that the system was not completely implemented at this point and it tried to classify the malware based on malware-family class labels. Thus, it only retrieved these metrics.

C.4 DLL Import In Code

```
// example code for loading DLL's in code withouth using DLL
// -references defined in project.

#include "stdafx.h"
#include "windows.h"

int _tmain(int argc, _TCHAR* argv[])
{
        HMODULE mod;
        FARPROC fn;
        DWORD erno;

        // trying to load DLL
        mod=LoadLibrary(L"<<PATH>>\\test.dll");
        if (!mod)
        {
                erno=GetLastError();
                printf("Loadlibrary error: %x\n", HRESULT_FROM_WIN32(erno));
                exit(1);
        }
        // pointer to DLL
        fn=GetProcAddress(mod,"test");
        if (!fn)
        {
                printf("GetProcAddress error: %x\n",HRESULT_FROM_WIN32(GetLastError()));
                exit(1);
        }
        // response from function
        printf("Response from DLL: %d\n",fn());
        return 0;
}
```

C.5 Clam AntiVirus Results

peter@peter-A8J:~$ clamscan /home/peter/all/
LibClamAV Warning: ***

```
LibClamAV Warning: ***  This version of the ClamAV engine is outdated.     ***
LibClamAV Warning: *** DON’T PANIC! Read http://www.clamav.net/support/faq ***
LibClamAV Warning: ***********************************************************
LibClamAV Warning: ***********************************************************
LibClamAV Warning: ***  This version of the ClamAV engine is outdated.     ***
LibClamAV Warning: *** DON’T PANIC! Read http://www.clamav.net/support/faq ***
LibClamAV Warning: ***********************************************************
/home/peter/all/Halite.exe: OK
/home/peter/all/MACAddressView.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.e: Trojan.Spy.Sinowal-24 FOUND
/home/peter/all/Nassau.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.ce: OK
/home/peter/all/ozSync.exe: OK
/home/peter/all/Backdoor.Win32.Agent.abfh: OK
/home/peter/all/PAGEANT.EXE: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.at: OK
/home/peter/all/pn.exe: OK
/home/peter/all/Backdoor.Win32.Agent.acfx: Trojan.Agent-179822 FOUND
/home/peter/all/Backdoor.Win32.Agent.adnh: OK
/home/peter/all/whoistd.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.peo: OK
/home/peter/all/angry_ipscan.exe: OK
/home/peter/all/nPOPuk.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.gz: Worm.Puce.D FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.t: Trojan.Spybot.gen-2 FOUND
/home/peter/all/Backdoor.Win32.Agent.abka: Trojan.Agent-66928 FOUND
/home/peter/all/FTPWanderer.exe: OK
/home/peter/all/GetMyIp.exe: OK
/home/peter/all/VBPing.exe: OK
/home/peter/all/HydraIRC.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.jc: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.ew: Trojan.Spybot.gen-2 FOUND
/home/peter/all/Webdeling.exe: OK
/home/peter/all/Backdoor.Win32.Agent.aawp: OK
/home/peter/all/Backdoor.Win32.Agent.abv: Adware.Stud-1 FOUND
/home/peter/all/Backdoor.Win32.Agent.adei: OK
/home/peter/all/Backdoor.Win32.Agent.aao: Trojan.Hupigon-9116 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.b: Trojan.Spy.Sinowal-24 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.gen: Trojan.Spybot.gen-2 FOUND
/home/peter/all/eMule.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.l: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/Backdoor.Win32.Agent.abcf: OK
/home/peter/all/WinSCP.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pbw: Trojan.SdBot-13167 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.cc: OK
/home/peter/all/Backdoor.Win32.Agent.adjs: OK
```

```
/home/peter/all/megairc.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.cl: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.eu: Trojan.Spybot.gen-2 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.k: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/MacMakeUp.exe: OK
/home/peter/all/0irc.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.paz: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.n: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/LANMessenger.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.ih: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.av: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/Tcpview.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pdy: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.m: Trojan.Spy.Sinowal-46 FOUND
/home/peter/all/PUTTYGEN.EXE: OK
/home/peter/all/DropUpLoad.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.n: Trojan.Spybot.gen-2 FOUND
/home/peter/all/eToolz.exe: OK
/home/peter/all/Backdoor.Win32.Agent.achf: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.ci: OK
/home/peter/all/PLINK.EXE: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.aq: Trojan.Spy.Sinowal-44 FOUND
/home/peter/all/nPOP.exe: OK
/home/peter/all/NetRouteView.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.br: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/PortableWackGet.exe: OK
/home/peter/all/popcorn.exe: OK
/home/peter/all/netscan.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.k: Trojan.Spybot.gen-2 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.pdv: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.gi: Trojan.Spybot.gen-3 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.ped: OK
/home/peter/all/Backdoor.Win32.Agent.aahl: OK
/home/peter/all/Backdoor.Win32.Agent.aal: OK
/home/peter/all/nPOPuk_win98.exe: OK
/home/peter/all/Backdoor.Win32.Agent.aaa: Trojan.Agent-131839 FOUND
/home/peter/all/Backdoor.Win32.Agent.adjv: Trojan.Delf-8261 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.au: OK
/home/peter/all/Backdoor.Win32.Agent.adbl: Trojan.Vundo-10901 FOUND
/home/peter/all/ipscan.exe: OK
/home/peter/all/PSFTP.EXE: OK
/home/peter/all/DbxConv.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.bn: Trojan.Spybot.gen-3 FOUND
/home/peter/all/Backdoor.Win32.Agent.adhy: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.je: OK
/home/peter/all/Backdoor.Win32.Agent.aagp: Trojan.Agent-192978 FOUND
```

```
/home/peter/all/Backdoor.Win32.Agent.acnz: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.i: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.bf: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/Backdoor.Win32.Agent.adow: Trojan.Dropper-18605 FOUND
/home/peter/all/Backdoor.Win32.Agent.aap: Trojan.Agent-115740 FOUND
/home/peter/all/hfs.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.bb: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.ay: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pgf: Trojan.Spybot-116 FOUND
/home/peter/all/smsniff.exe: OK
/home/peter/all/Backdoor.Win32.Agent.adne: Trojan.Dropper-18605 FOUND
/home/peter/all/Backdoor.Win32.Agent.aama: Trojan.Agent-192978 FOUND
/home/peter/all/PSCP.EXE: OK
/home/peter/all/Backdoor.Win32.Agent.abck: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.j: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/FollowMeIPLite.exe: OK
/home/peter/all/Backdoor.Win32.Agent.adb: Trojan.Agent-9071 FOUND
/home/peter/all/Backdoor.Win32.Agent.adqc: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pda: Trojan.SdBot-13168 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.b: Trojan.Spybot.gen-2 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.h: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.bk: OK
/home/peter/all/Backdoor.Win32.Agent.ab: Trojan.Agent-605 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.pdu: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.cj: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pdj: Trojan.SdBot-13167 FOUND
/home/peter/all/Tcpvcon.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.a: Trojan.Spybot.gen-2 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.m: Trojan.Spybot.gen-2 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.ak: OK
/home/peter/all/avant.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.ar: Trojan.Spy.Sinowal-44 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.hw: Trojan.Spybot.gen-2 FOUND
/home/peter/all/ipnetinfo.exe: OK
/home/peter/all/NetworkStuff.exe: OK
/home/peter/all/Fact200.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.o: Trojan.Spybot.gen-2 FOUND
/home/peter/all/Backdoor.Win32.Agent.acnq: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.ae: Trojan.Spy.Sinowal-40 FOUND
/home/peter/all/RouterPassView.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.o: Trojan.Spy.Sinowal-25 FOUND
/home/peter/all/Trojan-PSW.Win32.Sinowal.aa: Trojan.Spy.Sinowal-33 FOUND
/home/peter/all/Backdoor.Win32.Agent.adfc: OK
/home/peter/all/connectionmonitoring.exe: OK
/home/peter/all/qm.exe: OK
/home/peter/all/Backdoor.Win32.Agent.adf: Trojan.Agent-526 FOUND
```

```
/home/peter/all/PUTTY.EXE: OK
/home/peter/all/Backdoor.Win32.Agent.acrv: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.af: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/IPInfoOffline.exe: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.bd: Trojan.Spy.Sinowal-33 FOUND
/home/peter/all/Backdoor.Win32.Agent.acly: Trojan.Agent-92729 FOUND
/home/peter/all/P2P-Worm.Win32.SpyBot.pdl: OK
/home/peter/all/DNSBench.exe: OK
/home/peter/all/P2P-Worm.Win32.SpyBot.pdp: OK
/home/peter/all/Trojan-PSW.Win32.Sinowal.cn: Trojan.Spy.Sinowal-26 FOUND
/home/peter/all/PixaMSN.exe: OK
/home/peter/all/Backdoor.Win32.Agent.aaua: Trojan.Dropper-18605 FOUND

----------- SCAN SUMMARY -----------
Known viruses: 972002
Engine version: 0.96.5
Scanned directories: 1
Scanned files: 143
Infected files: 56
Data scanned: 62.20 MB
Data read: 55.05 MB (ratio 1.13:1)
Time: 32.904 sec (0 m 32 s)
```

D Classification Results

D.1 Complete Feature Sets

out_combo.arff with Naive Bayes
True positives: 83.0, True positive rate: 0.916083916083916
False negatives: 10.0, False negative rate: 0.08391608391608392
False positives: 2.0, False positive rate: 0.06361079780434618
True negatives: 48.0, True negative rate: 0.9363892021956538
Detection rate: 0.8924731182795699, Accuracy: 0.916083916083916
Precision: 0.9244131122427268, Recall: 0.916083916083916

out_combo.arff with K-NN
True positives: 89.0, True positive rate: 0.8951048951048951
False negatives: 4.0, False negative rate: 0.1048951048951049
False positives: 11.0, False positive rate: 0.15811564779306717
True negatives: 39.0, True negative rate: 0.8418843522069329
Detection rate: 0.956989247311828, Accuracy: 0.8951048951048951
Precision: 0.8959359245405756, Recall: 0.8951048951048951

out_combo.arff with C4.5
True positives: 82.0, True positive rate: 0.8601398601398601
False negatives: 11.0, False negative rate: 0.13986013986013987
False positives: 9.0, False positive rate: 0.15841943003233325
True negatives: 41.0, True negative rate: 0.8415805699676667
Detection rate: 0.8817204301075269, Accuracy: 0.8601398601398601
Precision: 0.8617152078690541, Recall: 0.8601398601398601

out_combo.arff with SVM
True positives: 91.0, True positive rate: 0.8321678321678322
False negatives: 2.0, False negative rate: 0.16783216783216784
False positives: 22.0, False positive rate: 0.2936732085119182
True negatives: 28.0, True negative rate: 0.7063267914880818
Detection rate: 0.978494623655914, Accuracy: 0.8321678321678322
Precision: 0.850073230604204, Recall: 0.8321678321678322

out_combo.arff with Bayes Network
True positives: 91.0, True positive rate: 0.8671328671328671
False negatives: 2.0, False negative rate: 0.13286713286713286
False positives: 17.0, False positive rate: 0.22863824347695316
True negatives: 33.0, True negative rate: 0.771361756523047
Detection rate: 0.978494623655914, Accuracy: 0.8671328671328671
Precision: 0.8776501276501276, Recall: 0.8671328671328671

```
out_dynamic.arff with Naive Bayes
True positives: 78.0, True positive rate: 0.8671328671328671
False negatives: 15.0, False negative rate: 0.13286713286713286
False positives: 4.0, False positive rate: 0.1084231897135123
True negatives: 46.0, True negative rate: 0.8915768102864877
Detection rate: 0.8387096774193549, Accuracy: 0.8671328671328671
Precision: 0.8822960326358966, Recall: 0.8671328671328671

out_dynamic.arff with K-NN
True positives: 80.0, True positive rate: 0.8741258741258742
False negatives: 13.0, False negative rate: 0.1258741258741259
False positives: 5.0, False positive rate: 0.11391082036243326
True negatives: 45.0, True negative rate: 0.8860891796375668
Detection rate: 0.8602150537634409, Accuracy: 0.8741258741258742
Precision: 0.8833742322586137, Recall: 0.8741258741258742

out_dynamic.arff with C4.5
True positives: 79.0, True positive rate: 0.8461538461538461
False negatives: 14.0, False negative rate: 0.15384615384615385
False positives: 8.0, False positive rate: 0.1566914805624483
True negatives: 42.0, True negative rate: 0.8433085194375517
Detection rate: 0.8494623655913979, Accuracy: 0.8461538461538461
Precision: 0.8527851458885942, Recall: 0.8461538461538461

out_dynamic.arff with SVM
True positives: 87.0, True positive rate: 0.8321678321678322
False negatives: 6.0, False negative rate: 0.16783216783216784
False positives: 18.0, False positive rate: 0.25668396120009024
True negatives: 32.0, True negative rate: 0.7433160387999098
Detection rate: 0.9354838709677419, Accuracy: 0.8321678321678322
Precision: 0.8333035385666965, Recall: 0.8321678321678322

out_dynamic.arff with Bayes Network
True positives: 88.0, True positive rate: 0.8811188811188811
False negatives: 5.0, False negative rate: 0.11888111888111888
False positives: 12.0, False positive rate: 0.17488232197909617
True negatives: 38.0, True negative rate: 0.8251176780209039
Detection rate: 0.946236559139785, Accuracy: 0.8811188811188811
Precision: 0.8813010245568386, Recall: 0.8811188811188811

out_static.arff with Naive Bayes
True positives: 89.0, True positive rate: 0.8741258741258742
False negatives: 4.0, False negative rate: 0.1258741258741259
False positives: 14.0, False positive rate: 0.1971366268140462
True negatives: 36.0, True negative rate: 0.8028633731859538
```

Detection rate: 0.956989247311828, Accuracy: 0.8741258741258742
Precision: 0.8766379251816144, Recall: 0.8741258741258742

out_static.arff with K-NN
True positives: 90.0, True positive rate: 0.9020979020979021
False negatives: 3.0, False negative rate: 0.0979020979020979
False positives: 11.0, False positive rate: 0.15435596661403114
True negatives: 39.0, True negative rate: 0.8456440333859689
Detection rate: 0.967741935483871, Accuracy: 0.9020979020979021
Precision: 0.9041948150859044, Recall: 0.9020979020979021

out_static.arff with C4.5
True positives: 92.0, True positive rate: 0.9090909090909091
False negatives: 1.0, False negative rate: 0.09090909090909091
False positives: 12.0, False positive rate: 0.1598435972629521
True negatives: 38.0, True negative rate: 0.840156402737048
Detection rate: 0.989247311827957, Accuracy: 0.9090909090909091
Precision: 0.9159942621481082, Recall: 0.9090909090909091

out_static.arff with SVM
True positives: 92.0, True positive rate: 0.8461538461538461
False negatives: 1.0, False negative rate: 0.15384615384615385
False positives: 21.0, False positive rate: 0.27690653432588913
True negatives: 29.0, True negative rate: 0.7230934656741108
Detection rate: 0.989247311827957, Accuracy: 0.8461538461538461
Precision: 0.8674835488994781, Recall: 0.8461538461538461

out_static.arff with Bayes Network
True positives: 92.0, True positive rate: 0.8741258741258742
False negatives: 1.0, False negative rate: 0.1258741258741259
False positives: 17.0, False positive rate: 0.22487856229791714
True negatives: 33.0, True negative rate: 0.7751214377020829
Detection rate: 0.989247311827957, Accuracy: 0.8741258741258742
Precision: 0.8882854867744238, Recall: 0.8741258741258742

D.2 Reduced Feature Sets - CFS

out_combo_cs.arff with Naive Bayes
True positives: 86.0, True positive rate: 0.951048951048951
False negatives: 7.0, False negative rate: 0.04895104895104895
False positives: 0.0, False positive rate: 0.026317768253252126
True negatives: 50.0, True negative rate: 0.973682231746748
Detection rate: 0.9247311827956989, Accuracy: 0.951048951048951
Precision: 0.9570604833762728, Recall: 0.951048951048951

out_combo_cs.arff with K-NN
True positives: 91.0, True positive rate: 0.9440559440559441

```
False negatives: 2.0, False negative rate: 0.055944055944055944
False positives: 6.0, False positive rate: 0.08556132040003009
True negatives: 44.0, True negative rate: 0.9144386795999698
Detection rate: 0.978494623655914, Accuracy: 0.9440559440559441
Precision: 0.9445699974610777, Recall: 0.9440559440559441

out_combo_cs.arff with C4.5
True positives: 86.0, True positive rate: 0.9300699300699301
False negatives: 7.0, False negative rate: 0.06993006993006994
False positives: 3.0, False positive rate: 0.06533874727423114
True negatives: 47.0, True negative rate: 0.9346612527257688
Detection rate: 0.9247311827956989, Accuracy: 0.9300699300699301
Precision: 0.9327530563485621, Recall: 0.9300699300699301

out_combo_cs.arff with SVM
True positives: 83.0, True positive rate: 0.9230769230769231
False negatives: 10.0, False negative rate: 0.07692307692307693
False positives: 1.0, False positive rate: 0.050603804797353186
True negatives: 49.0, True negative rate: 0.9493961952026467
Detection rate: 0.8924731182795699, Accuracy: 0.9230769230769231
Precision: 0.9329949711305644, Recall: 0.9230769230769231

out_combo_cs.arff with Bayes Network
True positives: 85.0, True positive rate: 0.9440559440559441
False negatives: 8.0, False negative rate: 0.055944055944055944
False positives: 0.0, False positive rate: 0.030077449432288145
True negatives: 50.0, True negative rate: 0.9699225505677118
Detection rate: 0.9139784946236559, Accuracy: 0.9440559440559441
Precision: 0.951772365565469, Recall: 0.9440559440559441

out_dynamic_cs.arff with Naive Bayes
True positives: 82.0, True positive rate: 0.916083916083916
False negatives: 11.0, False negative rate: 0.08391608391608392
False positives: 1.0, False positive rate: 0.0543634859763892
True negatives: 49.0, True negative rate: 0.9456365140236107
Detection rate: 0.8817204301075269, Accuracy: 0.916083916083916
Precision: 0.9280618979414161, Recall: 0.916083916083916

out_dynamic_cs.arff with K-NN
True positives: 85.0, True positive rate: 0.9090909090909091
False negatives: 8.0, False negative rate: 0.09090909090909091
False positives: 5.0, False positive rate: 0.09511241446725319
True negatives: 45.0, True negative rate: 0.9048875855327468
Detection rate: 0.9139784946236559, Accuracy: 0.9090909090909091
Precision: 0.9110920526014865, Recall: 0.9090909090909091
```

```
out_dynamic_cs.arff with C4.5
True positives: 80.0, True positive rate: 0.9020979020979021
False negatives: 13.0, False negative rate: 0.09790209790209789
False positives: 1.0, False positive rate: 0.06188284833446123
True negatives: 49.0, True negative rate: 0.9381171516655388
Detection rate: 0.8602150537634409, Accuracy: 0.9020979020979021
Precision: 0.9186572089797895, Recall: 0.9020979020979021

out_dynamic_cs.arff with SVM
True positives: 80.0, True positive rate: 0.9020979020979021
False negatives: 13.0, False negative rate: 0.09790209790209789
False positives: 1.0, False positive rate: 0.06188284833446123
True negatives: 49.0, True negative rate: 0.9381171516655388
Detection rate: 0.8602150537634409, Accuracy: 0.9020979020979021
Precision: 0.9186572089797895, Recall: 0.9020979020979021

out_dynamic_cs.arff with Bayes Network
True positives: 81.0, True positive rate: 0.9090909090909091
False negatives: 12.0, False negative rate: 0.09090909090909091
False positives: 1.0, False positive rate: 0.05812316715542521
True negatives: 49.0, True negative rate: 0.9418768328445748
Detection rate: 0.8709677419354839, Accuracy: 0.9090909090909091
Precision: 0.9232852313620007, Recall: 0.9090909090909091

out_static_cs.arff with Naive Bayes
True positives: 91.0, True positive rate: 0.9440559440559441
False negatives: 2.0, False negative rate: 0.055944055944055944
False positives: 6.0, False positive rate: 0.08556132040003009
True negatives: 44.0, True negative rate: 0.9144386795999698
Detection rate: 0.978494623655914, Accuracy: 0.9440559440559441
Precision: 0.9445699974610777, Recall: 0.9440559440559441

out_static_cs.arff with K-NN
True positives: 91.0, True positive rate: 0.9370629370629371
False negatives: 2.0, False negative rate: 0.06293706293706294
False positives: 7.0, False positive rate: 0.0985683134070231
True negatives: 43.0, True negative rate: 0.9014316865929769
Detection rate: 0.978494623655914, Accuracy: 0.9370629370629371
Precision: 0.938006438006438, Recall: 0.9370629370629371

out_static_cs.arff with C4.5
True positives: 91.0, True positive rate: 0.9230769230769231
False negatives: 2.0, False negative rate: 0.07692307692307693
False positives: 9.0, False positive rate: 0.12458229942100908
True negatives: 41.0, True negative rate: 0.8754177005789908
Detection rate: 0.978494623655914, Accuracy: 0.9230769230769231
```

```
Precision: 0.9252057245080502, Recall: 0.9230769230769231
```

```
out_static_cs.arff with SVM
True positives: 91.0, True positive rate: 0.916083916083916
False negatives: 2.0, False negative rate: 0.08391608391608392
False positives: 10.0, False positive rate: 0.1375892924280021
True negatives: 40.0, True negative rate: 0.8624107075719979
Detection rate: 0.978494623655914, Accuracy: 0.916083916083916
Precision: 0.9189589288599189, Recall: 0.916083916083916
```

```
out_static_cs.arff with Bayes Network
True positives: 91.0, True positive rate: 0.916083916083916
False negatives: 2.0, False negative rate: 0.08391608391608392
False positives: 10.0, False positive rate: 0.1375892924280021
True negatives: 40.0, True negative rate: 0.8624107075719979
Detection rate: 0.978494623655914, Accuracy: 0.916083916083916
Precision: 0.9189589288599189, Recall: 0.916083916083916
```

