

Analysing Web-based Malware Behaviour through Client Honey pots

**A thesis submitted in partial fulfilment of the requirement for the degree of
Doctor of Philosophy**

Yaser Alosefer

February 2012

**Cardiff University
School of Computer Science & Informatics**

Declaration

This work has not been submitted in substance for any other degree or award at this or any other university or place of learning, nor is being submitted concurrently in candidature for any degree or other award.

Signed (candidate)

Date

Statement 1

This thesis is being submitted in partial fulfilment of the requirements for the degree of PhD.

Signed (candidate)

Date

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references. The views expressed are my own.

Signed (candidate)

Date

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed (candidate)

Date

Abstract

With an increase in the use of the internet, there has been a rise in the number of attacks on servers. These attacks can be successfully defended against using security technologies such as firewalls, IDS and anti-virus software, so attackers have developed new methods to spread their malicious code by using web pages, which can affect many more victims than the traditional approach. The attackers now use these websites to threaten users without the user's knowledge or permission. The defence against such websites is less effective than traditional security products meaning the attackers have the advantage of being able to target a greater number of users.

Malicious web pages attack users through their web browsers and the attack can occur even if the user only visits the web page; this type of attack is called a drive-by download attack. This dissertation explores how web-based attacks work and how users can be protected from this type of attack based on the behaviour of a remote web server. We propose a system that is based on the use of client Honeypot technology. The client Honeypot is able to scan malicious web pages based on their behaviour and can therefore work as an anomaly detection system. The proposed system has three main models: state machine, clustering and prediction models. All these three models work together in order to protect users from known and unknown web-based attacks.

This research demonstrates the challenges faced by end users and how the attacker can easily target systems using drive-by download attacks. In this dissertation we discuss how the proposed system works and the research challenges that we are trying to solve, such as how to group web-based attacks into behaviour groups, how to avoid attempts at obfuscation used by attackers and how to predict future malicious behaviour for a given web-based attack based on its behaviour in real time. Finally, we have demonstrate how the proposed system will work by implementing a prototype application and conducting a number of experiments to show how we were able to model, cluster and predict web-based attacks based on their behaviour. The experiment data was collected randomly from online blacklist websites.

Acknowledgements

During the time of writing of this PhD thesis I received support and help from many people. In particular, I am thankful to my supervisor, Prof. Omer Rana, who was very generous with his time and knowledge and assisted me in each step to complete the thesis.

I would like to thank Dr. Peter Komisarczuk and Mr Damian Southard who helped me with the experimental data and the use of HoneyPot, without their help and support I could not complete my research.

I am grateful to all my friends for their encouragement and friendship. To them and to the other friends I met in Cardiff University, Ioan Petri, Sultan Alyahya, Marco Lackovic, Rafael Tolosana, Sathish Periyasamy, Adrien Maglo, Ricardo Rodríguez, Ahmad Alqoud, Ahmed Alazzaw, Waleed AlNuwaiser, René Brunner, Khalid Almuzaini, Haya AlMagwash, Shada Alsalamah and Yulia Cherdantseva, goes my deepest gratitude for the intellectual and cultural growth they have provided me.

And finally, but not least, thanks go to my whole family, who have been an important and indispensable source of moral support, especially my mother, father and wife, whose precious presence has been vital to the achievement of this goal.

Table of Contents

1	Introduction	1
1.1	Honeypot	3
1.2	Research Hypothesis	4
1.3	Research Aims and Objectives	5
1.4	Research Questions	6
1.5	Overview of the Thesis	8
1.6	Publications	9
2	Background	12
2.1	Web-based Malware	12
2.1.1	Malware sources	14
2.1.2	Malware types	15
2.1.3	Web-based malware tools	16
2.2	Honeypot	17
2.3	Active Honeypots (Client Honeypots)	19
2.3.1	Client Honeypot architecture	20
2.3.2	Low-interaction client Honeypot	21
2.3.3	High-interaction client Honeypot	21
2.3.4	Low-interaction client Honeypot tools	23