D.3 New Combined Feature Set - CFS

```
out_combined_merged with Naive Bayes
True positives: 85.0, True positive rate: 0.9440559440559441
False negatives: 8.0, False negative rate: 0.055944055944055944
False positives: 0.0, False positive rate: 0.030077449432288145
True negatives: 50.0, True negative rate: 0.9699225505677118
Detection rate: 0.9139784946236559, Accuracy: 0.9440559440559441
Precision: 0.951772365565469, Recall: 0.9440559440559441
```

```
out_combined_merged with K-NN
True positives: 90.0, True positive rate: 0.9230769230769231
False negatives: 3.0, False negative rate: 0.07692307692307693
False positives: 8.0, False positive rate: 0.11533498759305211
True negatives: 42.0, True negative rate: 0.8846650124069478
Detection rate: 0.967741935483871, Accuracy: 0.9230769230769231
Precision: 0.9236002093144952, Recall: 0.9230769230769231
```

```
out_combined_merged with C4.5
True positives: 88.0, True positive rate: 0.9020979020979021
False negatives: 5.0, False negative rate: 0.0979020979020979
False positives: 9.0, False positive rate: 0.13586134295811714
True negatives: 41.0, True negative rate: 0.8641386570418828
Detection rate: 0.946236559139785, Accuracy: 0.9020979020979021
Precision: 0.9016528070763838, Recall: 0.9020979020979021
```

```
out_combined_merged with SVM
True positives: 87.0, True positive rate: 0.9230769230769231
False negatives: 6.0, False negative rate: 0.07692307692307693
False positives: 5.0, False positive rate: 0.08759305210918115
True negatives: 45.0, True negative rate: 0.9124069478908188
Detection rate: 0.9354838709677419, Accuracy: 0.9230769230769231
Precision: 0.9235195750541019, Recall: 0.9230769230769231
```

```
out_combined_merged with Bayes Network
True positives: 86.0, True positive rate: 0.951048951048951
False negatives: 7.0, False negative rate: 0.04895104895104895
False positives: 0.0, False positive rate: 0.026317768253252126
True negatives: 50.0, True negative rate: 0.973682231746748
Detection rate: 0.9247311827956989, Accuracy: 0.951048951048951
Precision: 0.9570604833762728, Recall: 0.951048951048951
```

D.4 Reduced Feature Sets - G*e*FS$_{\text{CFS}}$

```
out_combo_red.arff with Naive Bayes
True positives: 85.0, True positive rate: 0.9440559440559441
False negatives: 8.0, False negative rate: 0.055944055944055944
False positives: 0.0, False positive rate: 0.030077449432288145
True negatives: 50.0, True negative rate: 0.9699225505677118
Detection rate: 0.9139784946236559, Accuracy: 0.9440559440559441
Precision: 0.951772365565469, Recall: 0.9440559440559441
```

```
out_combo_red.arff with K-NN
True positives: 91.0, True positive rate: 0.951048951048951
False negatives: 2.0, False negative rate: 0.04895104895104895
False positives: 5.0, False positive rate: 0.07255432739303708
True negatives: 45.0, True negative rate: 0.927445672606963
Detection rate: 0.978494623655914, Accuracy: 0.951048951048951
Precision: 0.9512488840946288, Recall: 0.951048951048951
```

```
out_combo_red.arff with C4.5
True positives: 85.0, True positive rate: 0.9090909090909091
False negatives: 8.0, False negative rate: 0.09090909090909091
False positives: 5.0, False positive rate: 0.09511241446725319
True negatives: 45.0, True negative rate: 0.9048875855327468
Detection rate: 0.9139784946236559, Accuracy: 0.9090909090909091
Precision: 0.9110920526014865, Recall: 0.9090909090909091
```

```
out_combo_red.arff with SVM
True positives: 85.0, True positive rate: 0.9370629370629371
False negatives: 8.0, False negative rate: 0.06293706293706294
False positives: 1.0, False positive rate: 0.043084442439281154
True negatives: 49.0, True negative rate: 0.9569155575607189
Detection rate: 0.9139784946236559, Accuracy: 0.9370629370629371
Precision: 0.9433640614791164, Recall: 0.9370629370629371
```

```
out_combo_red.arff with Bayes Network
True positives: 86.0, True positive rate: 0.951048951048951
False negatives: 7.0, False negative rate: 0.04895104895104895
False positives: 0.0, False positive rate: 0.026317768253252126
True negatives: 50.0, True negative rate: 0.973682231746748
Detection rate: 0.9247311827956989, Accuracy: 0.951048951048951
Precision: 0.9570604833762728, Recall: 0.951048951048951

out_dynamic_red.arff with Naive Bayes
True positives: 82.0, True positive rate: 0.916083916083916
False negatives: 11.0, False negative rate: 0.08391608391608392
False positives: 1.0, False positive rate: 0.0543634859763892
True negatives: 49.0, True negative rate: 0.9456365140236107
Detection rate: 0.8817204301075269, Accuracy: 0.916083916083916
Precision: 0.9280618979414161, Recall: 0.916083916083916

out_dynamic_red.arff with K-NN
True positives: 82.0, True positive rate: 0.8741258741258742
False negatives: 11.0, False negative rate: 0.1258741258741259
False positives: 7.0, False positive rate: 0.13240544401834728
True negatives: 43.0, True negative rate: 0.8675945559816527
Detection rate: 0.8817204301075269, Accuracy: 0.8741258741258742
Precision: 0.8776238326800125, Recall: 0.8741258741258742

out_dynamic_red.arff with C4.5
True positives: 81.0, True positive rate: 0.8741258741258742
False negatives: 12.0, False negative rate: 0.1258741258741259
False positives: 6.0, False positive rate: 0.12315813219039023
True negatives: 44.0, True negative rate: 0.8768418678096097
Detection rate: 0.8709677419354839, Accuracy: 0.8741258741258742
Precision: 0.8802232250508112, Recall: 0.8741258741258742

out_dynamic_red.arff with SVM
True positives: 81.0, True positive rate: 0.8881118881118881
False negatives: 12.0, False negative rate: 0.11188811188811189
False positives: 4.0, False positive rate: 0.09714414617640424
True negatives: 46.0, True negative rate: 0.9028558538235959
Detection rate: 0.8709677419354839, Accuracy: 0.8881118881118881
Precision: 0.8970538589199848, Recall: 0.8881118881118881

out_dynamic_red.arff with Bayes Network
True positives: 82.0, True positive rate: 0.916083916083916
False negatives: 11.0, False negative rate: 0.08391608391608392
False positives: 1.0, False positive rate: 0.0543634859763892
True negatives: 49.0, True negative rate: 0.9456365140236107
Detection rate: 0.8817204301075269, Accuracy: 0.916083916083916
```

```
Precision: 0.9280618979414161, Recall: 0.916083916083916
```

```
out_static_red.arff with Naive Bayes
True positives: 90.0, True positive rate: 0.916083916083916
False negatives: 3.0, False negative rate: 0.08391608391608392
False positives: 9.0, False positive rate: 0.12834198060004512
True negatives: 41.0, True negative rate: 0.8716580193999548
Detection rate: 0.967741935483871, Accuracy: 0.916083916083916
Precision: 0.9170375079465988, Recall: 0.916083916083916
```

```
out_static_red.arff with K-NN
True positives: 90.0, True positive rate: 0.9230769230769231
False negatives: 3.0, False negative rate: 0.07692307692307693
False positives: 8.0, False positive rate: 0.11533498759305211
True negatives: 42.0, True negative rate: 0.8846650124069478
Detection rate: 0.967741935483871, Accuracy: 0.9230769230769231
Precision: 0.9236002093144952, Recall: 0.9230769230769231
```

```
out_static_red.arff with C4.5
True positives: 92.0, True positive rate: 0.9090909090909091
False negatives: 1.0, False negative rate: 0.09090909090909091
False positives: 12.0, False positive rate: 0.1598435972629521
True negatives: 38.0, True negative rate: 0.840156402737048
Detection rate: 0.989247311827957, Accuracy: 0.9090909090909091
Precision: 0.9159942621481082, Recall: 0.9090909090909091
```

```
out_static_red.arff with SVM
True positives: 90.0, True positive rate: 0.9090909090909091
False negatives: 3.0, False negative rate: 0.09090909090909091
False positives: 10.0, False positive rate: 0.14134897360703813
True negatives: 40.0, True negative rate: 0.858651026392962
Detection rate: 0.967741935483871, Accuracy: 0.9090909090909091
Precision: 0.9105708245243128, Recall: 0.9090909090909091
```

```
out_static_red.arff with Bayes Network
True positives: 90.0, True positive rate: 0.9090909090909091
False negatives: 3.0, False negative rate: 0.09090909090909091
False positives: 10.0, False positive rate: 0.14134897360703813
True negatives: 40.0, True negative rate: 0.858651026392962
Detection rate: 0.967741935483871, Accuracy: 0.9090909090909091
Precision: 0.9105708245243128, Recall: 0.9090909090909091
```

D.5 New Combined Feature Set - GeFS$_{CFS}$

```
out_combined_merged with Naive Bayes
True positives: 87.0, True positive rate: 0.9230769230769231
False negatives: 6.0, False negative rate: 0.07692307692307693
False positives: 5.0, False positive rate: 0.08759305210918115
True negatives: 45.0, True negative rate: 0.9124069478908188
Detection rate: 0.9354838709677419, Accuracy: 0.9230769230769231
Precision: 0.9235195750541019, Recall: 0.9230769230769231
```

```
out_combined_merged with K-NN
True positives: 90.0, True positive rate: 0.9230769230769231
False negatives: 3.0, False negative rate: 0.07692307692307693
False positives: 8.0, False positive rate: 0.11533498759305211
True negatives: 42.0, True negative rate: 0.8846650124069478
Detection rate: 0.967741935483871, Accuracy: 0.9230769230769231
Precision: 0.9236002093144952, Recall: 0.9230769230769231
```

```
out_combined_merged with C4.5
True positives: 89.0, True positive rate: 0.9370629370629371
False negatives: 4.0, False negative rate: 0.06293706293706294
False positives: 5.0, False positive rate: 0.08007368975110911
True negatives: 45.0, True negative rate: 0.9199263102488909
Detection rate: 0.956989247311828, Accuracy: 0.9370629370629371
Precision: 0.936864047806297, Recall: 0.9370629370629371
```

```
out_combined_merged with SVM
True positives: 90.0, True positive rate: 0.916083916083916
False negatives: 3.0, False negative rate: 0.08391608391608392
False positives: 9.0, False positive rate: 0.12834198060004512
True negatives: 41.0, True negative rate: 0.8716580193999548
Detection rate: 0.967741935483871, Accuracy: 0.916083916083916
Precision: 0.9170375079465988, Recall: 0.916083916083916
```

```
out_combined_merged with Bayes Network
True positives: 90.0, True positive rate: 0.9440559440559441
False negatives: 3.0, False negative rate: 0.055944055944055944
False positives: 5.0, False positive rate: 0.0763140085720731
True negatives: 45.0, True negative rate: 0.923685991427927
Detection rate: 0.967741935483871, Accuracy: 0.9440559440559441
Precision: 0.9439179241810821, Recall: 0.9440559440559441
```

D.6 Complete Feature Sets - Unsupervised Discretization

```
out_combo_d.arff with Naive Bayes
True positives: 92.0, True positive rate: 0.7272727272727273
False negatives: 1.0, False negative rate: 0.2727272727272727
False positives: 38.0, False positive rate: 0.4980254154447703
True negatives: 12.0, True negative rate: 0.5019745845552297
Detection rate: 0.989247311827957, Accuracy: 0.7272727272727273
Precision: 0.7830016137708445, Recall: 0.7272727272727273
```

```
out_combo_d.arff with K-NN
True positives: 89.0, True positive rate: 0.8951048951048951
False negatives: 4.0, False negative rate: 0.1048951048951049
False positives: 11.0, False positive rate: 0.15811564779306717
True negatives: 39.0, True negative rate: 0.8418843522069329
Detection rate: 0.956989247311828, Accuracy: 0.8951048951048951
Precision: 0.8959359245405756, Recall: 0.8951048951048951

out_combo_d.arff with C4.5
True positives: 82.0, True positive rate: 0.8391608391608392
False negatives: 11.0, False negative rate: 0.16083916083916083
False positives: 12.0, False positive rate: 0.19744040905331228
True negatives: 38.0, True negative rate: 0.8025595909466878
Detection rate: 0.8817204301075269, Accuracy: 0.8391608391608392
Precision: 0.8384837047451029, Recall: 0.8391608391608392

out_combo_d.arff with SVM
True positives: 87.0, True positive rate: 0.8391608391608392
False negatives: 6.0, False negative rate: 0.16083916083916083
False positives: 17.0, False positive rate: 0.2436769681930972
True negatives: 33.0, True negative rate: 0.7563230318069027
Detection rate: 0.9354838709677419, Accuracy: 0.8391608391608392
Precision: 0.8399004841312533, Recall: 0.8391608391608392

out_combo_d.arff with Bayes Network
True positives: 91.0, True positive rate: 0.8671328671328671
False negatives: 2.0, False negative rate: 0.13286713286713286
False positives: 17.0, False positive rate: 0.22863824347695316
True negatives: 33.0, True negative rate: 0.771361756523047
Detection rate: 0.978494623655914, Accuracy: 0.8671328671328671
Precision: 0.8776501276501276, Recall: 0.8671328671328671

out_dynamic_d.arff with Naive Bayes
True positives: 93.0, True positive rate: 0.6573426573426573
False negatives: 0.0, False negative rate: 0.34265734265734266
False positives: 49.0, False positive rate: 0.6373426573426574
True negatives: 1.0, True negative rate: 0.3626573426573427
Detection rate: 1.0, Accuracy: 0.6573426573426573
Precision: 0.7755835713582192, Recall: 0.6573426573426573

out_dynamic_d.arff with K-NN
True positives: 80.0, True positive rate: 0.8741258741258742
False negatives: 13.0, False negative rate: 0.1258741258741259
False positives: 5.0, False positive rate: 0.11391082036243326
True negatives: 45.0, True negative rate: 0.8860891796375668
Detection rate: 0.8602150537634409, Accuracy: 0.8741258741258742
```

```
Precision: 0.8833742322586137, Recall: 0.8741258741258742

out_dynamic_d.arff with C4.5
True positives: 81.0, True positive rate: 0.8391608391608392
False negatives: 12.0, False negative rate: 0.16083916083916083
False positives: 11.0, False positive rate: 0.1881930972253553
True negatives: 39.0, True negative rate: 0.8118069027746447
Detection rate: 0.8709677419354839, Accuracy: 0.8391608391608392
Precision: 0.8399701321696207, Recall: 0.8391608391608392

out_dynamic_d.arff with SVM
True positives: 86.0, True positive rate: 0.8531468531468531
False negatives: 7.0, False negative rate: 0.14685314685314685
False positives: 14.0, False positive rate: 0.20841567035115424
True negatives: 36.0, True negative rate: 0.7915843296488457
Detection rate: 0.9247311827956989, Accuracy: 0.8531468531468531
Precision: 0.8520312245893642, Recall: 0.8531468531468531

out_dynamic_d.arff with Bayes Network
True positives: 88.0, True positive rate: 0.8811188811188811
False negatives: 5.0, False negative rate: 0.11888111888111888
False positives: 12.0, False positive rate: 0.17488232197909617
True negatives: 38.0, True negative rate: 0.8251176780209039
Detection rate: 0.946236559139785, Accuracy: 0.8811188811188811
Precision: 0.8813010245568386, Recall: 0.8811188811188811

out_static_d.arff with Naive Bayes
True positives: 92.0, True positive rate: 0.8461538461538461
False negatives: 1.0, False negative rate: 0.15384615384615385
False positives: 21.0, False positive rate: 0.27690653432588913
True negatives: 29.0, True negative rate: 0.7230934656741108
Detection rate: 0.989247311827957, Accuracy: 0.8461538461538461
Precision: 0.8674835488994781, Recall: 0.8461538461538461

out_static_d.arff with K-NN
True positives: 90.0, True positive rate: 0.9020979020979021
False negatives: 3.0, False negative rate: 0.0979020979020979
False positives: 11.0, False positive rate: 0.15435596661403114
True negatives: 39.0, True negative rate: 0.8456440333859689
Detection rate: 0.967741935483871, Accuracy: 0.9020979020979021
Precision: 0.9041948150859044, Recall: 0.9020979020979021

out_static_d.arff with C4.5
True positives: 92.0, True positive rate: 0.9370629370629371
False negatives: 1.0, False negative rate: 0.06293706293706294
False positives: 8.0, False positive rate: 0.10781562523498008
```

True negatives: 42.0, True negative rate: 0.8921843747650198
Detection rate: 0.989247311827957, Accuracy: 0.9370629370629371
Precision: 0.9398406244917872, Recall: 0.9370629370629371

out_static_d.arff with SVM
True positives: 82.0, True positive rate: 0.8461538461538461
False negatives: 11.0, False negative rate: 0.15384615384615385
False positives: 11.0, False positive rate: 0.18443341604631927
True negatives: 39.0, True negative rate: 0.8155665839536808
Detection rate: 0.8817204301075269, Accuracy: 0.8461538461538461
Precision: 0.8461538461538461, Recall: 0.8461538461538461

out_static_d.arff with Bayes Network
True positives: 92.0, True positive rate: 0.8741258741258742
False negatives: 1.0, False negative rate: 0.1258741258741259
False positives: 17.0, False positive rate: 0.22487856229791714
True negatives: 33.0, True negative rate: 0.7751214377020829
Detection rate: 0.989247311827957, Accuracy: 0.8741258741258742
Precision: 0.8882854867744238, Recall: 0.8741258741258742

D.7 Complete Feature Sets - Supervised Discretization

out_combo_d_sv.arff with Naive Bayes
True positives: 78.0, True positive rate: 0.8881118881118881
False negatives: 15.0, False negative rate: 0.11188811188811189
False positives: 1.0, False positive rate: 0.06940221069253327
True negatives: 49.0, True negative rate: 0.9305977893074666
Detection rate: 0.8387096774193549, Accuracy: 0.8881118881118881
Precision: 0.9098184252456405, Recall: 0.8881118881118881

out_combo_d_sv.arff with K-NN
True positives: 88.0, True positive rate: 0.916083916083916
False negatives: 5.0, False negative rate: 0.08391608391608392
False positives: 7.0, False positive rate: 0.10984735694413114
True negatives: 43.0, True negative rate: 0.8901526430558689
Detection rate: 0.946236559139785, Accuracy: 0.916083916083916
Precision: 0.9156575880260092, Recall: 0.916083916083916

out_combo_d_sv.arff with C4.5
True positives: 83.0, True positive rate: 0.8811188811188811
False negatives: 10.0, False negative rate: 0.11888111888111888
False positives: 7.0, False positive rate: 0.12864576283931123
True negatives: 43.0, True negative rate: 0.8713542371606888
Detection rate: 0.8924731182795699, Accuracy: 0.8811188811188811
Precision: 0.8834454853322778, Recall: 0.8811188811188811

out_combo_d_sv.arff with SVM

```
True positives: 92.0, True positive rate: 0.7762237762237763
False negatives: 1.0, False negative rate: 0.22377622377622378
False positives: 31.0, False positive rate: 0.4069764643958192
True negatives: 19.0, True negative rate: 0.5930235356041809
Detection rate: 0.989247311827957, Accuracy: 0.7762237762237763
Precision: 0.8186082210472454, Recall: 0.7762237762237763

out_combo_d_sv.arff with Bayes Network
True positives: 78.0, True positive rate: 0.8881118881118881
False negatives: 15.0, False negative rate: 0.11188811188811189
False positives: 1.0, False positive rate: 0.06940221069253327
True negatives: 49.0, True negative rate: 0.9305977893074666
Detection rate: 0.8387096774193549, Accuracy: 0.8881118881118881
Precision: 0.9098184252456405, Recall: 0.8881118881118881

out_dynamic_d_sv.arff with Naive Bayes
True positives: 76.0, True positive rate: 0.8741258741258742
False negatives: 17.0, False negative rate: 0.1258741258741259
False positives: 1.0, False positive rate: 0.0769215730506053
True negatives: 49.0, True negative rate: 0.9230784269493947
Detection rate: 0.8172043010752689, Accuracy: 0.8741258741258742
Precision: 0.9014924469469924, Recall: 0.8741258741258742

out_dynamic_d_sv.arff with K-NN
True positives: 82.0, True positive rate: 0.8601398601398601
False negatives: 11.0, False negative rate: 0.13986013986013987
False positives: 9.0, False positive rate: 0.15841943003233325
True negatives: 41.0, True negative rate: 0.8415805699676667
Detection rate: 0.8817204301075269, Accuracy: 0.8601398601398601
Precision: 0.8617152078690541, Recall: 0.8601398601398601

out_dynamic_d_sv.arff with C4.5
True positives: 78.0, True positive rate: 0.8881118881118881
False negatives: 15.0, False negative rate: 0.11188811188811189
False positives: 1.0, False positive rate: 0.06940221069253327
True negatives: 49.0, True negative rate: 0.9305977893074666
Detection rate: 0.8387096774193549, Accuracy: 0.8881118881118881
Precision: 0.9098184252456405, Recall: 0.8881118881118881

out_dynamic_d_sv.arff with SVM
True positives: 93.0, True positive rate: 0.6503496503496503
False negatives: 0.0, False negative rate: 0.34965034965034963
False positives: 50.0, False positive rate: 0.6503496503496503
True negatives: 0.0, True negative rate: 0.34965034965034963
Detection rate: 1.0, Accuracy: 0.6503496503496503
Precision: 0.42295466770991247, Recall: 0.6503496503496503
```

```
out_dynamic_d_sv.arff with Bayes Network
True positives: 76.0, True positive rate: 0.8741258741258742
False negatives: 17.0, False negative rate: 0.1258741258741259
False positives: 1.0, False positive rate: 0.0769215730506053
True negatives: 49.0, True negative rate: 0.9230784269493947
Detection rate: 0.8172043010752689, Accuracy: 0.8741258741258742
Precision: 0.9014924469469924, Recall: 0.8741258741258742

out_static_d_sv.arff with Naive Bayes
True positives: 90.0, True positive rate: 0.8811188811188811
False negatives: 3.0, False negative rate: 0.11888111888111888
False positives: 14.0, False positive rate: 0.19337694563501018
True negatives: 36.0, True negative rate: 0.8066230543649898
Detection rate: 0.967741935483871, Accuracy: 0.8811188811188811
Precision: 0.8855567509413663, Recall: 0.8811188811188811

out_static_d_sv.arff with K-NN
True positives: 91.0, True positive rate: 0.8951048951048951
False negatives: 2.0, False negative rate: 0.1048951048951049
False positives: 13.0, False positive rate: 0.17661027144898112
True negatives: 37.0, True negative rate: 0.8233897285510188
Detection rate: 0.978494623655914, Accuracy: 0.8951048951048951
Precision: 0.9007755065447373, Recall: 0.8951048951048951

out_static_d_sv.arff with C4.5
True positives: 92.0, True positive rate: 0.9370629370629371
False negatives: 1.0, False negative rate: 0.06293706293706294
False positives: 8.0, False positive rate: 0.10781562523498008
True negatives: 42.0, True negative rate: 0.8921843747650198
Detection rate: 0.989247311827957, Accuracy: 0.9370629370629371
Precision: 0.9398406244917872, Recall: 0.9370629370629371

out_static_d_sv.arff with SVM
True positives: 92.0, True positive rate: 0.8461538461538461
False negatives: 1.0, False negative rate: 0.15384615384615385
False positives: 21.0, False positive rate: 0.27690653432588913
True negatives: 29.0, True negative rate: 0.7230934656741108
Detection rate: 0.989247311827957, Accuracy: 0.8461538461538461
Precision: 0.8674835488994781, Recall: 0.8461538461538461

out_static_d_sv.arff with Bayes Network
True positives: 90.0, True positive rate: 0.8811188811188811
False negatives: 3.0, False negative rate: 0.11888111888111888
False positives: 14.0, False positive rate: 0.19337694563501018
True negatives: 36.0, True negative rate: 0.8066230543649898
```

```
Detection rate: 0.967741935483871, Accuracy: 0.8811188811188811
Precision: 0.8855567509413663, Recall: 0.8811188811188811
```

E Source Code

E.1 deLink Integration

```
from arff import arffread,arffwrite # using arff module!
import commands                     # commands module
import re                           # regex module
import sys                          # for input commands
import pefile                       # static PE parser
import os                           # io interface
import submit_to_anubis             # anubis sanbox submitter
import urllib2                      # web interface
import urllib                       # web interface
import time                         # time interface

# delink feature extractor function
def delink():
    # Gets all neccesarily information:
    ddimage = raw_input("Image file (whole path)\n")
    innfilearff = raw_input("Corresponding ARFF input file (whole path)\n")
    arffin = open(innfilearff)
    arffout = raw_input("Output filename\n")
    machinenr = raw_input("Machine number\n")
    medianr = raw_input("Media number\n")

    fout = open(arffout, 'w') # Opens input arff file

    # parses the input ARFF file:
    print "Starting Parsing ARFF file..."
    (name, sparse, alist, m) = arffread(arffin)
    print "Done Parsing!"

    # Include all new attributes for the ARFF file:
    newent = ['entropy',1,[]] # indexing attribute entropy
    machine = ['machine',1,[]] # indexing attribute machine metadata (numeric)
    media = ['media',1,[]] # indexing attribute media metadata (e.g., disk = 1)
    ipstr = ['IPs',0,[]] # indexing attribute IPs
    mailstr = ['mails',0,[]] # indexing attribute mails
    urlstr = ['urls',0,[]] # indexing attribute URLs
    alist.append(machine) # adding machine attribute to alist
    alist.append(media) # adding media attribute to alist
    alist.append(newent) # adding ent attribute to alist
    alist.append(ipstr) # adding ip attribute to alist
    alist.append(mailstr) # adding mail attribute to alist
    alist.append(urlstr) # adding url attribute to alist

    # goes through all file objects parsed from arff file and adds features:
    for obj in m:
        iptotal = "" # resetting for current file object's ip string
        mailtotal = "" # resetting for current file object's mail string
        urltotal = "" # resetting for current file object's url string
        obj.append(machinenr) # machine metadata add
        obj.append(medianr) # media metadata add
        cmdent = "icat "+ddimage+" "+str(obj[15]) + " | ent" # command to calculate entropy
        entoutput = commands.getoutput(cmdent) # gets entropy
        ent = re.findall('Entropy = ([0-9].[0-9]+)',entoutput) # extracts entropy value
        print "Entropy for: ", obj[0]," - ", ent[0] # prints status and entropy
        obj.append(ent[0]) # adds the entropy to object

        # opening current file object's content from image file, using icat
        print "IP, mails and URLs is added to: ",obj[0] # adding status
        cmdstr = "icat "+ddimage+" "+str(obj[15]) # command to get file content from image
        using icat
        catinput = commands.getoutput(cmdstr) # executes command and stores output for
        searching

        # find all ips, based on following regular expression:
        ips = re.findall('(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})\.(?:[\d]{1,3})',
             catinput)
        if len(ips) > 0: # if any IP found
            for ip in ips: # go through all found
                iptotal += '"' + ip + '" ' # add to IP to string, seperated by quotes
            obj.append(iptotal) # appending ips to list
        else: # if no ip
            obj.append('?') # add ? (empty)

        # find all emails, based on following regular expression:
        emails = re.findall(
        '[a-zA-Z0-9]+[\.\_\-]*[a-zA-Z0-9]*@[a-zA-Z0-9]+[\.\-\_]*[a-zA-Z0-9]*[\.][a-zA-Z]+',
              catinput)
        if len(emails) > 0: # if any email found
            for mail in emails: # go through all found
                mailtotal += '"' + mail + '" ' # add mail to string
            obj.append(mailtotal) # appending emails to list
        else: # if no email
            obj.append('?') # add ? (empty)

        #find all URLS, based on following regular expression:
        urls = re.findall(
        'http[s]?://(?:[a-zA-Z]|[0-9]|[!*\(\),]|[\.]|(?:%[0-9a-fA-F][0-9a-fA-F]))+',
             catinput)
        if len(urls) > 0: # if any email found
            for url in urls: # go through all found
                urltotal += '"' + url + '" ' # add mail to string
            obj.append(urltotal) #appending urls to list
        else: # if no email
            obj.append('?') # add ? (empty)

    #writes the new lists with attributes and values to "fout"-file:
    arffwrite(fout,alist,m, name)
    #close the used files:
    fout.close()
    arffin.close()

# malware analysis function
def malware_analysis
    # path to malware samples
    directory_path = "Full path to malware samples"
    # load names into list
    file_list = os.listdir(directory_path)
    # iterating through list and generating static reports
    for name in file_list:
        print 'current malware '+name
        pe = pefile.PE(directory_path + name)
        # check if patch exists
        d = './static_data/'
        if not os.path.exists(d):
            os.makedirs(d)
        # store report as txt file
        report_name = d+name+'.txt'
        report = open(report_name,'w')
        report.write(pe.dump_info())

    # path to xml-file
    xml_path = "http://anubis.iseclab.org/index.php?action=result&task_id="
    # add task ids to list
    id_list = []

    # iterating through list and upload files
    for name in file_list:
        # upload files to anubis
        print 'current file '+name
        fname = directory_path+name
        task_id = submit_to_anubis.submit('FILE',fname)
        id_list.append(task_id)
        print 'get your report with this id '+task_id

    counter = 0
    # iterate through id list
    for task_id in id_list:
        print 'processing current task '+task_id
        done = False

        # loop until xml-file is ready -- may take long time depending on queue
        while not done:
            # download xml or html report
            url = xml_path+task_id+'&format=xml'
            user_agent = 'Internet Explorer'
            values = {'name':'User','location':'No','language':'Python'}
            headers = {'User-Agent':user_agent}

            print 'fetching file from '+url
            data = urllib.urlencode(values)
            req = urllib2.Request(url,data,headers)
            xml_url = urllib2.urlopen(req).geturl()
            xml_file = urllib2.urlopen(xml_url).read()

            # parse file
            html_file = False
            if xml_file.find(’<html’) != −1:
                print ’no xml file is available’
                html_file = True

            # if anubis is ready
            if not html_file:
                # check if patch exists
                d = ’./dynamic_data/’
                if not os.path.exists(d):
                    os.makedirs(d)
                # store file
                file_name = d + file_list.index(counter) + ’.xml’
                localf = open(file_name,’w’)
                localf.write(xml_file)
                localf.close()
                done = True
                counter += 1
                print ’xml retrieved and stored as ’+task_id+’.xml’
            else:
                # sleep for 5 second
                print ’sleeping for 5 seconds ... zzz’
                time.sleep(5)

    print ’done retrieving static and dynamic reports!’

    #must call the jar−file for ARFF file creation
    os.system("java −jar feature_extractor.jar")

# main function
def main(argv):
    for arg in sys.argv:
        if arg == ’−d’:
            # deLink feature extraction
            delink()
        elif arg == ’−f’:
            # static and dynamic feature extraction
            malware_analysis()

if __name__ == "__main__":
    main(sys.argv)
```

E.2 Feature Extractor

Main

```
package feature.extractor;

//main class for feature extractor component
//extracts static and dynamic features
//builds static, dynamic and a combined feature set
//generates arff-files reflecting the feature sets
public class Main
{
        //main method
        public static void main(String[] args)
        {
                try
                {
                        FeatureExtractor extractor =
                                new FeatureExtractor();

                        //EXTRACT FEATURES!
                        extractor.extract_dynamic_features();
                        System.out.println();
                        extractor.extract_static_features();

                        //COMBINE FEATURES!
                        extractor.combine_features();

                        //EXTRACT LABELS!
                        extractor.extract_labels();

                        //GENERATE ARFF!
                        System.out.println();
                        extractor.generate_arff();


                } catch (Exception e)
                {
                        e.printStackTrace();
                }
        }

}
```

Parser

```
package feature.extractor;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
//DOM libraries for XML parsing
import javax.xml.parsers.*;
import org.w3c.dom.*;

//class Parser is the controlling entity of the XML and text parser
//it loads the dynamic and static reports and initiates parsing
public class Parser
{
        //directories
        private static final String DYNAMIC_PATH = ".\\dynamic_data\\";
        private static final String STATIC_PATH = ".\\static_data\\";

        //samples ready for further processing
        private ArrayList<XML> xml_samples;
        private ArrayList<PE> pe_samples;

        //xml and txt files
        private ArrayList<File> pe_files;
        private ArrayList<File> xml_files;

        public Parser()
        {
                xml_samples = new ArrayList<XML>();
                pe_samples = new ArrayList<PE>();

                pe_files = new ArrayList<File>();
                xml_files = new ArrayList<File>();
        }
```

```
//return xml samples
public ArrayList<XML> get_xml_samples()
{
        return xml_samples;
}

//return pe samples
public ArrayList<PE> get_pe_samples()
{
        return pe_samples;
}

//loads dynamic reports
public void open_dynamic_reports()
{
        //open static_data -> loop through each file name
        File data_folder = new File(DYNAMIC_PATH);
        File[] contents = data_folder.listFiles();

        for (int i = 0; i < contents.length; i++)
        {
              if (contents[i].isFile())
              {
                xml_files.add(contents[i]);
              }
        }
}

//initiate xml parser
public void parse_xml() throws Exception
{
        //iterate through loaded files
        for(int i = 0; i < xml_files.size(); i++)
        {
                XML xml = new XML();
                String id = xml_files.get(i).getName();
                id = id.substring(0, id.length()-4);

                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance
                     ();
                DocumentBuilder builder = factory.newDocumentBuilder();
                Document doc = builder.parse( xml_files.get(i) );

                xml.analyze_document(doc,id);

                xml_samples.add(xml);
        }
}

// loads static reports
public void open_static_reports()
{
        //open static_data -> loop through each file name
        File data_folder = new File(STATIC_PATH);
        File[] contents = data_folder.listFiles();

        for (int i = 0; i < contents.length; i++)
        {
                if (contents[i].isFile())
                {
                        pe_files.add(contents[i]);
                }
        }
}

//intiate txt parser
public void parse_pe() throws IOException
{
        //iterate through loaded files
        for(int i = 0; i < pe_files.size(); i++)
        {
                File file = pe_files.get(i);
                String id = file.getName();
                id = id.substring(0, id.length()-4);

                int len;
                char[] chr = new char[4096];
                final StringBuffer buffer = new StringBuffer();
                final FileReader reader = new FileReader(file);
```

```
                        try
                        {
                                while ((len = reader.read(chr)) > 0)
                                {
                                        buffer.append(chr, 0, len);
                                }
                        }
                        finally
                        {
                                reader.close();
                        }

                        PE pe = new PE();
                        pe.add_dll(buffer.toString(),id);
                        pe_samples.add(pe);
                }
        }
}
```

PE

```
package feature.extractor;

import java.util.ArrayList;

//class PE is the text-parser for static
//created by the PE parser pefile
public class PE
{
        private String sample_id;
        private ArrayList<DLL> dll;

        public PE()
        {
                dll = new ArrayList<DLL>();
        }

        //method parsers static text files and look
        //for DLL dependencies in the report
        public void add_dll(String text, String id)
        {
                sample_id = id;

                //trigger on '.dll' string
                String delimiter = ".dll";

                //iterating through static report
                for(int i = 0; i < text.length()-5;i++)
                {
                        String dll_sub = text.substring(i, i+4);
                        //if found a used DLL
                        if(dll_sub.compareTo(delimiter)==0)
                        {
                                //need to parse backward to find DLL name
                                int back_pos = i;
                                boolean found_name = false;

                                while(!found_name)
                                {
                                        String n = text.substring(back_pos-1,back_pos);

                                        if(n.compareTo("\n") == 0)
                                        {
                                                found_name = true;
                                        }
                                        else back_pos--;
                                }

                                String name = text.substring(back_pos, i);

                                //if function name is found
                                if(found_name)
                                {
                                        //need to add "unknown" function names as well
                                        String unknown = " Ordinal";
                                        String sub = text.substring(i+4,i+12);

                                        //extra parsing statement variable
                                        String d = text.substring(i, i+5);
                                        if(unknown.compareTo(sub) == 0)
                                        {
                                                String function = "unknown";

                                                // add to ArrayList
                                                dll.add(new DLL(name,function));
                                        }
                                        else if(d.compareTo(".dll.")==0)
                                        {
                                                //need to parse forward to find function
                                                     name
                                                int for_pos = i;
                                                boolean found_function = false;

                                                while(!found_function)
                                                {
                                                        String n = text.substring(for_pos,
                                                             for_pos+1);

                                                        if(n.compareTo(" ") == 0)
                                                        {
                                                                found_function = true;
                                                        }
                                                        else for_pos++;
                                                }

                                                String function = text.substring(i+5,
                                                     for_pos);
                                                // add to ArrayList
                                                dll.add(new DLL(name,function));
                                        }
                                }
                        }
                }
        }

        //returns sample id
        public String get_sample_id()
        {
                return sample_id;
        }

        //returns list of dlls
        public ArrayList<DLL> get_dll()
        {
                return dll;
        }
}
```