2.3.5	High-interaction client Honeypot tools	23
2.3.6	High-interaction client Honeypot challenges	25
3	Literature review	27
3.1	Malware analysis	27
3.1.1	Static analysis	27
3.2.2	Dynamic analysis	29
3.2	Malware modelling and representations	36
3.3	Malware detection	39
3.3.1	Hidden Markov Model (HMM) for malware analysis and detection	42
4	Automated State Machine	47
4.1	web-based attack behaviour models	49
4.1.1	The motivation of web based attacks	49
4.1.2	The state order	50
4.1.3	The challenges	51
4.1.4	web-based modeling methods	52
4.2	State Machine	53
4.2.1	State Machine types	54
4.2.2	Finite state machine modeling for web-based attacks	54
4.2.3	4.2.3 A client Honeypot with a state machine	57
4.2.4	Monitoring and detection tools	58
4.2.5	An example of drive-by download attack	59
4.2.6	Evaluation	63

4.3	Summary and Discussion	68
5	Clustering model	69
5.1	Clustering model overview	69
5.1.1	The motivation of web based attacks clustering	70
5.1.2	Clustering methodology types	71
5.1.3	Clustering model based on FSM	72
5.2	Clustering model	73
5.2.1	FP-growth clustering model	73
5.2.2	FP-growth clustering methodology	74
5.2.3	FP-growth clustering algorithm	75
5.2.4	FP-growth clustering groups	76
5.2.5	FP-growth evaluation	78
5.3	Summary and Discussion	80
6	Prediction model	82
6.1	Prediction model overview	84
6.1.1	Prediction model methodology	85
6.1.2	Prediction algorithm	85
6.1.3	Implementation	90
6.2	Classification	92
6.3	Summary and Discussion	93
7	Evaluation and Usage	95
7.1	An overview of Honeyware	96

7.1.1	The motivation of Honeyware	97
7.1.2	Overcoming client Honey-pot challenges	97
7.1.3	Honeyware architecture	99
7.1.4	Honeyware design	101
7.1.5	The Honeyware client	105
7.2	Web-based behaviour model evolution overview	106
7.2.1	Data collection	106
7.2.2	Experiment methodology	106
7.2.3	Experiments	107
7.2.4	The results	112
7.3	Hybrid system	118
7.4	Summary and Discussion	122
8	Conclusion and future work	124
8.1	Evaluating research hypothesis	125
8.2	Answers to the Research Questions and Problems	126
8.2.1	Behaviour states	126
8.2.2	Grouping web-based attacks	127
8.2.3	Predicting malicious behaviour	128
8.3	Contributions and novelty	129
8.3.1	Common behaviour clustering approach	129
8.3.2	Predicting malicious behaviour	129
8.4	Future Work	130
8.4.1	Improvement of the scan engine	130
8.4.2	Clustering obfuscation	131

8.4.3	Client-server model	131
8.5	Web-based Malware Framework	132
8.6	Lessons Learned	133
8.7	Concluding Remarks	134
	Bibliography	135

List of Figures

1-1	Total number of vulnerabilities by Symantec 2011	5
2-1	The attack pathways from the 2010 data-breach investigations report	14
2-2	Screenshot of MPack.	17
2-3	Client Honeypot architecture.	21
2-4	The Capture-HPC architecture	24
2-5	The Capture-HPC client inside the operating system	24
2-6	Example of obfuscating malicious code	25
2-7	The two types of client Honeypots, their advantages, disadvantages and tools	26
3-1	An overview of malware analysis approaches.	34
3-2	MIST representation	38
4-1	Web-based behaviour system.	48
4-2	The drive-by download attack scenario	52
4-3	The state machine diagram	56
4-4	The processes monitor in action	61
4-5	Port activity after the exploitation of the vulnerability on the user's machine.	61
4-6	The automated state machine of the CVE-2005-0553 exploit	62
4-7	Snapshot of Capture-HPC log file	63
4-8	The state machine states in Windows Workflow Foundation	64