DLL

```
package feature.extractor;

//class DLL holds information found by the text parser
//analyzing static reports from pefile
public class DLL
{
        private String dll_type;
        private String dll_function;
        private String dll_name_function; //name + function

        public DLL(String dll_type, String dll_function)
        {
                this.dll_type = dll_type;
                this.dll_function = dll_function;
                this.dll_name_function = dll_type.toLowerCase()+"."+dll_function;
        }

        public void set_dll_type(String dll_type)
        {
                this.dll_type = dll_type;
        }

        public void set_dll_function(String dll_function)
        {
                this.dll_function = dll_function;
        }

        public String get_dll_type()
        {
                return dll_type;
        }

        public String get_dll_function()
        {
                return dll_function;
        }

        public String get_dll_name_function()
        {
                return dll_name_function;
        }

}
```

XML

```
package feature.extractor;

import org.w3c.dom.*;

//class XML is a xml—parser for dynamic reports by Anubis
//it looks for different entities related to activities of
//DLLs, network, registry, files and processes
public class XML
{
        private String sample_id;
        private DLLDependency dll_dependencies;
        private RegistryActivity registry_activities;
        private FileActivity file_activities;
        private ProcessActivity process_activities;
        private NetworkActivity network_activities;

        public XML()
        {
                this.dll_dependencies = null;
                this.registry_activities = null;
                this.file_activities = null;
                this.process_activities = null;
                this.network_activities = null;
        }

        //return sample_id
        public String get_sample_id()
        {
                return sample_id;
        }

        //returns dll dependencies
        public DLLDependency get_dll()
        {
                return dll_dependencies;
        }

        //returns registry activities
        public RegistryActivity get_registry()
        {
                return registry_activities;
        }

        //returns file activities
        public FileActivity get_file()
        {
                return file_activities;
        }

        //returns process activities
        public ProcessActivity get_process()
        {
                return process_activities;
        }

        //returns network activities
        public NetworkActivity get_network()
        {
                return network_activities;
        }

        public void analyze_document(Document doc, String id)
        {
```

```
sample_id = id;

//DLL DEPENDENCIES
//get loaded_dll attributes
NodeList loaded_dll = doc.getElementsByTagName("loaded_dll");
DLLDependency dll_dep = new DLLDependency();
for(int i = 0; i < loaded_dll.getLength(); i++)
{
        Element element = (Element)loaded_dll.item(i);
        dll_dep.add_loaded_dll(element);
}
dll_dependencies = dll_dep;

//REGISTRY ACTIVITY
//get reg_key_created
RegistryActivity reg_act = new RegistryActivity();
NodeList reg_key_created = doc.getElementsByTagName("reg_key_created");
for(int i = 0; i < reg_key_created.getLength(); i++)
{
        Element element = (Element)reg_key_created.item(i);
        reg_act.add_reg_key_created(element);
}
//get reg_value_modified
NodeList reg_value_modified = doc.getElementsByTagName("reg_value_modified"
     );
for(int i = 0; i < reg_value_modified.getLength(); i++)
{
        Element element = (Element)reg_value_modified.item(i);
        reg_act.add_reg_value_modified(element);
}
//get reg_value_read
NodeList reg_value_read = doc.getElementsByTagName("reg_value_read");
for(int i = 0; i < reg_value_read.getLength(); i++)
{
        Element element = (Element)reg_value_read.item(i);
        reg_act.add_reg_value_read(element);
}
registry_activities = reg_act;

//FILE ACTIVITY!
//get file_created
FileActivity file_act = new FileActivity();
NodeList file_created = doc.getElementsByTagName("file_created");
for(int i = 0; i < file_created.getLength(); i++)
{
        Element element = (Element)file_created.item(i);
        file_act.add_file_created(element);
}
//get file_modified
NodeList file_modified = doc.getElementsByTagName("file_modified");
for(int i = 0; i < file_modified.getLength(); i++)
{
        Element element = (Element)file_modified.item(i);
        file_act.add_file_modified(element);
}
//get file_deleted
NodeList file_deleted = doc.getElementsByTagName("file_deleted");
for(int i = 0; i < file_deleted.getLength(); i++)
{
        Element element = (Element)file_deleted.item(i);
        file_act.add_file_deleted(element);
}
//get file_read
NodeList file_read = doc.getElementsByTagName("file_read");
for(int i = 0; i < file_read.getLength(); i++)
{
        Element element = (Element)file_read.item(i);
        file_act.add_file_read(element);
}
//get section_object_created
NodeList section_object = doc.getElementsByTagName("section_object_created"
     );
for(int i = 0; i < section_object.getLength(); i++)
{
        Element element = (Element)section_object.item(i);
        file_act.add_section_object_created(element);
}
//get device_control_communication
NodeList device_control = doc.getElementsByTagName("
     device_control_communication");
```

```
for(int i = 0; i < device_control.getLength(); i++)
{
        Element element = (Element)device_control.item(i);
        file_act.add_device_control_communication(element);
}
//get fs_control_communication
NodeList fs_control = doc.getElementsByTagName("fs_control_communication");
for(int i = 0; i < fs_control.getLength(); i++)
{
        Element element = (Element)fs_control.item(i);
        file_act.add_fs_control_communication(element);
}
file_activities = file_act;

//PROCESS ACTIVITY!
//get thread_status
ProcessActivity process_act = new ProcessActivity();
NodeList thread_status = doc.getElementsByTagName("thread_status");
for(int i = 0; i < thread_status.getLength(); i++)
{
        Element element = (Element)thread_status.item(i);
        process_act.add_thread_status(element);
}
//get remote_thread_created
NodeList remote_thread = doc.getElementsByTagName("remote_thread_created");
for(int i = 0; i < remote_thread.getLength(); i++)
{
        Element element = (Element)remote_thread.item(i);
        process_act.add_remote_thread_created(element);
}
//get process_created
NodeList process_created = doc.getElementsByTagName("process_created");
for(int i = 0; i < process_created.getLength(); i++)
{
        Element element = (Element)process_created.item(i);
        process_act.add_process_created(element);
}
process_activities = process_act;

//NETWORK ACTIVITY!
//get socket
NetworkActivity network_act = new NetworkActivity();
NodeList socket = doc.getElementsByTagName("socket");
for(int i = 0; i < socket.getLength(); i++)
{
        Element element = (Element)socket.item(i);
        network_act.add_socket(element);
}
//get dns_query
NodeList dns = doc.getElementsByTagName("dns_query");
for(int i = 0; i < dns.getLength(); i++)
{
        Element element = (Element)dns.item(i);
        network_act.add_dns_query(element);
}
//get smtp_conversation
NodeList smtp = doc.getElementsByTagName("smtp_conversation");
for(int i = 0; i < smtp.getLength(); i++)
{
        Element element = (Element)smtp.item(i);
        network_act.add_smtp_conv(element);
}
//get http_conversation
NodeList http = doc.getElementsByTagName("http_conversation");
for(int i = 0; i < http.getLength(); i++)
{
        Element element = (Element)http.item(i);
        network_act.add_http_conv(element);
}
//get tcp_conversation
NodeList tcp = doc.getElementsByTagName("tcp_conversation");
for(int i = 0; i < tcp.getLength(); i++)
{
        Element element = (Element)tcp.item(i);
        network_act.add_tcp_conv(element);
}
//get udp_conversation
NodeList udp = doc.getElementsByTagName("udp_conversation");
for(int i = 0; i < udp.getLength(); i++)
{
```

```
                        Element element = (Element)udp.item(i);
                        network_act.add_udp_conv(element);
                }
                network_activities = network_act;
        }
}
```

DLLDependency

```
package feature.extractor;

import org.w3c.dom.*;
import java.util.ArrayList;

//class DLLDependency holding DLL dependency related information
//found by the XML-parser analyzing dynamic reports from Anubis
public class DLLDependency
{
        private ArrayList<LoadedDLL> loaded_dlls;

        public DLLDependency()
        {
                loaded_dlls = new ArrayList<LoadedDLL>();
        }

        //adding loaded dll values
        public void add_loaded_dll(Element dll)
        {
                NamedNodeMap attrs = dll.getAttributes();
                LoadedDLL loaded_dll = new LoadedDLL();

                for(int i = 0; i < attrs.getLength(); i++)
                {
                        Attr current = (Attr)attrs.item(i);

                        if(current.getName().compareTo("base_address") == 0)
                                loaded_dll.set_base_address(current.getValue());
                        if(current.getName().compareTo("base_name") == 0)
                                loaded_dll.set_base_name(current.getValue());
                        if(current.getName().compareTo("full_name") == 0)
                                loaded_dll.set_full_name(current.getValue());
                        if(current.getName().compareTo("is_load_time_dependency") == 0)
                                loaded_dll.set_is_load_time_dependency(current.getValue());
                        if(current.getName().compareTo("load_time") == 0)
                                loaded_dll.set_load_time(current.getValue());
                        if(current.getName().compareTo("size") == 0)
                                loaded_dll.set_size(current.getValue());
                }
                loaded_dlls.add(loaded_dll);

        }

        public ArrayList<LoadedDLL> get_loaded_dlls()
        {
                return loaded_dlls;
        }

        //inner class
        public class LoadedDLL
        {
                private String base_address;
                private String base_name;
                private String full_name;
                private String is_load_time_dependency;
                private String load_time;
                private String size;

                public LoadedDLL()
                {
                        base_address = null;
                        base_name = null;
                        full_name = null;
                        is_load_time_dependency = null;
                        load_time = null;
                        size = null;
                }

                public void set_base_address(String base_address){this.base_address =
                     base_address;}
                public void set_base_name(String base_name){this.base_name = base_name;}
                public void set_full_name(String full_name){this.full_name = full_name;}
                public void set_is_load_time_dependency(String is_load_time_dependency){
                    this.is_load_time_dependency = is_load_time_dependency;}
                public void set_load_time(String load_time){this.load_time = load_time;}
                public void set_size(String size){this.size = size;}

                public String get_base_address(){return base_address;}
                public String get_base_name(){return base_name;}
                public String get_full_name(){return full_name;}
                public String get_is_load_time_dependency(){return is_load_time_dependency
                    ;}
                public String get_load_time(){return load_time;}
                public String get_size(){return size;}
        }
}
```

RegistryActivity

```
package feature.extractor;

import org.w3c.dom.*;

import java.util.ArrayList;

//class RegistryActivity storing registry activity related information
//found by the XML-parser analyzing dynamic reports from Anubis
public class RegistryActivity
{
        private ArrayList<RegKeyCreated> reg_key_created;
        private ArrayList<RegValueModified> reg_value_modified;
        private ArrayList<RegValueRead> reg_value_read;

        public RegistryActivity()
        {
                reg_key_created = new ArrayList<RegKeyCreated>();
                reg_value_modified = new ArrayList<RegValueModified>();
                reg_value_read = new ArrayList<RegValueRead>();
        }

        //adding created registry keys
        public void add_reg_key_created(Element reg)
        {
                NamedNodeMap attrs = reg.getAttributes();
                RegKeyCreated reg_key = new RegKeyCreated();

                if(attrs.getLength() > 0)
                {
                        Attr a_name = (Attr)attrs.item(0);
                        reg_key.set_name(a_name.getValue());
                        reg_key_created.add(reg_key);
                }
        }

        //adding modified registry values
        public void add_reg_value_modified(Element reg)
        {
                NamedNodeMap attrs = reg.getAttributes();
                RegValueModified reg_mod = new RegValueModified();

                for(int i = 0; i < attrs.getLength(); i++)
                {
                        Attr current = (Attr)attrs.item(i);

                        if(current.getName().compareTo("count") == 0)
                                reg_mod.set_count(current.getValue());
                        if(current.getName().compareTo("description") == 0)
                                reg_mod.set_description(current.getValue());
                        if(current.getName().compareTo("key") == 0)
                                reg_mod.set_key(current.getValue());
                        if(current.getName().compareTo("value_data") == 0)
                                reg_mod.set_value_data(current.getValue());
                        if(current.getName().compareTo("value_name") == 0)
                                reg_mod.set_value_name(current.getValue());
                }
                reg_value_modified.add(reg_mod);
        }

        //adding read registry values
        public void add_reg_value_read(Element reg)
        {
```

```
		NamedNodeMap attrs = reg.getAttributes();
		RegValueRead reg_read = new RegValueRead();

		for(int i = 0; i < attrs.getLength(); i++)
		{
			Attr current = (Attr)attrs.item(i);

			if(current.getName().compareTo("count") == 0)
				reg_read.set_count(current.getValue());
			if(current.getName().compareTo("key") == 0)
				reg_read.set_key(current.getValue());
			if(current.getName().compareTo("value_data") == 0)
				reg_read.set_value_data(current.getValue());
			if(current.getName().compareTo("value_name") == 0)
				reg_read.set_value_name(current.getValue());
		}
		reg_value_read.add(reg_read);
	}

	//returning created registry keys
	public ArrayList<RegKeyCreated> get_reg_key_created()
	{
		return reg_key_created;
	}

	//returning modified registry values
	public ArrayList<RegValueModified> get_reg_value_modified()
	{
		return reg_value_modified;
	}

	//returning read registry value
	public ArrayList<RegValueRead> get_reg_value_read()
	{
		return reg_value_read;
	}

	//inner class
	public class RegKeyCreated
	{
		private String name;

		public RegKeyCreated()
		{
			name = null;
		}

		public void set_name(String name){this.name = name;}
		public String get_name(){return name;}
	}

	//inner class
	public class RegValueModified
	{
		private String count;
		private String description;
		private String key;
		private String value_data;
		private String value_name;

		public RegValueModified()
		{
			count = null;
			description = null;
			key = null;
			value_data = null;
			value_name = null;
		}

		public void set_count(String count){this.count = count;}
		public void set_description(String description){this.description =
		    description;}
		public void set_key(String key){this.key = key;}
		public void set_value_data(String value_data){this.value_data = value_data
		    ;}
		public void set_value_name(String value_name){this.value_name = value_name
		    ;}

		public String get_count(){return count;}
		public String get_description(){return description;}
```

```
                public String get_key(){return key;}
                public String get_value_data(){return value_data;}
                public String get_value_name(){return value_name;}
        }

        //inner class
        public class RegValueRead
        {
                private String count;
                private String key;
                private String value_data;
                private String value_name;

                public RegValueRead()
                {
                        count = null;
                        key = null;
                        value_data = null;
                        value_name = null;
                }

                public void set_count(String count){this.count = count;}
                public void set_key(String key){this.key = key;}
                public void set_value_data(String value_data){this.value_data = value_data
                    ;}
                public void set_value_name(String value_name){this.value_name = value_name
                    ;}

                public String get_count(){return count;}
                public String get_key(){return key;}
                public String get_value_data(){return value_data;}
                public String get_value_name(){return value_name;}
        }
}
```

NetworkActivity

```
package feature.extractor;

import java.util.ArrayList;
import org.w3c.dom.*;

//class NetworkActivity storing network activity related information
//found by the XML-parser analyzing dynamic reports from Anubis
public class NetworkActivity
{
        private ArrayList<Socket> socket;
        private ArrayList<DNSQuery> dns_query;
        private ArrayList<SMTPConversation> smtp_conversation;
        private ArrayList<HTTPConversation> http_conversation;
        private ArrayList<TCPConversation> tcp_conversation;
        private ArrayList<UDPConversation> udp_conversation;

        public NetworkActivity()
        {
                socket = new ArrayList<Socket>();
                dns_query = new ArrayList<DNSQuery>();
                smtp_conversation = new ArrayList<SMTPConversation>();
                http_conversation = new ArrayList<HTTPConversation>();
                tcp_conversation = new ArrayList<TCPConversation>();
                udp_conversation = new ArrayList<UDPConversation>();
        }

        //adding socket values
        public void add_socket(Element net)
        {
                NamedNodeMap attrs = net.getAttributes();
                Socket s = new Socket();

                for(int i = 0; i < attrs.getLength(); i++)
                {
                        Attr current = (Attr)attrs.item(i);

                        if(current.getName().compareTo("close_time") == 0)
                                s.set_close_time(current.getValue());
                        if(current.getName().compareTo("create_time") == 0)
                                s.set_create_time(current.getValue());
                        if(current.getName().compareTo("created_by_thread") == 0)
                                s.set_create_time(current.getValue());
                        if(current.getName().compareTo("foreign_ip") == 0)
                        s.set_foreign_ip(current.getValue());
                if(current.getName().compareTo("foreign_port") == 0)
                        s.set_foreign_port(current.getValue());
                if(current.getName().compareTo("is_listening") == 0)
                        s.set_is_listening(current.getValue());
                if(current.getName().compareTo("local_ip") == 0)
                        s.set_local_ip(current.getValue());
                if(current.getName().compareTo("local_port") == 0)
                        s.set_local_port(current.getValue());
        }
        socket.add(s);
}

//adding dns query values
public void add_dns_query(Element net)
{
        NamedNodeMap attrs = net.getAttributes();
        DNSQuery dq= new DNSQuery();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("name") == 0)
                        dq.set_name(current.getValue());
                if(current.getName().compareTo("result") == 0)
                        dq.set_result(current.getValue());
                if(current.getName().compareTo("sucessfull") == 0)
                        dq.set_successfull(current.getValue());
                if(current.getName().compareTo("type") == 0)
                        dq.set_type(current.getValue());
        }
        dns_query.add(dq);
}

//adding smtp conversation values
public void add_smtp_conv(Element net)
{
        NamedNodeMap attrs = net.getAttributes();
        SMTPConversation sc = new SMTPConversation();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("content") == 0)
                        sc.set_content(current.getValue());
                if(current.getName().compareTo("description") == 0)
                        sc.set_description(current.getValue());
                if(current.getName().compareTo("dest_ip") == 0)
                        sc.set_dest_ip(current.getValue());
                if(current.getName().compareTo("dest_port") == 0)
                        sc.set_dest_port(current.getValue());
                if(current.getName().compareTo("recipient") == 0)
                        sc.set_recipient(current.getValue());
                if(current.getName().compareTo("sender") == 0)
                        sc.set_sender(current.getValue());
                if(current.getName().compareTo("server_reply") == 0)
                        sc.set_server_reply(current.getValue());
                if(current.getName().compareTo("src_ip") == 0)
                        sc.set_src_ip(current.getValue());
                if(current.getName().compareTo("src_port") == 0)
                        sc.set_src_port(current.getValue());
                if(current.getName().compareTo("subject") == 0)
                        sc.set_subject(current.getValue());
        }
        smtp_conversation.add(sc);
}

//adding http conversation values
public void add_http_conv(Element net)
{
        NamedNodeMap attrs = net.getAttributes();
        HTTPConversation hc = new HTTPConversation();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("dest_ip") == 0)
```

```
                                hc.set_dest_ip(current.getValue());
                        if(current.getName().compareTo("dest_port") == 0)
                                hc.set_dest_port(current.getValue());
                        if(current.getName().compareTo("hostname") == 0)
                                hc.set_hostname(current.getValue());
                        if(current.getName().compareTo("src_ip") == 0)
                                hc.set_src_ip(current.getValue());
                        if(current.getName().compareTo("src_port") == 0)
                                hc.set_src_port(current.getValue());
                }
                http_conversation.add(hc);
        }

        //adding tcp conversation values
        public void add_tcp_conv(Element net)
        {
                NamedNodeMap attrs = net.getAttributes();
                TCPConversation tc = new TCPConversation();

                for(int i = 0; i < attrs.getLength(); i++)
                {
                        Attr current = (Attr)attrs.item(i);

                        if(current.getName().compareTo("dest_ip") == 0)
                                tc.set_dest_ip(current.getValue());
                        if(current.getName().compareTo("dest_port") == 0)
                                tc.set_dest_port(current.getValue());
                        if(current.getName().compareTo("org_bytes_sent") == 0)
                                tc.set_org_bytes_sent(current.getValue());
                        if(current.getName().compareTo("res_bytes_sent") == 0)
                                tc.set_res_bytes_sent(current.getValue());
                        if(current.getName().compareTo("src_ip") == 0)
                                tc.set_src_ip(current.getValue());
                        if(current.getName().compareTo("src_port") == 0)
                                tc.set_src_port(current.getValue());
                        if(current.getName().compareTo("state") == 0)
                                tc.set_state(current.getValue());
                }
                tcp_conversation.add(tc);
        }

        //adding udp conversation values
        public void add_udp_conv(Element net)
        {
                NamedNodeMap attrs = net.getAttributes();
                UDPConversation tc = new UDPConversation();

                for(int i = 0; i < attrs.getLength(); i++)
                {
                        Attr current = (Attr)attrs.item(i);

                        if(current.getName().compareTo("dest_ip") == 0)
                                tc.set_dest_ip(current.getValue());
                        if(current.getName().compareTo("dest_port") == 0)
                                tc.set_dest_port(current.getValue());
                        if(current.getName().compareTo("org_bytes_sent") == 0)
                                tc.set_org_bytes_sent(current.getValue());
                        if(current.getName().compareTo("res_bytes_sent") == 0)
                                tc.set_res_bytes_sent(current.getValue());
                        if(current.getName().compareTo("src_ip") == 0)
                                tc.set_src_ip(current.getValue());
                        if(current.getName().compareTo("src_port") == 0)
                                tc.set_src_port(current.getValue());
                        if(current.getName().compareTo("state") == 0)
                                tc.set_state(current.getValue());
                }
                udp_conversation.add(tc);
        }

        public ArrayList<Socket> get_socket()
        {
                return socket;
        }

        public ArrayList<DNSQuery> get_dns_query()
        {
                return dns_query;
        }

        public ArrayList<SMTPConversation> get_smtp_conversation()
```

```
{
        return smtp_conversation;
}

public ArrayList<HTTPConversation> get_http_conversation()
{
        return http_conversation;
}

public ArrayList<TCPConversation> get_tcp_conversation()
{
        return tcp_conversation;
}

public ArrayList<UDPConversation> get_udp_conversation()
{
        return udp_conversation;
}

//inner class
public class Socket
{
        private String close_time;
        private String create_time;
        private String created_by_thread;
        private String foreign_ip;
        private String foreign_port;
        private String is_listening;
        private String local_ip;
        private String local_port;

        public Socket()
        {
                close_time = null;
                create_time = null;
                created_by_thread = null;
                foreign_ip = null;
                foreign_port = null;
                is_listening = null;
                local_ip = null;
                local_port = null;
        }

        public void set_close_time(String close_time){this.close_time = close_time
             ;}
        public void set_create_time(String create_time){this.create_time =
             create_time;}
        public void set_created_by_thread(String created_by_thread){this.
             created_by_thread = created_by_thread;}
        public void set_foreign_ip(String foreign_ip){this.foreign_ip = foreign_ip
             ;}
        public void set_foreign_port(String foreign_port){this.foreign_port =
             foreign_port;}
        public void set_is_listening(String is_listening){this.is_listening =
             is_listening;}
        public void set_local_ip(String local_ip){this.local_ip = local_ip;}
        public void set_local_port(String local_port){this.local_port = local_port
             ;}

        public String get_close_time(){return close_time;}
        public String get_create_time(){return create_time;}
        public String get_created_by_thread(){return created_by_thread;}
        public String get_foreign_ip(){return foreign_ip;}
        public String get_foreign_port(){return foreign_port;}
        public String get_is_listening(){return is_listening;}
        public String get_local_ip(){return local_ip;}
        public String get_local_port(){return local_port;}
}
//inner class
public class DNSQuery
{
        private String name;
        private String result;
        private String successfull;
        private String type;

        public DNSQuery()
        {
                name = null;
                result = null;
```

```
                        successfull = null;
                        type = null;
                }

                public void set_name(String name){this.name = name;}
                public void set_result(String resutl){this.result = result;}
                public void set_successfull(String successfull){this.successfull =
                        successfull;}
                public void set_type(String type){this.type = type;}

                public String get_name(){return name;}
                public String get_result(){return result;}
                public String get_successfull(){return successfull;}
                public String get_type(){return type;}
        }
        //inner class
        public class SMTPConversation
        {
                private String content;
                private String description;
                private String dest_ip;
                private String dest_port;
                private String recipient;
                private String sender;
                private String server_reply;
                private String src_ip;
                private String src_port;
                private String subject;

                public SMTPConversation()
                {
                        content = null;
                        description = null;
                        dest_ip = null;
                        dest_port = null;
                        recipient = null;
                        sender = null;
                        server_reply = null;
                        src_ip = null;
                        src_port = null;
                        subject = null;
                }

                public void set_content(String content){this.content = content;}
                public void set_description(String description){this.description =
                        description;}
                public void set_dest_ip(String dest_ip){this.dest_ip = dest_ip;}
                public void set_dest_port(String dest_port){this.dest_port = dest_port;}
                public void set_recipient(String recipient){this.recipient = recipient;}
                public void set_sender(String sender){this.sender = sender;}
                public void set_server_reply(String server_reply){this.server_reply =
                        server_reply;}
                public void set_src_ip(String src_ip){this.src_ip = src_ip;}
                public void set_src_port(String src_port){this.src_port = src_port;}
                public void set_subject(String subject){this.subject = subject;}

                public String get_content(){return content;}
                public String get_description(){return description;}
                public String get_dest_ip(){return dest_ip;}
                public String get_dest_port(){return dest_port;}
                public String get_recipient(){return recipient;}
                public String get_sender(){return sender;}
                public String get_server_reply(){return server_reply;}
                public String get_src_ip(){return src_ip;}
                public String get_src_port(){return src_port;}
                public String get_subject(){return subject;}
        }
        //inner class
        public class HTTPConversation
        {
                private String dest_ip;
                private String dest_port;
                private String hostname;
                private String src_ip;
                private String src_port;

                public HTTPConversation()
                {
                        dest_ip = null;
                        dest_port = null;
                hostname = null;
                src_ip = null;
                src_port = null;
        }

        public void set_dest_ip(String dest_ip){this.dest_ip = dest_ip;}
        public void set_dest_port(String dest_port){this.dest_port = dest_port;}
        public void set_hostname(String hostname){this.hostname = hostname;}
        public void set_src_ip(String src_ip){this.src_ip = src_ip;}
        public void set_src_port(String src_port){this.src_port = src_port;}

        public String get_dest_ip(){return dest_ip;}
        public String get_dest_port(){return dest_port;}
        public String get_hostname(){return hostname;}
        public String get_src_ip(){return src_ip;}
        public String get_src_port(){return src_port;}
}
//inner class
public class TCPConversation
{
        private String dest_ip;
        private String dest_port;
        private String org_bytes_sent;
        private String res_bytes_sent;
        private String src_ip;
        private String src_port;
        private String state;

        public TCPConversation()
        {
                dest_ip = null;
                dest_port = null;
                org_bytes_sent = null;
                res_bytes_sent = null;
                src_ip = null;
                state = null;
        }

        public void set_dest_ip(String dest_ip){this.dest_ip = dest_ip;}
        public void set_dest_port(String dest_port){this.dest_port = dest_port;}
        public void set_org_bytes_sent(String org_bytes_sent){this.org_bytes_sent =
              org_bytes_sent;}
        public void set_res_bytes_sent(String res_bytes_sent){this.res_bytes_sent =
              res_bytes_sent;}
        public void set_src_ip(String src_ip){this.src_ip = src_ip;}
        public void set_src_port(String src_port){this.src_port = src_port;}
        public void set_state(String state){this.state = state;}

        public String get_dest_ip(){return dest_ip;}
        public String get_dest_port(){return dest_port;}
        public String get_org_bytes_sent(){return org_bytes_sent;}
        public String get_res_bytes_sent(){return res_bytes_sent;}
        public String get_src_ip(){return src_ip;}
        public String get_src_port(){return src_port;}
        public String get_state(){return state;}
}
//inner class
public class UDPConversation
{
        private String dest_ip;
        private String dest_port;
        private String org_bytes_sent;
        private String res_bytes_sent;
        private String src_ip;
        private String src_port;
        private String state;

        public UDPConversation()
        {
                dest_ip = null;
                dest_port = null;
                org_bytes_sent = null;
                res_bytes_sent = null;
                src_ip = null;
                state = null;
        }

        public void set_dest_ip(String dest_ip){this.dest_ip = dest_ip;}
        public void set_dest_port(String dest_port){this.dest_port = dest_port;}
        public void set_org_bytes_sent(String org_bytes_sent){this.org_bytes_sent =
                org_bytes_sent;}
        public void set_res_bytes_sent(String res_bytes_sent){this.res_bytes_sent =
                res_bytes_sent;}
        public void set_src_ip(String src_ip){this.src_ip = src_ip;}
        public void set_src_port(String src_port){this.src_port = src_port;}
        public void set_state(String state){this.state = state;}

        public String get_dest_ip(){return dest_ip;}
        public String get_dest_port(){return dest_port;}
        public String get_org_bytes_sent(){return org_bytes_sent;}
        public String get_res_bytes_sent(){return res_bytes_sent;}
        public String get_src_ip(){return src_ip;}
        public String get_src_port(){return src_port;}
        public String get_state(){return state;}
    }
}
```

FileActivity

```
package feature.extractor;

import org.w3c.dom.*;

import java.util.ArrayList;

//class FileActivity storing file activity related information
//found by the XML-parser analyzing dynamic reports from Anubis
public class FileActivity
{
    private ArrayList<FileCreated> file_created;
    private ArrayList<FileModified> file_modified;
    private ArrayList<FileDeleted> file_deleted;
    private ArrayList<FileRead> file_read;
    private ArrayList<SectionObjectCreated> section_object_created;
    private ArrayList<DeviceControlCommunication> device_control_communication;
    private ArrayList<FsControlCommunication> fs_control_communication;

    public FileActivity()
    {
        file_created = new ArrayList<FileCreated>();
        file_modified = new ArrayList<FileModified>();
        file_deleted = new ArrayList<FileDeleted>();
        file_read = new ArrayList<FileRead>();
        section_object_created = new ArrayList<SectionObjectCreated>();
        device_control_communication = new ArrayList<DeviceControlCommunication>();
        fs_control_communication = new ArrayList<FsControlCommunication>();
    }

    //adding file created values
    public void add_file_created(Element file)
    {
        NamedNodeMap attrs = file.getAttributes();
        FileCreated fcreated = new FileCreated();

        if(attrs.getLength() > 0)
        {
            Attr a_name = (Attr)attrs.item(0);
            fcreated.set_name(a_name.getValue());
            file_created.add(fcreated);
        }
    }

    //adding file created values
    public void add_file_modified(Element file)
    {
        NamedNodeMap attrs = file.getAttributes();
        FileModified fmodified = new FileModified();

        for(int i = 0; i < attrs.getLength(); i++)
        {
            Attr current = (Attr)attrs.item(i);

            if(current.getName().compareTo("description") == 0)
                fmodified.set_description(current.getValue());
            if(current.getName().compareTo("name") == 0)
                fmodified.set_name(current.getValue());
        }
        file_modified.add(fmodified);
    }
```

```
//adding file deleted values
public void add_file_deleted(Element file)
{
        NamedNodeMap attrs = file.getAttributes();
        FileDeleted fdeleted = new FileDeleted();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("description") == 0)
                        fdeleted.set_description(current.getValue());
                if(current.getName().compareTo("name") == 0)
                        fdeleted.set_name(current.getValue());
        }
        file_deleted.add(fdeleted);
}

//adding file read values
public void add_file_read(Element file)
{
        NamedNodeMap attrs = file.getAttributes();
        FileRead fread = new FileRead();

        if(attrs.getLength() > 0)
        {
                Attr a_name = (Attr)attrs.item(0);
                fread.set_name(a_name.getValue());
                file_read.add(fread);
        }
}

//adding section object created values
public void add_section_object_created(Element object)
{
        NamedNodeMap attrs = object.getAttributes();
        SectionObjectCreated soc = new SectionObjectCreated();

        if(attrs.getLength() > 0)
        {
                Attr a_file_name = (Attr)attrs.item(0);
                soc.set_file_name(a_file_name.getValue());
                section_object_created.add(soc);
        }
}

//adding device control communication values
public void add_device_control_communication(Element control)
{
        NamedNodeMap attrs = control.getAttributes();
        DeviceControlCommunication dcc = new DeviceControlCommunication();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("control_code") == 0)
                        dcc.set_control_code(current.getValue());
                if(current.getName().compareTo("count") == 0)
                        dcc.set_count(current.getValue());
                if(current.getValue().compareTo("file") == 0)
                        dcc.set_file(current.getValue());
        }
        device_control_communication.add(dcc);
}

//adding fs control communication values
public void add_fs_control_communication(Element control)
{
        NamedNodeMap attrs = control.getAttributes();
        FsControlCommunication dcc = new FsControlCommunication();

        for(int i = 0; i < attrs.getLength(); i++)
        {
                Attr current = (Attr)attrs.item(i);

                if(current.getName().compareTo("control_code") == 0)
                        dcc.set_control_code(current.getValue());
                if(current.getName().compareTo("count") == 0)
                        dcc.set_count(current.getValue());
                if(current.getValue().compareTo("file") == 0)
                    dcc.set_file(current.getValue());
            }
            fs_control_communication.add(dcc);
    }

    public ArrayList<FileCreated> get_file_created()
    {
            return file_created;
    }

    public ArrayList<FileModified> get_file_modified()
    {
            return file_modified;
    }

    public ArrayList<FileDeleted> get_file_deleted()
    {
            return file_deleted;
    }

    public ArrayList<FileRead> get_file_read()
    {
            return file_read;
    }

    public ArrayList<SectionObjectCreated> get_section_object_created()
    {
            return section_object_created;
    }

    public ArrayList<DeviceControlCommunication> get_device_control_communication()
    {
            return device_control_communication;
    }

    public ArrayList<FsControlCommunication> get_fs_control_communcation()
    {
            return fs_control_communication;
    }

    //inner class
    public class FileCreated
    {
            private String name;

            public FileCreated()
            {
                    name = null;
            }

            public void set_name(String name){this.name = name;}
            public String get_name(){return name;}
    }

    //inner class
    public class FileModified
    {
            private String description;
            private String name;

            public FileModified()
            {
                    description = null;
                    name = null;
            }

            public void set_description(String description){this.description =
                 description;}
            public void set_name(String name){this.name = name;}

            public String get_description(){return description;}
            public String get_name(){return name;}
    }

    //inner class
    public class FileDeleted
    {
            private String description;
            private String name;
```

```
        public FileDeleted()
        {
                description = null;
                name = null;
        }

        public void set_description(String description){this.description =
            description;}
        public void set_name(String name){this.name = name;}

        public String get_description(){return description;}
        public String get_name(){return name;}
}

//inner class
public class FileRead
{
        private String name;

        public FileRead()
        {
                name = null;
        }

        public void set_name(String name){this.name=name;}
        public String get_name(){return name;}
}

//inner class
public class SectionObjectCreated
{
        private String file_name;

        public SectionObjectCreated()
        {
                file_name = null;
        }

        public void set_file_name(String file_name){this.file_name = file_name;}
        public String get_file_name(){return file_name;}
}

//inner class
public class DeviceControlCommunication
{
        private String control_code;
        private String count;
        private String file;

        public DeviceControlCommunication()
        {
                control_code = null;
                count = null;
                file = null;
        }

        public void set_control_code(String control_code){this.control_code =
            control_code;}
        public void set_count(String count){this.count = count;}
        public void set_file(String file){this.file = file;}

        public String get_control_code(){return control_code;}
        public String get_count(){return count;}
        public String get_file(){return file;}
}

//inner class
public class FsControlCommunication
{
        private String control_code;
        private String count;
        private String file;

        public FsControlCommunication()
        {
                control_code = null;
                count = null;
                file = null;
        }
```

```
                public void set_control_code(String control_code){this.control_code =
                     control_code;}
                public void set_count(String count){this.count = count;}
                public void set_file(String file){this.file = file;}

                public String get_control_code(){return control_code;}
                public String get_count(){return count;}
                public String get_file(){return file;}
        }
}
```

FeatureExtractor

```
package feature.extractor;

import java.io.IOException;
import java.util.ArrayList;

import feature.extractor.FeatureDynamic.DynamicString;
import feature.extractor.FeatureStatic.StaticDLL;

//class FeatureExtractor is the controlling entity for
//extracting features and building static, dynamic and
//combined feature sets
public class FeatureExtractor
{
        private Parser parser;
        private ARFF arff;

        private ArrayList<FeatureStatic> feature_static;
        private ArrayList<FeatureDynamic> feature_dynamic;
        private ArrayList<FeatureCombo> feature_combo;

        public FeatureExtractor()
        {
                parser = new Parser();
                arff = new ARFF();
                feature_static = new ArrayList<FeatureStatic>();
                feature_dynamic = new ArrayList<FeatureDynamic>();
                feature_combo = new ArrayList<FeatureCombo>();
        }

        //method extracts static features and build a static feature set
        public void extract_static_features() throws IOException
        {
                System.out.println("Start static feature extraction");

                //open files
                parser.open_static_reports();

                //call parser
                System.out.println("Start analyzing TXT documents");
                parser.parse_pe();
                System.out.println("Done analyzing TXT documents");

                //prepare features
                System.out.println("Start prepering static features");
                ArrayList<PE> pe_samples = parser.get_pe_samples();

                //iterate through list
                for(int i = 0; i < pe_samples.size(); i++)
                {
                        PE current_pe = pe_samples.get(i);
                        FeatureStatic feature = new FeatureStatic();
                        feature.add_static_feature(current_pe);
                        //System.out.println();
                        //store object in list
                        feature_static.add(feature);
                }

                System.out.println("Done preparing static features");
        }

        //method extracts dynamic features and build a dynamic feature set
        public void extract_dynamic_features() throws Exception
        {
                System.out.println("Start dynamic feature extraction");

                //open files
                parser.open_dynamic_reports();

                //call parser
                System.out.println("Start analyzing XML documents");
                parser.parse_xml();
                System.out.println("Done anlyzing XML documents");

                //prepare features
                System.out.println("Start preparing dynamic features");
                ArrayList<XML> xml_samples = parser.get_xml_samples();
                //iterate through list
                for(int i = 0; i < xml_samples.size(); i++)
                {
                        XML current_xml = xml_samples.get(i);
                        FeatureDynamic feature = new FeatureDynamic();
                        feature.add_dynamic_features(current_xml);
                        //store object in list
                        feature_dynamic.add(feature);
                }
                System.out.println("Done preparing dynamic features");
        }

        //method combines the static and dynamic feature sets
        public void combine_features()
        {
                System.out.println("\nStart combining features");

                //find dynamic and static feature object with same sample id
                for(int i = 0; i < feature_dynamic.size(); i++)
                {
                        FeatureDynamic fd = feature_dynamic.get(i);
                        String fd_id = fd.get_sample_id();

                        for(int ii = 0; ii < feature_static.size(); ii++)
                        {
                                FeatureStatic fs = feature_static.get(i);
                                String fs_id = fs.get_sample_id();

                                //if sample id match
                                if(fd_id.compareTo(fs_id) == 0)
                                {
                                        //extract list of StaticDLL and DynamicString
                                        ArrayList<StaticDLL> sdll = fs.get_static_dll();
                                        ArrayList<DynamicString> dstring = fd.get_features
                                            ();

                                        //create FeatureCombo object and add to list
                                        feature_combo.add(new FeatureCombo(sdll,dstring,
                                            fd_id));

                                        break;
                                }
                        }
                }
                System.out.println("Done combining features");
        }

        //extracting labels
        public void extract_labels()
        {
                //extract from static data
                for(int i = 0; i < feature_static.size(); i++)
                {
                        feature_static.get(i).extract_label();
                }
                //extract from dynamic data
                for(int i = 0; i < feature_dynamic.size(); i++)
                {
                        feature_dynamic.get(i).extract_label();
                }
                //extract from combo
                for(int i = 0; i < feature_combo.size(); i++)
                {
                        feature_combo.get(i).extract_label();
                }
        }

        //generate ARFF file
        public void generate_arff() throws Exception
        {
                System.out.println("Generate ARFF file");

                arff.generate_arff_dynamic(feature_dynamic);
                arff.generate_arff_static(feature_static);
                arff.generate_arff_combo(feature_combo);

                //arff.generate_arff_ld(feature_dynamic);
        }
}
```