4-9	The file produced by the state machine tool.	65
4-10	Example of a state machine signature.	65
4-11	Weka graphical output of the 26 malicious web pages	67
5-1	FP-growth clustering	74
5-2	Main and sub-behaviour group of the Trojan.Apher family.	78
6-1	The prediction model overview	86
6-2	An overview idea of the prediction algorithm.	87
6-3	The prediction model interface.	90
6-4	A user interface showing prediction outcomes	91
6-5	Classification model	92
7-1	IP tracking scenario using Honeyware.	98
7-2	Mpack attack methods.	101
7-3	Honeyware architecture.	102
7-4	Honeyware results page	103
7-5	The Honeyware main page.	104
7-6	Communication between Honeyware and its clients.	105
7-7	The state machine model while interacting with malicious web page.	108
7-8	The scan results of Poison Ivy on http://vscan.novirusthanks.org .	109
7-9	The scan results of the encrypted Poison Ivy on http://vscan.novirusthanks.org .	110
7-10	One state machine signature from the malicious web page experiment.	112
7-11	Four state machine signatures from Group ID 1 in Table 7-2.	114
7-12	The prediction scenario for the second experiment.	117
7-13	Experiment results on 4 November.	119
7-14	Experiment results on 8th November.	120
7-15	The antivirus detection results.	121
7-16	The hybrid system.	122

List of Tables

1-1	Total number of vulnerabilities by Symantec 2011	2
2-1	Example of Mpack malware state changes.	22
3-1	A comparison between different clustering algorithms and FP-Growth approach.	35
3-2	A comparison between the three malware detection systems	42
4-1	The state transitions table of the finite state machine in Fig 4-1	56
4-2	The state types.	58
4-3	Each monitored activity along with the relevant tools	59
4-4	The state's characters and descriptions	66
5-1	Example of state machine signatures	70
5-2	Three state machine signatures and their label underlined.	73
5-3	Scan result of the Trojan.Apher family	77
5-4	Three Trojans state machine signatures.	77
5-5	The behaviour groups output from FP-Growth clustering.	79
5-6	The experiment behaviour groups by HAC	80
5-7	A comparison between FP-Growth and HAC.	80
7-1	Web browsers supported by Honeyware with each user agent string.	100
7-2	Number of malicious URLs used in our experiment	108
7-3	The path table of poison ivy.	111
7-4	The FP-growth experiment results.	113

7-5	The third sub-groups from the FP-growth clustering experiment	113
7-6	The third sub-groups from the FP-growth clustering experiment	113
7-7	The variation of states for each malicious stage.	115
7-8	Demonstrating the variations in the next state given the current state and the contents of the behaviour database	116
7-9	Predicting the next three states experiment results	118

Listings

5-1	The clustering algorithm pseudo-code.	76
6-1	How we select the similar state machine signatures from the malware database	87
6-2	The prediction model pseudo-code.	88
6-3	The forward and backward probability algorithm.	90
7-1	An example of behaviour rule to detect malicious web page	118

Chapter 1

The internet represents a vital resource identifying a collaborative process between organisations and individuals, which can communicate and perform business processes. The internet arose from a research environment, so it was not designed to be a very secure environment. Therefore, various internet users, such as agencies and organizations have been attacked or probed by intruders, with resultant losses to productivity and reputation. In some cases, organizations have decided to disable their internet access and have invested significant resources for improving the security of their internet connection. Internet connectivity offers enormous benefits in terms of increased access to information, but using the internet can become a dangerous experience for those with low levels of security. Major threats of using the internet are problems with TCP/IP services, the complexity of host configuration and vulnerabilities introduced in the software development process.

Internet security appears as a policy for deciding how an organisation or internet user is going to protect themselves against different threats. Within this policy, there are two important parts: (i) a mechanism of protection against internet vulnerabilities and (ii) specific rules implemented in accordance with the context of internet usage.

Before the internet, computer security was restricted to 'closed systems' or networked computers in offices or banks, where only users could operate the computer system. Once the internet appeared, computer users were able to work in an 'open system', but security became a sensitive aspect. Even though users can connect to the internet and perform remote transactions, there is a security issue involved, as the transaction takes place through the internet by bouncing the information through various computers on the route from user to destination and back again.

There are many researchers, as well as industry, working to improve the level of internet security to organizations and to home users [2]. In the past, most of the attacks used viruses or worms to open a new window, change the wallpaper or pop-up an advertisement, without serious impact to the victim's system. However, recent attacks use complex and serious malicious methods and exploits to attack the target networks, sponsored by government or by individual attackers. According to the Internet Security Threat report 2011 by Symantec [138], shows the total number of vulnerabilities in Table 1-1:

Year	Total number of vulnerabilities
2008	5,562
2009	4,814
2010	6,253
2011	4,989

Table 1-1: total number of vulnerabilities by Symantec 2011 [138].