FeatureDynamic

```
package feature.extractor;

import java.util.ArrayList;

import feature.extractor.DLLDependency.*;
import feature.extractor.FileActivity.*;
import feature.extractor.NetworkActivity.*;
import feature.extractor.ProcessActivity.*;
import feature.extractor.RegistryActivity.*;

public class FeatureDynamic
{
        //used when applying > 2 labels
        private String label;
        //used when applying malicious vs benign
        private String label_type;
        private String sample_id;
        private ArrayList<DynamicString> dynamic;

        public FeatureDynamic()
        {
                dynamic = new ArrayList<DynamicString>();
        }

        //return sample id
        public String get_sample_id()
        {
                return sample_id;
        }

        //return label
        public String get_label()
        {
                return label;
        }

        //return label type
        public String get_label_type()
        {
                return label_type;
        }

        //method ass dynamic features
        public void add_dynamic_features(XML xml)
        {
                sample_id = xml.get_sample_id();

                //DLL DEPENDENCY ATTRIBUTES
                DLLDependency dll = xml.get_dll();

                DynamicString dll_base_address = new DynamicString("dll_base_address");
                DynamicString dll_base_name = new DynamicString("dll_base_name");
                DynamicString dll_full_name = new DynamicString("dll_full_name");
                DynamicString dll_is_load_time_dependency = new DynamicString("
                      dll_is_load_time_dependency");
                DynamicString dll_load_time = new DynamicString("dll_load_time");
                DynamicString dll_size= new DynamicString("dll_size");

                ArrayList<LoadedDLL> ldll = dll.get_loaded_dlls();
                for(int i = 0; i < ldll.size(); i++)
                {
                        dll_base_address.add_content(ldll.get(i).get_base_address());
                        dll_base_name.add_content(ldll.get(i).get_base_name());
                        dll_full_name.add_content(ldll.get(i).get_full_name());
                        dll_is_load_time_dependency.add_content(ldll.get(i).
                              get_is_load_time_dependency());
                        dll_load_time.add_content(ldll.get(i).get_load_time());
                        dll_size.add_content(ldll.get(i).get_size());
}
dynamic.add(dll_base_address);
dynamic.add(dll_base_name);
dynamic.add(dll_full_name);
dynamic.add(dll_is_load_time_dependency);
dynamic.add(dll_load_time);
dynamic.add(dll_size);


//REGISTRY ACTIVITY ATTRIBUTES
RegistryActivity registry = xml.get_registry();

DynamicString reg_key_created_name = new DynamicString("
     reg_key_created_name");
ArrayList<RegKeyCreated> rkc = registry.get_reg_key_created();
for(int i = 0; i < rkc.size(); i++)
{
        reg_key_created_name.add_content(rkc.get(i).get_name());
}
dynamic.add(reg_key_created_name);

DynamicString reg_value_modified_count = new DynamicString("
     reg_value_modified_count");
DynamicString reg_value_modified_description = new DynamicString("
     reg_value_modified_description");
DynamicString reg_value_modified_key = new DynamicString("
     reg_value_modified_key");
DynamicString reg_value_modified_value_data = new DynamicString("
     reg_value_modified_value_data");
DynamicString reg_value_modified_value_name = new DynamicString("
     reg_value_modified_value_name");
ArrayList<RegValueModified> rvm = registry.get_reg_value_modified();
for(int i = 0; i < rvm.size(); i++)
{
        reg_value_modified_count.add_content(rvm.get(i).get_count());
        reg_value_modified_description.add_content(rvm.get(i).
             get_description());
        reg_value_modified_key.add_content(rvm.get(i).get_key());
        reg_value_modified_value_data.add_content(rvm.get(i).get_value_data
             ());
        reg_value_modified_value_name.add_content(rvm.get(i).get_value_name
             ());
}
dynamic.add(reg_value_modified_count);
dynamic.add(reg_value_modified_description);
dynamic.add(reg_value_modified_key);
dynamic.add(reg_value_modified_value_data);
dynamic.add(reg_value_modified_value_name);

DynamicString reg_value_read_count = new DynamicString("
     reg_value_read_count");
DynamicString reg_value_read_key = new DynamicString("reg_value_read_key");
DynamicString reg_value_read_value_data = new DynamicString("
     reg_value_read_value_data");
DynamicString reg_value_read_value_name = new DynamicString("
     reg_value_read_value_name");
ArrayList<RegValueRead> rvr = registry.get_reg_value_read();
for(int i = 0; i < rvr.size(); i++)
{
        reg_value_read_count.add_content(rvr.get(i).get_count());
        reg_value_read_key.add_content(rvr.get(i).get_key());
        reg_value_read_value_data.add_content(rvr.get(i).get_value_data());
        reg_value_read_value_name.add_content(rvr.get(i).get_value_name());
}
dynamic.add(reg_value_read_count);
dynamic.add(reg_value_read_key);
dynamic.add(reg_value_read_value_data);
dynamic.add(reg_value_read_value_name);


//FILE ACTITIVIES ATTRIBUTES
FileActivity file = xml.get_file();

DynamicString file_created_name = new DynamicString("file_created_name");
ArrayList<FileCreated> fc = file.get_file_created();
for(int i = 0; i < fc.size(); i++)
{
        file_created_name.add_content(fc.get(i).get_name());
}
```

```
DynamicString file_deleted_description = new DynamicString("
     file_deleted_description");
DynamicString file_deleted_name = new DynamicString("file_deleted_name");
ArrayList<FileDeleted> fd = file.get_file_deleted();
for(int i = 0; i < fd.size(); i++)
{
        file_deleted_description.add_content(fd.get(i).get_description());
        file_deleted_name.add_content(fd.get(i).get_name());
}
dynamic.add(file_deleted_description);
dynamic.add(file_deleted_name);

DynamicString file_modified_description = new DynamicString("
     file_modified_description");
DynamicString file_modified_name = new DynamicString("file_modified_name");
ArrayList<FileModified> fm = file.get_file_modified();
for(int i = 0; i < fm.size(); i++)
{
        file_modified_description.add_content(fm.get(i).get_description());
        file_modified_name.add_content(fm.get(i).get_name());
}
dynamic.add(file_modified_description);
dynamic.add(file_modified_name);

DynamicString file_read_name = new DynamicString("file_read_name");
ArrayList<FileRead> fr = file.get_file_read();
for(int i = 0; i < fr.size(); i++)
{
        file_read_name.add_content(fr.get(i).get_name());
}
dynamic.add(file_read_name);

DynamicString file_object_created_file_name = new DynamicString("
     file_object_created_file_name");
ArrayList<SectionObjectCreated> soc = file.get_section_object_created();
for(int i = 0; i < soc.size(); i++)
{
        file_object_created_file_name.add_content(soc.get(i).get_file_name
             ());
}
dynamic.add(file_object_created_file_name);

DynamicString file_fscom_control_code = new DynamicString("
     file_fscom_control_code");
DynamicString file_fscom_count = new DynamicString("file_fscom_count");
DynamicString file_fscom_file = new DynamicString("file_fscom_file");
ArrayList<FsControlCommunication> fcc = file.get_fs_control_communcation();
for(int i = 0; i < fcc.size(); i++)
{
        file_fscom_control_code.add_content(fcc.get(i).get_control_code());
        file_fscom_count.add_content(fcc.get(i).get_count());
        file_fscom_file.add_content(fcc.get(i).get_file());
}
dynamic.add(file_fscom_control_code);
dynamic.add(file_fscom_count);
dynamic.add(file_fscom_file);

DynamicString file_devicecom_control_code = new DynamicString("
     file_devicecom_control_code");
DynamicString file_devicecom_count = new DynamicString("
     file_devicecom_count");
DynamicString file_devicecom_file = new DynamicString("file_devicecom_file"
     );
ArrayList<DeviceControlCommunication> dcc = file.
     get_device_control_communication();
for(int i = 0; i < dcc.size(); i++)
{
        file_devicecom_control_code.add_content(dcc.get(i).get_control_code
             ());
        file_devicecom_count.add_content(dcc.get(i).get_count());
        file_devicecom_file.add_content(dcc.get(i).get_file());
}
dynamic.add(file_devicecom_control_code);
dynamic.add(file_devicecom_count);
dynamic.add(file_devicecom_file);


//PROCESS ACTIVITY ATTRIBUTES
ProcessActivity process = xml.get_process();
```

```
DynamicString thread_status_number_threads = new DynamicString("
     thread_status_number_threads");
DynamicString thread_status_time = new DynamicString("thread_status_time");
ArrayList<ThreadStatus> ts = process.get_thread_status();
for(int i = 0; i < ts.size(); i++)
{
        thread_status_number_threads.add_content(ts.get(i).
             get_number_of_threads());
        thread_status_time.add_content(ts.get(i).get_time());
}
dynamic.add(thread_status_number_threads);
dynamic.add(thread_status_time);

DynamicString remote_thread_created_process = new DynamicString("
     remote_thread_created_process");
ArrayList<RemoteThreadCreated> rts = process.get_remote_thread_created();
for(int i = 0; i < rts.size(); i++)
{
        remote_thread_created_process.add_content(rts.get(i).get_process())
             ;
}
dynamic.add(remote_thread_created_process);

DynamicString process_created_cmd_line = new DynamicString("
     process_created_cmd_line");
DynamicString process_created_description = new DynamicString("
     process_created_description");
DynamicString process_created_exe_name = new DynamicString("
     process_created_exe_name");
ArrayList<ProcessCreated> pc = process.get_process_created();
for(int i = 0; i < pc.size(); i++)
{
        process_created_cmd_line.add_content(pc.get(i).get_cmd_line());
        process_created_description.add_content(pc.get(i).get_description()
             );
        process_created_exe_name.add_content(pc.get(i).get_exe_name());
}
dynamic.add(process_created_cmd_line);
dynamic.add(process_created_description);
dynamic.add(process_created_exe_name);


//NETWORK ACTIVITY ATTRIBUTES
NetworkActivity network = xml.get_network();

DynamicString socket_close_time = new DynamicString("socket_close_time");
DynamicString socket_create_time = new DynamicString("socket_create_time");
DynamicString socket_created_by_thread = new DynamicString("
     socket_created_by_thread");
DynamicString socket_foreign_ip = new DynamicString("socket_foreign_ip");
DynamicString socket_foreign_port = new DynamicString("socket_foreign_port"
     );
DynamicString socket_is_listening = new DynamicString("socket_is_listening"
     );
DynamicString socket_local_ip = new DynamicString("socket_local_ip");
DynamicString socket_local_port = new DynamicString("socket_local_port");
ArrayList<Socket> socket = network.get_socket();
for(int i = 0; i < socket.size(); i++)
{
        socket_close_time.add_content(socket.get(i).get_close_time());
        socket_create_time.add_content(socket.get(i).get_create_time());
        socket_created_by_thread.add_content(socket.get(i).
             get_created_by_thread());
        socket_foreign_ip.add_content(socket.get(i).get_foreign_ip());
        socket_foreign_port.add_content(socket.get(i).get_foreign_port());
        socket_is_listening.add_content(socket.get(i).get_is_listening());
        socket_local_ip.add_content(socket.get(i).get_local_ip());
        socket_local_port.add_content(socket.get(i).get_local_port());
}
dynamic.add(socket_close_time);
dynamic.add(socket_create_time);
dynamic.add(socket_created_by_thread);
dynamic.add(socket_foreign_ip);
dynamic.add(socket_foreign_port);
dynamic.add(socket_is_listening);
dynamic.add(socket_local_ip);
dynamic.add(socket_local_port);

DynamicString dns_name = new DynamicString("dns_name");
DynamicString dns_result = new DynamicString("dns_result");
DynamicString dns_successfull = new DynamicString("dns_successfull");
DynamicString dns_type = new DynamicString("dns_type");
ArrayList<DNSQuery> dns = network.get_dns_query();
for(int i = 0; i < dns.size(); i++)
{
        dns_name.add_content(dns.get(i).get_name());
        dns_result.add_content(dns.get(i).get_result());
        dns_successfull.add_content(dns.get(i).get_successfull());
        dns_type.add_content(dns.get(i).get_type());
}
dynamic.add(dns_name);
dynamic.add(dns_result);
dynamic.add(dns_successfull);
dynamic.add(dns_type);

DynamicString smtp_content = new DynamicString("smtp_content");
DynamicString smtp_description = new DynamicString("smtp_description");
DynamicString smtp_dest_ip = new DynamicString("smtp_dest_ip");
DynamicString smtp_dest_port = new DynamicString("smtp_dest_port");
DynamicString smtp_recipient = new DynamicString("smtp_recipient");
DynamicString smtp_sender = new DynamicString("smtp_sender");
DynamicString smtp_server_reply = new DynamicString("smtp_server_reply");
DynamicString smtp_src_ip = new DynamicString("smtp_src_ip");
DynamicString smtp_src_port = new DynamicString("smtp_src_port");
DynamicString smtp_subject = new DynamicString("smtp_subject");
ArrayList<SMTPConversation> smtp = network.get_smtp_conversation();
for(int i = 0; i < smtp.size(); i++)
{
        smtp_content.add_content(smtp.get(i).get_content());
        smtp_description.add_content(smtp.get(i).get_description());
        smtp_dest_ip.add_content(smtp.get(i).get_dest_ip());
        smtp_dest_port.add_content(smtp.get(i).get_dest_port());
        smtp_recipient.add_content(smtp.get(i).get_recipient());
        smtp_sender.add_content(smtp.get(i).get_sender());
        smtp_server_reply.add_content(smtp.get(i).get_server_reply());
        smtp_src_ip.add_content(smtp.get(i).get_src_ip());
        smtp_src_port.add_content(smtp.get(i).get_src_port());
        smtp_subject.add_content(smtp.get(i).get_subject());
}
dynamic.add(smtp_content);
dynamic.add(smtp_description);
dynamic.add(smtp_dest_ip);
dynamic.add(smtp_dest_port);
dynamic.add(smtp_recipient);
dynamic.add(smtp_sender);
dynamic.add(smtp_server_reply);
dynamic.add(smtp_src_ip);
dynamic.add(smtp_src_port);
dynamic.add(smtp_subject);

DynamicString http_dest_ip = new DynamicString("http_dest_ip");
DynamicString http_dest_port = new DynamicString("http_dest_port");
DynamicString http_hostname = new DynamicString("http_hostname");
DynamicString http_src_ip = new DynamicString("http_src_ip");
DynamicString http_src_port = new DynamicString("http_src_port");
ArrayList<HTTPConversation> http = network.get_http_conversation();
for(int i = 0; i < http.size(); i++)
{
        http_dest_ip.add_content(http.get(i).get_dest_ip());
        http_dest_port.add_content(http.get(i).get_dest_port());
        http_hostname.add_content(http.get(i).get_hostname());
        http_src_ip.add_content(http.get(i).get_src_ip());
        http_src_port.add_content(http.get(i).get_src_port());
}
dynamic.add(http_dest_ip);
dynamic.add(http_dest_port);
dynamic.add(http_hostname);
dynamic.add(http_src_ip);
dynamic.add(http_src_port);

DynamicString tcp_dest_ip = new DynamicString("tcp_dest_ip");
DynamicString tcp_dest_port = new DynamicString("tcp_dest_port");
DynamicString tcp_org_bytes_sent = new DynamicString("tcp_org_bytes_sent");
DynamicString tcp_res_bytes_sent = new DynamicString("tcp_res_bytes_sent");
DynamicString tcp_src_ip = new DynamicString("tcp_src_ip");
DynamicString tcp_src_port = new DynamicString("tcp_src_port");
DynamicString tcp_state = new DynamicString("tcp_state");
ArrayList<TCPConversation> tcp = network.get_tcp_conversation();
for(int i = 0; i < tcp.size(); i++)
{
```

```
                tcp_dest_ip.add_content(tcp.get(i).get_dest_ip());
                tcp_dest_port.add_content(tcp.get(i).get_dest_port());
                tcp_org_bytes_sent.add_content(tcp.get(i).get_org_bytes_sent());
                tcp_res_bytes_sent.add_content(tcp.get(i).get_res_bytes_sent());
                tcp_src_ip.add_content(tcp.get(i).get_src_ip());
                tcp_src_port.add_content(tcp.get(i).get_src_port());
                tcp_state.add_content(tcp.get(i).get_state());
            }
            dynamic.add(tcp_dest_ip);
            dynamic.add(tcp_dest_port);
            dynamic.add(tcp_org_bytes_sent);
            dynamic.add(tcp_res_bytes_sent);
            dynamic.add(tcp_src_ip);
            dynamic.add(tcp_src_port);
            dynamic.add(tcp_state);

            DynamicString udp_dest_ip = new DynamicString("udp_dest_ip");
            DynamicString udp_dest_port = new DynamicString("udp_dest_port");
            DynamicString udp_org_bytes_sent = new DynamicString("udp_org_bytes_sent");
            DynamicString udp_res_bytes_sent = new DynamicString("udp_rest_bytes_sent")
                  ;
            DynamicString udp_src_ip = new DynamicString("udp_src_ip");
            DynamicString udp_src_port = new DynamicString("udp_src_port");
            DynamicString udp_state = new DynamicString("udp_state");

            ArrayList<UDPConversation> udp = network.get_udp_conversation();
            for(int i = 0; i < udp.size(); i++)
            {
                udp_dest_ip.add_content(udp.get(i).get_dest_ip());
                udp_dest_port.add_content(udp.get(i).get_dest_port());
                udp_org_bytes_sent.add_content(udp.get(i).get_org_bytes_sent());
                udp_res_bytes_sent.add_content(udp.get(i).get_res_bytes_sent());
                udp_src_ip.add_content(udp.get(i).get_src_ip());
                udp_src_port.add_content(udp.get(i).get_src_port());
                udp_state.add_content(udp.get(i).get_state());
            }
            dynamic.add(udp_dest_ip);
            dynamic.add(udp_dest_port);
            dynamic.add(udp_org_bytes_sent);
            dynamic.add(udp_res_bytes_sent);
            dynamic.add(udp_src_ip);
            dynamic.add(udp_src_port);
            dynamic.add(udp_state);
        }

        //return dynamic features
        public ArrayList<DynamicString> get_features()
        {
            return dynamic;
        }

        //extract labels from samples
        public void extract_label()
        {
            if(sample_id != null)
            {
                //go backwards from string name to exclude specific version of
                      malware family
                for(int i = sample_id.length()-1; i > 0; i--)
                {
                    int a = i;
                    int b = i-1;

                    String n = sample_id.substring(b,a);

                    if(n.compareTo(".") == 0)
                    {
                        label = sample_id.substring(0,b);

                        if(label.substring(0,6).compareTo("Benign") == 0)
                        {
                            label_type = "benign";
                        }
                        else
                        {
                            label_type = "malicious";
                        }

                        break;
                    }
            }
        }
        else
        {
            System.out.println("Cannot extract label from sample id!");
        }
    }

    //inner class for holding name and content
    public class DynamicString
    {
        private String name;
        private String content;
        private ArrayList<String> string_list;
        private int ld; //levensthein

        public DynamicString(String name)
        {
            this.name = name;
            this.content = null;
            this.string_list = new ArrayList<String>();
            this.ld = 0;
        }

        public void add_content(String c)
        {
            String m = null;

            if(c != null)
            {
                string_list.add(c);
                m = c;
            }
            else
            {
                m = "";
            }

            if(content == null)
            {
                String n = "\"" + m + "\"";
                content = n;
            }
            else
            {
                String n = " \"" + m + "\"";
                content = content + n;
            }
        }

        public void calc_ld()
        {
            if(string_list.size() == 1)
            {
                //not sure if this is good enough
                ld = string_list.get(0).length();
            }
            else
            {
                ArrayList<Integer> ld_score = new ArrayList<Integer>();

                //calculate all combinations
                for(int i = 0; i < string_list.size(); i++)
                {
                    int score = 0;
                    for(int ii = 0; ii < string_list.size(); ii++)
                        score += LD.computeLevenshteinDistance
                            (string_list.get(i), string_list.
                              get(ii));
                    ld_score.add(score);
                }

                //find lowest score
                int lowest_current = 0;
                for(int i = 0; i < ld_score.size()-1; i++)
                {
                    lowest_current = ld_score.get(i);
                    int next = ld_score.get(i+1);

                    if(lowest_current > next)
```

```
                                                lowest_current = next;
                                }

                                //set lowest score
                                ld = lowest_current;
                        }
                }

                public String get_name(){return name;}
                public String get_content(){return content;}
                public ArrayList<String> get_string_list(){return string_list;}
                public int get_ld(){return ld;}
        }

        @Deprecated //testing levensthein-distance
        public static class LD
        {
                private static int minimum(int a, int b, int c)
                {
                        return Math.min(Math.min(a, b), c);
            }

            public static int computeLevenshteinDistance(CharSequence str1, CharSequence
                 str2)
            {
                int[][] distance = new int[str1.length() + 1][str2.length() + 1];

                for (int i = 0; i <= str1.length(); i++)
                        distance[i][0] = i;
                for (int j = 0; j <= str2.length(); j++)
                    distance[0][j] = j;

                for (int i = 1; i <= str1.length(); i++)
                        for (int j = 1; j <= str2.length(); j++)
                                distance[i][j] = minimum(
                                        distance[i - 1][j] + 1,
                            distance[i][j - 1] + 1,
                            distance[i - 1][j - 1]
                                + ((str1.charAt(i - 1) == str2.charAt(j - 1)) ? 0
                                                : 1));

                return distance[str1.length()][str2.length()];
            }
        }

}
```