A study has shown that a successful compromise of a computer with the default installation of Redhat 6.2 occurs within 72 hours. Where a computer has the default installation of Windows XP, it is compromised within 15 minutes [3] [126]. Many aspects have to be made secure to ensure the security of the internet, including hardware, servers and clients. This work is concerned with the client side and the threats that arise out of it. Generally, web attacks are divided into two forms:

1. **Server-side attack:** The server provides some services, such as Apache HTTP server, which contains vulnerabilities that lead to attacks on that service, including the possible takeover of the whole server. OWASP report [139] provides the top 10 web security attacks, 8 out of 10 top attacks are server attacks such as SQL injection where it compromised the SQL database in order to access unauthorised data.
2. **Client-side attack:** Here the attacker threatens the client application within the user environment. Examples of client applications are: web browsers, media players (WinZip or RealPlayer) and Microsoft office.

Attackers have preferred client-side attacks in recent years because server-side attacks have been made difficult by the improvement of a wide range of security measures, such as Firewalls and IDS. Therefore, attackers (Blackhats) are attracted to client-side attacks because of the lack of security measures.

1.1 Honeypot

“In warfare, information is power. The better you understand your enemy, the more able you are to defeat him”[4]. In order to defeat attackers, security professionals need to learn and understand the attacker’s methods, tactics, motivation and tools. A Honeypot is defined as a “security resource whose value lies in being probed, attacked or compromised” [5]. An alternative definition is “A program, machine or system put on a network as bait for attackers” [6]. The aim of the Honeypot is to collect data on attacks and attackers by monitoring the machine being attacked. Alternatively, it can emulate the operating system services to detect attacks. Because the Honeypot is a provision made by the owner of the network, it should not receive any internet traffic from the network or the outside world. Consequently, any traffic that comes to the Honeypot could be an attempt to attack or scan the system. Generally, Honeypots are divided into two main types: passive and active Honeypots (client Honeypots). A passive Honeypot involves setting up a vulnerable system or service, or possibly their simulation, and then monitoring activity to detect any attack on the system. The objective of this process is to then improve the security of the operating system and network, as well as gather information about the attacker’s motives and behaviour. By contrast, instead of waiting for attackers passively, an active Honeypot will go and search for the attackers. The Honeypots can be used:

1. By the network administrator, to learn how the attackers are getting into the networked computers, by monitoring attacker methods and exploits when they compromise the Honeypot system.
2. To collect known and unknown malicious codes for analysis by security professionals and companies to release patches or learn new attack methods used.
3. As an easy target for the attackers to compromise. Therefore, it will attract the attacker’s attention while keeping their eyes away from network servers, which are harder targets, meanwhile the network administrator is informed of the attack and able to defend it.

The log files of Honeypot traffic have a high value for the security team and owner of the network because they reveal the traffic of the attacker with a low false positive, compared to a firewall or an Intrusion Detection System (IDS). By using a Honeypot, the effort required to analyse the log files from other security techniques can be avoided.

1.2 Research Hypothesis

Malicious web pages can be detected by using their signature and behaviour. Most of the techniques are based on the signature, such as anti-malware software, or by using a pre-defined list, the blacklist, and then checking if the target web page matches or not. The detection and prediction of malicious web pages based on behaviour is an emerging area and there is limited work on how to build a real detection and prediction model to analyse malicious web pages based on their behaviour and activities on the end-user's system.

The hypothesis of this research is that monitoring behaviour is a more efficient, safer and accurate approach to detecting web-based malware than a signature-based approach. This thesis has created a number of approaches that can serve to build detection and prediction models. The first method is to encode malicious web page behaviour using a state machine, which is presented in chapter 3. A clustering model is developed that can group similar malicious web pages in order to understand the relationships and similarities between particular web-based malware and others. This clustering model, which is described in chapter 4, can also find additional correspondences by creating sub-groups within each main behaviour group. The objective is to detect malicious web pages through their behaviour and activity on a client. Combining a prediction model with clustering and classification algorithms enables the detection of malicious behaviour at run time and provides a useful way to block web sites without needing to analyse the signatures of malware. The web-based malware behaviour model can be used to analyse, group, detect and predict the activity of a malicious web page within the end user's environment while the client is surfing the web. Alternatively, it can be used in test environments, such as the Client Honeypot, in order to analyse and predict future behaviour of malicious web pages.

The client Honeypot monitors the web-based malware as it interacts with the end-user's system, interrupts its events and logs them. The events caused by the malicious web page may change the system state in order to compromise it, such as through writing new files or opening a back-door. We encode these events into a state machine in order to understand them and use them with clustering and prediction models. Fig 1-1 shows an overview of how the web-based malware events were monitored.

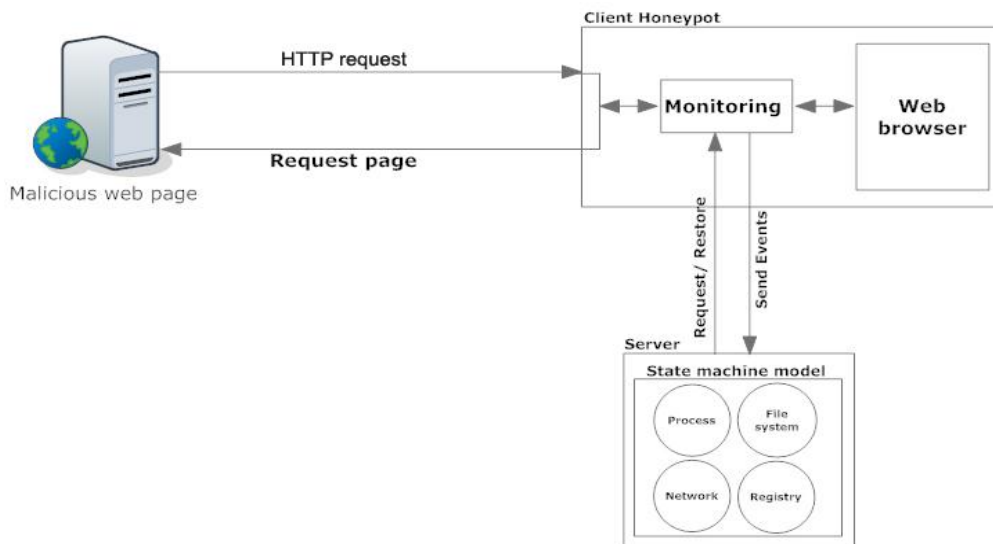


Fig 1-1: an overview of how the web-based malware events were monitored.

1.3 Research Aims and Objectives

The overall aims of this thesis can be summarised as follows:

“The use of the pre-defined malicious behaviour to detect web-based attacks by predicting future potential malicious behaviour”

The main aims of the thesis are as follows:

- Detect malicious web pages based on behaviour which represents actual malicious activity, even when the exploit code has been encrypted and packed.
- Provide real-time behaviour analysis of the web-based attack while interacting with a web browser.
- Identify and understand similarities between malicious web page exploits and define common characteristics to group them in one behaviour group.
- Improve the detection accuracy of Web-based attacks using information from previous attacks.

1.4 Research Questions

Question 1: How does the security analyst, network administrator and end-user interpret the behaviour of a malicious web page?

Malicious websites may cause a number of activities, as explained in chapter 4, to be performed on the victim's system; each activity is performed in different stages. Therefore, if a malicious website causes three activities, then they will be performed in three stages, represented as three independent states.

An automated state machine has been used to convert the malicious web page stages into pre-defined states. Websites may contain one or more malicious web pages, which can affect the visitors, while the other pages are benign. For this reason the author focuses on the term 'web page' as opposed to 'website'. These states can describe every malicious web page using the same state machine sequence. There will be a pre-defined sequence for any malicious web page, called a "state machine." The state machine advantages can be summarised as:

- **System administrator:** Helps the system administrator to explore web-based malware activities and protect networked machines, by applying the appropriate patches or reverse malicious states to undo their effects.
- **Security researcher:**
 - Creates a pre-defined sequence for any web-based behaviour and activity.
 - Provides easy and simple states (activities) describing web-based malware behaviour to security researchers.
- **End-user:**
 - Provides a web-based activity to the end-user in a simple and easy way that allows them to understand what malware actually does to a victim's system.
 - Provides the ability to explore the activities of current web-based malware attacking their machine.

Question 2: How can the relationships between different malicious web pages be analysed and displayed, based on their behaviour within the end-user's system?