FeatureStatic

```
package feature.extractor;

import java.util.ArrayList;

//class FeatureStatic holds the created static feature set
public class FeatureStatic
{
        private String sample_id;
        //used when applying > 2 labels
        private String label;
        //used when applying malicious vs benign
        private String label_type;
        private ArrayList<StaticDLL> static_dll;

        public FeatureStatic()
        {
                static_dll = new ArrayList<StaticDLL>();
        }

        //returns static dll
        public ArrayList<StaticDLL> get_static_dll()
        {
                return static_dll;
        }

        //returns sample id
        public String get_sample_id()
        {
                return sample_id;
        }
        //return label
        public String get_label()
        {
                return label;
        }

        //return label type
        public String get_label_type()
        {
                return label_type;
        }

        //extracting label from sample
        public void extract_label()
        {
                if(sample_id != null)
                {
                        //go backwards from string name to exclude specific version of
                             malware family
                        for(int i = sample_id.length()-1; i > 0; i--)
                        {
                                int a = i;
                                int b = i-1;

                                String n = sample_id.substring(b,a);

                                if(n.compareTo(".") == 0)
                                {
                                        label = sample_id.substring(0,b);

                                        if(label.substring(0,6).compareTo("Benign") == 0)
                                        {
                                                label_type = "benign";
                                        }
                                        else
                                        {
                                                label_type = "malicious";
                                        }

                                        break;
                                }
                        }
                }
                else
                {
                        System.out.println("Cannot extract label from sample id!");
                }
        }

        //method adds static features
        public void add_static_feature(PE pe)
        {
                sample_id = pe.get_sample_id();

                //find all DLL names
                ArrayList<DLL> dlls = pe.get_dll();

                String current_dll = null;
                String functions = "";
                String name_functions = "";
                String q = "\"";

                int counter = 0;

                if(!dlls.isEmpty())
                {
                        current_dll = dlls.get(0).get_dll_type();
                        if(dlls.get(0).get_dll_function().compareTo("unknown")!= 0)
                        {
                                //adding only function call
                                functions = q+dlls.get(0).get_dll_function()+q;
                                name_functions = q+dlls.get(0).get_dll_name_function()+q;
                        }
                }

                //count same function and build functions strings
                for(int i = 1; i < dlls.size(); i++)
                {
                        String next_dll = dlls.get(i).get_dll_type();
        if(current_dll.compareTo(next_dll)==0)
        {
            if(dlls.get(i).get_dll_function().compareTo("unknown")!= 0)
            {
                counter++;
                if(functions.compareTo("")==0)
                {
                    //adding functions to a long string
                    String itemp = q+dlls.get(i).
                         get_dll_function()+q;
                    String ftemp = functions+itemp;
                    functions = ftemp;

                    //adding function to the other string
                    String jtemp = q+dlls.get(i).
                         get_dll_name_function()+q;
                    String gtemp = name_functions + jtemp;
                    name_functions = gtemp;
                }
                else
                {
                    //adding functions to a long string
                    String itemp = " "+q+dlls.get(i).
                         get_dll_function()+q;
                    String ftemp = functions+itemp;
                    functions = ftemp;

                    //adding function to the other string
                    String jtemp = " "+q+dlls.get(i).
                         get_dll_name_function()+q;
                    String gtemp = name_functions + jtemp;
                    name_functions = gtemp;
                }
            }
        }
        else
        {
            //add dll
            StaticDLL dll = new StaticDLL();
            dll.set_dll_name(current_dll);
            dll.set_count(counter);
            dll.set_dll_functions(functions);
            dll.set_dll_name_functions(name_functions);

            //add to array
            static_dll.add(dll);

            //reset counter
            counter = 0;
            functions = ""; //or '?'
            name_functions = "";
            //change current string
            current_dll = next_dll;
        }
    }
}

//inner class
public class StaticDLL
{
    private String dll_name;
    private int    count;
    private String dll_functions;
    private String dll_name_functions;

    public void print_info()
    {
        System.out.println(dll_name + " -> " + count + " -> " +
             dll_functions);
    }

    public void set_dll_name(String dll_name){this.dll_name = dll_name.
         toLowerCase();}
    public void set_count(int count){this.count = count;}
    public void set_dll_functions(String dll_functions){this.dll_functions =
         dll_functions;}
    public void set_dll_name_functions(String n){this.dll_name_functions = n;}

    public String get_dll_name(){return dll_name;}
    public int get_count(){return count;}
```

```
                public String get_dll_functions(){return dll_functions;}
                public String get_dll_name_functions(){return dll_name_functions;}
        }
}
```

FeatureCombo

```
package feature.extractor;

import java.util.ArrayList;

import feature.extractor.FeatureDynamic.DynamicString;
import feature.extractor.FeatureStatic.StaticDLL;

//class FeatureCombo builds a feature set based
//on static and dynamic feature set
public class FeatureCombo
{
        private String sample_id;
        //used when applying > 2 labels
        private String label;
        //used when applying malicious vs benign
        private String label_type;
        private ArrayList<StaticDLL> static_dll;
        private ArrayList<DynamicString> dynamic_string;

        public FeatureCombo(ArrayList<StaticDLL> static_dll, ArrayList<DynamicString>
            dynamic_string, String sample_id)
        {
                this.static_dll = static_dll;
                this.dynamic_string = dynamic_string;
                this.sample_id = sample_id;
        }

        //extracting label
        public void extract_label()
        {
                if(sample_id != null)
                {
                        //go backwards from string name to exclude specific version of
                            malware family
                        for(int i = sample_id.length()-1; i > 0; i--)
                        {
                                int a = i;
                                int b = i-1;

                                String n = sample_id.substring(b,a);

                                if(n.compareTo(".") == 0)
                                {
                                        label = sample_id.substring(0,b);

                                        if(label.substring(0,6).compareTo("Benign") == 0)
                                        {
                                                label_type = "benign";
                                        }
                                        else
                                        {
                                                label_type = "malicious";
                                        }

                                        break;
                                }
                        }
                }
                else
                {
                        System.out.println("Cannot extract label from sample id!");
                }
        }

        //return sample id
        public String get_sample_id()
        {
                return sample_id;
        }

        //return label
        public String get_label()
        {
                return label;
        }

        //return label type
        public String get_label_type()
        {
                return label_type;
        }

        //return static features
        public ArrayList<StaticDLL> get_static_dll()
        {
                return static_dll;
        }

        //return dynamic features
        public ArrayList<DynamicString> get_dynamic_string()
        {
                return dynamic_string;
        }
}
```

ARFF

```
package feature.extractor;

import feature.extractor.FeatureDynamic.DynamicString;
import feature.extractor.FeatureStatic.StaticDLL;
//imported std java libraries
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
//imported weka libraries
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;
import weka.filters.unsupervised.attribute.StringToWordVector;
import weka.core.converters.ArffSaver;

//ARFF class generates 3 ARFF files with different attribute types
//reflecting static, dynamic and a combined feature set
public class ARFF
{
        //holds data that is going to written to file
        private Instances data;
        private ArrayList<FeatureStatic> fstatic;
        private ArrayList<FeatureDynamic> fdynamic;
        private ArrayList<FeatureCombo> fcombo;

        public ARFF()
        {
                //empty constructor
        }

        //private methods checks if arraylist contain certain string
        private boolean contains_string(ArrayList<String> list, String name)
        {
                for(int i = 0; i < list.size(); i++)
                {
                        String s = list.get(i);
                        //if list contain string
                        if(s.compareTo(name)==0)
                        {
                                return true;
                        }
                }
                return false;
        }

        //private method returns a index to input string search
        private int find_string(ArrayList<StaticDLL> list, String name)
        {
                for(int i = 0; i < list.size(); i++)
                {
                        String s = list.get(i).get_dll_name();
                        //if list contain string
                if(s.compareTo(name)==0)
                {
                        return i;
                }
        }
        return -1;
}

//method creates an arff file based on static features
public void generate_arff_static(ArrayList<FeatureStatic> feature_static) throws
     Exception
{
        //local variables
        double[] values;
        double[] values_label;
        Instances label_data;
        fstatic = feature_static;

        //add all dlls used by all instances in a list
        ArrayList<String> dll_list = new ArrayList<String>();
        for(int i = 0; i < fstatic.size(); i++)
        {
                ArrayList<StaticDLL> c_dll = feature_static.get(i).get_static_dll()
                     ;

                for(int ii = 0; ii < c_dll.size(); ii++)
                {
                        //add string if it does not exist in list
                        String c_string = c_dll.get(ii).get_dll_name();
                        if(!contains_string(dll_list,c_string))
                        {
                                dll_list.add(c_string);
                        }
                }
        }

        //add attributes for used dlls - true or false (nominal)
        FastVector attributes = new FastVector();
        FastVector att_true_false = new FastVector();
        att_true_false.addElement("true");
        att_true_false.addElement("false");

        //must create attributes for labels
        FastVector att_labels = new FastVector();
        FastVector label_attributes = new FastVector();
        ArrayList<String> slabels = new ArrayList<String>();
        //add dummy
        //att_labels.addElement("dummy");
        for(int i = 0; i < fstatic.size(); i++)
        {
                String l = fstatic.get(i).get_label_type();
                if(!contains_string(slabels,l))
                {
                        att_labels.addElement(l);
                        slabels.add(l);
                }
        }

        //add all used attributes in the attributes-object
        for(int i = 0; i < dll_list.size(); i++)
        {
                String dll_name = dll_list.get(i);
                String dll_functions = dll_name + "_functions";
                //add attribute DLL name
                attributes.addElement(new Attribute(dll_name,att_true_false));
                //add a string with function names
                attributes.addElement(new Attribute(dll_functions,(FastVector) null
                     ));
        }

        //add attribute sample label
        label_attributes.addElement(new Attribute("sample_label",att_labels));

        //add instance object
        data = new Instances("BehaviorSimilarity",attributes,0);
        label_data = new Instances("BehaviorSimilarity",label_attributes,0);

        //iterate through the dll list
        for(int i = 0; i < fstatic.size(); i++)
        {
            values = new double[data.numAttributes()];
            values_label = new double[label_data.numAttributes()];

            ArrayList<StaticDLL> static_dlls = fstatic.get(i).get_static_dll();
            int counter = 0;

            //iterate feature static objects
            for(int ii = 0; ii < dll_list.size(); ii++)
            {
                String c_string = dll_list.get(ii);
                int found = find_string(static_dlls,c_string);

                //if DLL is used in the static report
                if(found != -1)
                {
                    int index = found;
                    values[counter] = att_true_false.indexOf("true");
                    //int dll_calls = static_dlls.get(index).get_count
                        ();
                    String functions = static_dlls.get(index).
                        get_dll_functions();

                    if(functions.compareTo("")==0)
                    {
                        values[counter+1] = Instance.missingValue()
                            ;
                    }
                    else
                    {
                        values[counter+1] = data.attribute(counter
                            +1).addStringValue(functions);
                    }
                }
                //else add missing values
                else
                {
                    values[counter] = att_true_false.indexOf("false");
                    values[counter+1] = Instance.missingValue();
                }
                counter += 2;
            }
            //add values in instance object
            values_label[label_data.numAttributes()-1] = att_labels.indexOf(
                fstatic.get(i).get_label_type());

            Instance inst1 = new Instance(1.0, values);
            Instance inst2 = new Instance(1.0, values_label);

            data.add(inst1);
            label_data.add(inst2);
        }
        //split strings
        data = get_new_instances(data);
        //remove values without correlation
        data = correlation_filer(data, ".\\cf_vectors\\cf_static.txt");
        //merge instances so that sample_label is the last attribute
        Instances test = Instances.mergeInstances(data, label_data);
        write_to_file( test , ".\\arff_out\\out_static.arff" );
}

//method creates an arff file based on dynamic features
public void generate_arff_dynamic(ArrayList<FeatureDynamic> feature_dynamic) throws
        Exception
{
        //local variables
        fdynamic = feature_dynamic;
        double[] values;
        double[] values_label;
        Instances label_data;

        //add all entities used by all instances are added in a list
        ArrayList<String> entity_list = new ArrayList<String >();
        //since all entries are similar for all objects it will do with the first
            index
        ArrayList<DynamicString> dstring = fdynamic.get(0).get_features();
        for(int ii = 0; ii < dstring.size(); ii++)
        {
            //add entity names to string
            String c_string = dstring.get(ii).get_name();
            entity_list.add(c_string);
        }

        FastVector attributes = new FastVector();

        //must create attributes for labels
        FastVector att_labels = new FastVector();
        FastVector label_attributes = new FastVector();
        ArrayList<String> slabels = new ArrayList<String>();
        //must add dummy
        //att_labels.addElement("dummy");
        for(int i = 0; i < fdynamic.size(); i++)
        {
            String l = fdynamic.get(i).get_label_type();
            if(!contains_string(slabels,l))
            {
                att_labels.addElement(l);
                slabels.add(l);
            }
        }
        //add attribute sample label
        label_attributes.addElement(new Attribute("sample_label",att_labels));

        //adding attributes in fastvector
        for(int i = 0; i < entity_list.size(); i++)
        {
            String entity_name = entity_list.get(i);
            //add attributes
            attributes.addElement(new Attribute(entity_name,(FastVector) null))
                ;
        }

        //add instance object
        data = new Instances("BehaviorSimilarity",attributes,0);
        label_data = new Instances("BehaviorSimilarity",label_attributes,0);

        //iterate through features
        for(int i = 0; i < fdynamic.size(); i++)
        {
            values = new double[data.numAttributes()];
            values_label = new double[label_data.numAttributes()];

            ArrayList<DynamicString> ds = fdynamic.get(i).get_features();

            for(int ii = 0; ii < ds.size(); ii++)
            {
                //add feature contents
                String n = ds.get(ii).get_content();

                if(n == null)
                {
                    //when string is null insert missing value
                    values[ii] = Instance.missingValue();
                }
                else
                {
                    //else insert content
                    values[ii] = data.attribute(ii).addStringValue(n);
                }
            }
            //adding sample_label
            values_label[label_data.numAttributes()−1] = att_labels.indexOf(
                fdynamic.get(i).get_label_type());

            Instance inst1 = new Instance(1.0, values);
            Instance inst2 = new Instance(1.0, values_label);

            data.add(inst1);
            label_data.add(inst2);
        }
        //split strings
        data = get_new_instances(data);
        //remove values without correlation
        data = correlation_filer(data, ".\\cf_vectors\\cf_dynamic.txt");
        //merge instances so that sample_label is the last attribute
        Instances test = Instances.mergeInstances(data, label_data);
        //write to file
        write_to_file( test, ".\\arff_out\\out_dynamic.arff");
    }

    //method creates an arff file based on combined features
```

```
public void generate_arff_combo(ArrayList<FeatureCombo> feature_combo) throws
     Exception
{
        fcombo = feature_combo;

        //need two arrays for holding values
        ArrayList<String> dll_list = new ArrayList<String>();
        ArrayList<String> entity_list = new ArrayList<String>();

        for(int i = 0; i < fcombo.size(); i++)
        {
                //adding static features
                ArrayList<StaticDLL> c_dll = fcombo.get(i).get_static_dll();
                for(int ii = 0; ii < c_dll.size(); ii++)
                {
                        //add string if it does not exist in list
                        String c_string = c_dll.get(ii).get_dll_name();
                        if(!contains_string(dll_list,c_string))
                        {
                                dll_list.add(c_string);
                        }
                }
        }

        //adding dynamic features
        ArrayList<DynamicString> dstring = fcombo.get(0).get_dynamic_string();
        for(int ii = 0; ii < dstring.size(); ii++)
        {
                //add entity names to string
                String c_string = dstring.get(ii).get_name();
                entity_list.add(c_string);
        }

        //make attribute list
        FastVector attributes = new FastVector();
        FastVector label_attributes = new FastVector();

        //must create attributes for labels
        FastVector att_labels = new FastVector();
        ArrayList<String> slabels = new ArrayList<String>();
        //must add dummy
        //att_labels.addElement("dummy");
        for(int i = 0; i < fcombo.size(); i++)
        {
                String l = fcombo.get(i).get_label_type();

                if(!contains_string(slabels,l))
                {
                        att_labels.addElement(l);
                        slabels.add(l);
                }
        }
        //add attribute sample label
        label_attributes.addElement(new Attribute("sample_label",att_labels));

        //adding static attributes in fastvector
        for(int i = 0; i < dll_list.size(); i++)
        {
                String dll_name = dll_list.get(i);
                String dll_functions = dll_name + "_functions";
                //add a string with function names
                attributes.addElement(new Attribute(dll_functions,(FastVector) null
                     ));
        }

        //adding dynamic attributes in fastvector
        for(int i = 0; i < entity_list.size(); i++)
        {
                String entity_name = entity_list.get(i);
                //add attributes
                attributes.addElement(new Attribute(entity_name,(FastVector) null))
                     ;
        }

        //add values
        data = new Instances("BehaviorSimilarity",attributes,0);
        Instances label_data = new Instances("BehaviorSimilarity",label_attributes
             ,0);

        double values[];
        double[] values_label;

        //iterate through features
        for(int i = 0; i < fcombo.size(); i++)
        {
                values = new double[data.numAttributes()];
                values_label = new double[label_data.numAttributes()];

                ArrayList<DynamicString> ds = fcombo.get(i).get_dynamic_string();
                ArrayList<StaticDLL> sd = fcombo.get(i).get_static_dll();

                //resetting counter
                int counter = 0;

                //adding static features
                for(int ii = 0; ii < dll_list.size(); ii++)
                {
                        String c_string = dll_list.get(ii);
                        int found = find_string(sd,c_string);

                        //if DLL is used in the static report
                        if(found != -1)
                        {
                                int index = found;
                                String name_functions = sd.get(index).
                                     get_dll_name_functions();

                                //String name_functions = sd.get(index).
                                     get_dll_name_functions();
                                if(name_functions.compareTo("")==0) values[counter]
                                      = Instance.missingValue();
                                else values[counter] = data.attribute(counter).
                                     addStringValue(name_functions);
                        }
                        else
                        {
                                //else inserting missing values
                                values[counter] = Instance.missingValue();
                        }
                        counter++;
                }

                //adding dynamic features
                for(int ii = 0; ii < ds.size(); ii++)
                {
                        //add feature contents
                        String n = ds.get(ii).get_content();

                        if(n == null)
                        {
                                //when string is null insert missing value
                                values[counter] = Instance.missingValue();
                        }
                        else
                        {
                                //else insert content
                                values[counter] = data.attribute(counter).
                                     addStringValue(n);
                        }
                        counter++;
                }
                //inserting attributes objects into instance objects
                values_label[label_data.numAttributes()-1] = att_labels.indexOf(
                    fcombo.get(i).get_label_type());

                Instance inst1 = new Instance(1.0, values);
                Instance inst2 = new Instance(1.0, values_label);

                data.add(inst1);
                label_data.add(inst2);
        }
        //split strings
        data = get_new_instances(data);
        //remove values without correlation
        data = correlation_filer(data, ".\\cf_vectors\\cf_combo.txt");
        //merge instances in order to get label as last attribute
        Instances test = Instances.mergeInstances(data, label_data);
        //write arff-file
        write_to_file( test , ".\\arff_out\\out_combo.arff");
}
```

```java
//private method to save arff file
private void write_to_file(Instances data, String path) throws IOException
{
        ArffSaver saver = new ArffSaver();
        saver.setInstances(data);

        saver.setFile(new File(path));
        saver.writeBatch();

System.out.println("Stored result at " + path);
}

//private method for splitting strings with StringToWordVector
private Instances get_new_instances(Instances data) throws Exception
{
        StringToWordVector stwv = new StringToWordVector(10000);
        stwv.setInputFormat(data);
        String[] options = new String[14];

        //parameters
        options[0] = "-R";
        options[1] = "first-last";
        options[2] = "-W";
        options[3] = "10000";
        options[4] = "-prune-rate";
        options[5] = "-1.0";
        options[6] = "-N";
        options[7] = "0";
        options[8] = "-stemmer";
        options[9] = "weka.core.stemmers.NullStemmer";
        options[10] = "-M";
        options[11] = "1";
        options[12] = "-tokenizer";
        options[13] = "weka.core.tokenizers.WordTokenizer -delimiters \" \\r \\t
             \\\"\"";

        stwv.setOptions(options);
        Instances new_data = Filter.useFilter(data, stwv);

        return new_data;
}

//method
private Instances correlation_filer(Instances inst, String path) throws Exception
{
        ArrayList<Double> values = open_correlation_vector(path);
        ArrayList<Integer> indeces = new ArrayList<Integer>();

        Remove remove = new Remove();

        //iterate through correlation vector and store indeces for '0' values
        int count = 1;
        for(int i = 0; i < values.size(); i++)
        {
                if(values.get(i) < 0.1)
                {
                        indeces.add(i);
                        count++;
                }
        }
        System.out.println("Values that can be removed " + count);

        //convert arraylist into array
        int[] filter_array = new int[indeces.size()];
        for(int i = 0; i < filter_array.length; i++)
        {
                filter_array[i] = indeces.get(i);
        }
        //System.out.println("Double check " + filter_array.length);

        //remove
        remove.setAttributeIndicesArray(filter_array);
        remove.setInputFormat(inst);
        Instances new_data = Filter.useFilter(inst, remove);
        return new_data;
}

//mehod opens corrolation vectors
private ArrayList<Double> open_correlation_vector(String path) throws Exception
```

```
{
        //loading text file from path
        Scanner scanner = new Scanner(new File(path));
        ArrayList<Double> values = new ArrayList<Double>();
        while(scanner.hasNext())
        {
                double d = Double.parseDouble(scanner.nextLine());
                values.add(d);
        }
        return values;
}

@Deprecated //testing levenstein-distance
public void generate_arff_ld(ArrayList<FeatureDynamic> feature_dynamic) throws
     IOException
{
        fdynamic = feature_dynamic;
        double[] values;

        //add all entities used by all instances are added in a list
        ArrayList<String> entity_list = new ArrayList<String>();
        //since all entries are similar for all objects it will do with the first
             index
        ArrayList<DynamicString> dstring = fdynamic.get(0).get_features();
        for(int ii = 0; ii < dstring.size(); ii++)
        {
                //add entity names to string
                String c_string = dstring.get(ii).get_name();
                entity_list.add(c_string);
        }

        FastVector attributes = new FastVector();

        //must create attributes for labels
        FastVector att_labels = new FastVector();
        ArrayList<String> slabels = new ArrayList<String>();
        for(int i = 0; i < fdynamic.size(); i++)
        {
                String l = fdynamic.get(i).get_label_type();
                if(!contains_string(slabels,l))
                {
                        att_labels.addElement(l);
                        slabels.add(l);
                }
        }
        //add attribute sample label
        attributes.addElement(new Attribute("sample_label",att_labels));

        //adding attributes in fastvector
        for(int i = 0; i < entity_list.size(); i++)
        {
                String entity_name = entity_list.get(i);
                //add attributes -> modified for integers
                attributes.addElement(new Attribute(entity_name));
        }

        //add instance object
        data = new Instances("BehaviorSimilarity",attributes,0);

        //iterate through features
        for(int i = 0; i < fdynamic.size(); i++)
        {
                values = new double[data.numAttributes()];
                ArrayList<DynamicString> ds = fdynamic.get(i).get_features();

                //adding sample_id value
                values[0] = att_labels.indexOf(fdynamic.get(i).get_label_type());

                for(int ii = 0; ii < ds.size(); ii++)
                {
                        //add feature contents
                        int n = ds.get(ii).get_ld();

                        if(n == 0)
                        {
                                //when string is null insert missing value
                                values[ii+1] = Instance.missingValue();
                        }
                        else
                        {
```

```
                                        //else insert content
                                        values[ii+1] = n;
                                }
                        }
                        data.add(new Instance(1.0, values));
                }
                write_to_file(data,".\\arff_out\\out_ld.arff");
        }
}
```