To answer this question, two different clustering algorithms were implemented. The clustering algorithms are able to read the state machine signatures, encode malicious web page behaviour

and then group similar malicious web page exploits together. The output of clustering algorithms will be categorized to build up groups of similar web-based malware, which represent the malicious behaviour performed on the victim's system. The benefit of using this process is to identify families of malicious web pages with similar behaviours (behaviour families). The advantages are:

- **System administrator:** Provide the ability to explore the behaviour steps of web-based malware attacking the network in order to protect the network from other web-based malware in the same family.
- **Security researcher:**
 - Can build behavioural families of web-based malware. There are currently no accepted web-based malware categories based on behaviour.
 - Can analyse similar web-based malware to find differences between one malicious web-based activity and another.
 - Can reduce the time required to analyse web-based malicious files, by using behaviour families to cluster similar malicious web-based activities, thereby enabling security researchers to analyse and study them quicker.

Question 3: How can the future behaviour of a running web page be predicted based on its past behaviour?

The clustering models as described in chapter 6 were used to understand the relationships and similarity between the behaviour of malicious web pages. The prediction model will check the current web page behaviour and try to predict the subsequent activities of the malicious web page. If the model can predict the next activity of a web page, and determine it to be malicious, then it will be able to block the web page and protect the victim's machine. The aim of the prediction model is to act as an intrusion detection system (IDS) by using the behaviour and activities collected from the state machine and analysed by the clustering algorithm. The objective is to detect malicious web pages through their behaviour and activity with a client. Combining a prediction model with the clustering and classification algorithms enables the detection of malicious behaviour(s) at run time and provides a useful way to detect web sites. This is discussed in chapter 7 (without needing to analyse signatures of malware). The advantages of the prediction model are:

- **System administrator:**

- Help detect zero-day exploits (unknown attacks). The zero-day exploit is defined as “An exploit that has no corresponding patch to counteract it” [134].
 - Assist work with other security mechanisms to provide better security protection. Traditional security mechanisms include firewalls, intrusion detection systems (IDS) and anti-malware software.
- **Security researcher:** Enables the detection of web-based malware, based on activities and behaviour on a victim’s machine.
 - **End-user:** Enable the detection of known and unknown web-based malware by monitoring the current web page and predicting its future malicious behaviour based on previous knowledge of the prediction model.

1.5 Overview of the Thesis

Chapter 2 examines the approaches currently being used in the area of malware analysis, detection and modelling. This chapter aims to introduce related work in detail and explain how our approaches and models will differ from existing approaches.

Chapter 3 discusses web-based malware techniques and tools. Furthermore, it introduces Honeypot technology and describes how it can help by monitoring and detecting malicious web pages. At the end of this chapter, we introduce our web-based behaviour model which combines the three different models – state machine, clustering and prediction – in order to help detect and analyse malicious web pages dynamically.

Chapter 4 describes the state machine model and shows how it works to monitor and encode web-based malware behaviour. This chapter also shows how we implement our automated state machine and then evaluate it by scanning real web-based malware. It also uses simple statistical algorithms to illustrate the common states that can be gleaned from the state machine signatures such as how many web-based attacks start by writing a file or establish a network connection in the end of the chapter.

Chapter 5 describes the idea of using a clustering algorithm with the state machine signatures. The use of FP-Growth clustering algorithm to build the clustering model using state machine signatures is described.

Chapter 6 presents the last model involved in our web-based behaviour system, which is the prediction model. The prediction model is based on the Hidden Markov Model (HMM) and is

used to predict the potential future behaviour of malicious web pages in a multi-stage manner. This chapter discusses how we designed the prediction model, its implementation and finally gives a classification model that can help to find the correct behaviour group for a given state machine signature faster than the clustering algorithm.

Chapter 7 evaluates the proposed web-based behaviour system and shows our experimental results and findings.

Chapter 8 concludes the dissertation by discussing the contributions it has made to the literature and possible future work that could be conducted to improve its findings.