E.3 Evaluator

Main

```
package evaluation;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;

import evaluation.OpenARFF.DynamicInstances;

//main class for evaluation components
//opens arff files and runs 5 classifiers
public class Main
{
        public static void main(String[] args)
        {
                try
                {
                        //open ARFF files
                        OpenARFF open = new OpenARFF();
                        //get instances
                        ArrayList<DynamicInstances> dyn_inst = open.get_instances();
                        //ArrayList<DynamicInstances> dyn_inst = open.merge_instances();
                        //evaluate
                        Evaluator eval = new Evaluator(dyn_inst);
                        eval.EvaluateClassifiers();
                }
                catch (Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

Evaluator

```
package evaluation;

import java.awt.*;
import java.io.*;
import java.util.*;
import javax.swing.*;

import evaluation.OpenARFF.DynamicInstances;
import weka.core.*;
import weka.classifiers.*;

import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;
import weka.classifiers.functions.LibSVM;

import weka.classifiers.evaluation.*;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Discretize;
import weka.gui.visualize.*;

//class Evaluator runs 5 classifiers on the loaded feature sets
public class Evaluator
{
        private ArrayList<DynamicInstances> dyn_inst;

        public Evaluator(ArrayList<DynamicInstances> dyn_inst)
        {
                this.dyn_inst = dyn_inst;
        }

        //method evaluates the classifiers
        public void EvaluateClassifiers() throws Exception
        {
                for(int i = 0; i < dyn_inst.size(); i++)
                {
                        //get instances
                        Instances inst = dyn_inst.get(i).get_instance();
                        inst.setClassIndex(dyn_inst.get(i).get_instance().numAttributes()
                              −1);
                //classification with Naive Bayes
                System.out.println(dyn_inst.get(i).get_name() + " with Naive Bayes"
                     );
                Evaluation eval = NaiveBayesEvaluation(inst);
                print_results(eval);
                eval = null;

                //classification with K–NN
                System.out.println(dyn_inst.get(i).get_name() + " with K–NN");
                eval = KNNEvaluation(inst);
                print_results(eval);
                eval = null;

                //classification with C4.5
                System.out.println(dyn_inst.get(i).get_name() + " with C4.5");
                eval = C45Evaluation(inst);
                print_results(eval);
                eval = null;

                //classification with SVM
                System.out.println(dyn_inst.get(i).get_name() + " with SVM");
                eval = SVMEvaluation(inst);
                print_results(eval);
                eval = null;

                //classification with Bayes Network
                System.out.println(dyn_inst.get(i).get_name() + " with Bayes
                     Network");
                eval = BayesNetworkEvaluation(inst);
                print_results(eval);
                eval = null;
        }
}

//method for Naive Bayes
private Evaluation NaiveBayesEvaluation(Instances inst) throws Exception
{
        // train classifier
    Classifier cl = new NaiveBayes();
    Evaluation eval = new Evaluation(inst);
    eval.crossValidateModel(cl, inst, 10, new Random(1));

    return eval;
}

//method for Bayes Network
private Evaluation BayesNetworkEvaluation(Instances inst) throws Exception
{
        //discretize
        inst = get_unsupervised_instances(inst);

        // train classifier
    Classifier cl = new BayesNet();

    String[] options = new String[13];
    //parameters
    options[0] = "–D";
    options[1] = "–Q";
    options[2] = "weka.classifiers.bayes.net.search.local.K2";
    options[3] = "—";
    options[4] = "–P";
    options[5] = "1";
    options[6] = "–S";
    options[7] = "BAYES";
    options[8] = "–E";
    options[9] = "weka.classifiers.bayes.net.estimate.SimpleEstimator";
    options[10] = "—";
    options[11] = "–A";
    options[12] = "0.5";

    cl.setOptions(options);

    Evaluation eval = new Evaluation(inst);
    eval.crossValidateModel(cl, inst, 10, new Random(1));

    return eval;
}

//method for K–NN
private Evaluation KNNEvaluation(Instances inst) throws Exception
```

```
{
        // train classifier
    Classifier cl = new IBk();

    String[] options = new String[6];
    //parameters
    options[0] = "—K";
    options[1] = "1";
    options[2] = "—W";
    options[3] = "0";
    options[4] = "—A";
    options[5] = "weka.core.neighboursearch.LinearNNSearch —A \"weka.core.
         EuclideanDistance —R first—last\"";

    cl.setOptions(options);

    Evaluation eval = new Evaluation(inst);
    eval.crossValidateModel(cl, inst, 10, new Random(1));

    return eval;
}

//method for C4.5
private Evaluation C45Evaluation(Instances inst) throws Exception
{
        // train classifier
    Classifier cl = new J48();

    String[] options = new String[4];
    //parameters
    options[0] = "—C";
    options[1] = "0.25";
    options[2] = "—M";
    options[3] = "2";

    cl.setOptions(options);

    Evaluation eval = new Evaluation(inst);
    eval.crossValidateModel(cl, inst, 10, new Random(1));

    return eval;
}

//method for SVM
private Evaluation SVMEvaluation(Instances inst) throws Exception
{
        // train classifier
    Classifier cl = new LibSVM();

    String[] options = new String[20];
    //parameters
    options[0] = "—S";
    options[1] = "0";
    options[2] = "—K";
    options[3] = "2";
    options[4] = "—D";
    options[5] = "3";
    options[6] = "—G";
    options[7] = "0.0";
    options[8] = "—R";
    options[9] = "0.0";
    options[10] = "—N";
    options[11] = "0.5";
    options[12] = "—M";
    options[13] = "40.0";
    options[14] = "—C";
    options[15] = "1.0";
    options[16] = "—E";
    options[17] = "0.0010";
    options[18] = "—P";
    options[19] = "0.1";

    cl.setOptions(options);

    Evaluation eval = new Evaluation(inst);
    eval.crossValidateModel(cl, inst, 10, new Random(1));

    return eval;
}
```

```java
//method for unsupervised discretization
private Instances get_unsupervised_instances(Instances data) throws Exception
{
        Discretize d = new Discretize();
        d.setInputFormat(data);

        String[] options = new String[6];
        //parameters
        options[0] = "-B";
        options[1] = "10";
        options[2] = "-M";
        options[3] = "-1.0";
        options[4] = "-R";
        options[5] = "first-last";

        d.setOptions(options);

        Instances new_data = Filter.useFilter(data, d);

        return new_data;
}

//method for printing the evaluation results
private void print_results(Evaluation eval)
{
        double[][] n = eval.confusionMatrix();

    double TP = n[0][0];
    double FN = n[0][1];
    double FP = n[1][0];
    double TN = n[1][1];

    System.out.println("True positives: " + TP + ", True positive rate: " + eval.
         weightedTruePositiveRate());
    System.out.println("False negatives: " + FN + ", False negative rate: " + eval.
         weightedFalseNegativeRate());
    System.out.println("False positives: " + FP + ", False positive rate: " + eval.
         weightedFalsePositiveRate());
    System.out.println("True negatives: " + TN + ", True negative rate: " + eval.
         weightedTrueNegativeRate());

    System.out.println("Detection rate: " + TP/(TP+FN) + ", Accuracy: " + (TP+TN)/(
         TP+TN+FP+FN));
    System.out.println("Precision: " + eval.weightedPrecision() + ", Recall: " +
         eval.weightedRecall());
    System.out.println();
}

//method for showing a roc curve for a given classifier
public void get_ROC(Evaluation eval, int class_index) throws Exception
{
        // generate curve
        ThresholdCurve tc = new ThresholdCurve();
        int classIndex = 0;
        Instances result = tc.getCurve(eval.predictions(), classIndex);

        // plot curve
        ThresholdVisualizePanel vmc = new ThresholdVisualizePanel();
        vmc.setROCString("(Area under ROC = " +
        Utils.doubleToString(tc.getROCArea(result), 4) + ")");
        vmc.setName(result.relationName());
        PlotData2D tempd = new PlotData2D(result);
        tempd.setPlotName(result.relationName());
        tempd.addInstanceNumberAttribute();
        // specify which points are connected
        boolean[] cp = new boolean[result.numInstances()];
        for (int n = 1; n < cp.length; n++)
                cp[n] = true;

        tempd.setConnectPoints(cp);
        // add plot
        vmc.addPlot(tempd);

        // display curve
        String plotName = vmc.getName();
        final javax.swing.JFrame jf =
        new javax.swing.JFrame("Weka Classifier Visualize: "+plotName);
        jf.setSize(500,400);
        jf.getContentPane().setLayout(new BorderLayout());
        jf.getContentPane().add(vmc, BorderLayout.CENTER);
```

```
                jf.addWindowListener(new java.awt.event.WindowAdapter()
                {
                        public void windowClosing(java.awt.event.WindowEvent e)
                        {
                                jf.dispose();
                        }
                });
                jf.setVisible(true);
        }
}
```

OpenARFF

```
package evaluation;

import java.io.*;
import java.util.ArrayList;

import weka.core.Instances;
import weka.core.converters.ArffSaver;

//class OpenARFF opens arff files from a specified directory
public class OpenARFF
{
        private static final String ARFF_PATH = ".\\arff_out\\";
        private static final String ARFF_MERGE_PATH = ".\\merge\\";

        public OpenARFF()
        {
                //empty constructor
        }

        //method opens arff files and store the instances
        public ArrayList<DynamicInstances> get_instances() throws FileNotFoundException,
             IOException
        {
                ArrayList<DynamicInstances> instances = new ArrayList<DynamicInstances>();

                File data_folder = new File(ARFF_PATH);
                File[] contents = data_folder.listFiles();

                for (int i = 0; i < contents.length; i++)
                {
                      if (contents[i].isFile())
                      {
                          String name = contents[i].getName();

                          Instances inst = new Instances(
                                          new BufferedReader(
                                                          new FileReader(ARFF_PATH + name))
                                                               );

                          instances.add(new DynamicInstances(inst,name));
                      }
                }
                return instances;
        }

        //method opens arff files and store the instances
        public ArrayList<DynamicInstances> merge_instances() throws FileNotFoundException,
             IOException
        {
                ArrayList<DynamicInstances> instances = new ArrayList<DynamicInstances>();

                File data_folder = new File(ARFF_MERGE_PATH);
                File[] contents = data_folder.listFiles();

                //first ARFF
                String name = contents[0].getName();
                Instances inst = new Instances(
                                new BufferedReader(
                                                new FileReader(ARFF_MERGE_PATH + name)));

                //second ARFF
                String name2 = contents[1].getName();
                Instances inst2 = new Instances(
                                new BufferedReader(
                                                new FileReader(ARFF_MERGE_PATH + name2)));

                Instances merged = Instances.mergeInstances(inst2, inst);
                instances.add(new DynamicInstances(merged,"out_combined_merged"));

                //store arff
                ArffSaver saver = new ArffSaver();
                saver.setInstances(merged);

                saver.setFile(new File(".\\out_stat_dyn_merged.arff"));
                saver.writeBatch();

                return instances;
        }

        //inner class for holding info
        public class DynamicInstances
        {
                private String name;
                private Instances inst;

                public DynamicInstances(Instances inst, String name)
                {
                        this.inst = inst;
                        this.name = name;
                }

                public Instances get_instance()
                {
                        return inst;
                }

                public String get_name()
                {
                        return name;
                }
        }
}
```