The novelty of this research is the contributions to the security field in two main areas. The first contribution allows to group web-based attacks according to their common behaviour. This new grouping system outputs specific types of groups adapted to web attacks, which are different than local attack groups such as virus and worm. It can cluster the malicious web pages based on their behaviours but not based on their signatures as most of the available security approaches. Furthermore, the model produces a label for each group that shows the common behaviours held in that group. Their label can be used to identify similarities between attack groups as well as to merge them into one common behaviour group. The clustering model and experiments are discussed in details in chapter 5. The second contribution is the prediction of the malicious behaviour of web pages in real time. The prediction model works by finding attacks with similar behaviour to the current stage of the web-based interaction. Moreover, it calculates the possibilities of future activities for current web-based attacks based on similar state machine signatures. If the prediction is accurate, our system blocks further interaction with the external server, thereby limiting the effect of potential malicious activities. It also saves the new state machine signature to the malware database to speed up the prediction process in the future. The prediction model is described in details in chapter 6.

1.6 Publications

Alosefer, Y., Rana, O., "Automated State Machines Applied in Client Honeypots", 2010 5th International Conference on Future Information Technology (FutureTech), pp.1-8, 21-23 May 2010. doi: 10.1109/FUTURETECH.2010.5482695 URL:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5482695&isnumber=5482630>

This paper shows the benefits of client honeypots as well as the use of an automated state machine in conjunction with a client honeypot. The state machine provides a powerful framework to organize monitoring of malware activity and record the results. The automated state machine, when it is applied to a client honeypot as described in this paper, gives honeypot technology a number of benefits when analysing malware and tracing the details of malware steps.

Alosefer, Y., Rana, O., "Honeyware: A Web-Based Low Interaction Client Honeypot", 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW), pp.410-417, 6-10 April 2010. doi: 10.1109/ICSTW.2010.. URL:

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5463678&isnumber=5463636>

This paper presents the idea of using web-based technology and integrating it with a client honeypot by building a low interaction client honeypot tool called Honeyware. It describes the benefits of Honeyware as well as the challenges of a low interaction client honeypot and provides some ideas for how these challenges could be overcome.

Alosefer, Y and Rana, O., "Clustering Client Honeypot Data to Support Malware Analysis". 2010. Springer. Knowledge-Based and Intelligent Information and Engineering Systems 4th International Conference, KES 2010, Cardiff, UK, Proceedings, Part IV. Springer Verlag.

This paper shows a clustering algorithm to group similar malicious web page exploits in order to better understand how software can be developed to better respond to such attacks. The outputs of the clustering algorithm are categorized to build up groups of similar states that represent the malicious activities performed on the victim's system. The benefit of using this process is to build families of malicious web pages with similar behaviours (behaviour families).

Alosefer, Y., Rana, O.F., "Predicting client-side attacks via behaviour analysis using honeypot data," 2011 7th International Conference on Next Generation Web Services Practices (NWeSP), pp.31-36, 19-21 Oct. 2011doi: 10.1109/NWeSP.2011.6088149. URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6088149&isnumber=608814>

Web-based malware overcomes signature-based detection by modification of the code or using zero-day exploits. In this paper, we propose a malicious activity detection method using Hidden Markov Models (HMM) alongside a client honeypot system. Our algorithm is able to detect the potential malicious behaviour of a web server based on current and past interactions between the web client and the server and can also predict possible future behaviours. The prediction algorithm learns from previously scanned behaviours recorded by a client honeypot system. We group such behaviours in order to enable common characteristics to be investigated across these groups.

Alosefer, Y., Rana, O.F., "Predicting and Analysing Client-Side Honeypot Behaviours", 2012, The International Journal of Computer Information Systems and Industrial Management (IJCISIM), Accepted.

The client honeypot is useful for analysing malware behaviours with a low false positive rate. From a security point of view, this method is better than traditional malicious activity detection (signature-based), such as used by anti-virus products. The prediction model described in this paper is able to detect previously known and unknown malicious web behaviours at run time as well as provide a useful way to block web sites without the need to analyse malware signatures. Furthermore, our model is able to detect and analyse web-based malware behaviour in both testing and end-user environments.

Chapter 2

Background

There are many countermeasures that can help end-users and organisations stop some of the attacks on their computers and networks. These countermeasures include firewalls and intrusion detection systems (IDS). These products can reduce risk but not eliminate it. They have a high potential for false positives and false negatives, which makes it difficult for the security team to analyse them for the detection of true attack attempts. The numbers required for this kind of work are also rather high. However, a new method focuses on the collection of data, compared to traditional security technologies [136][137]. This data has high value and can reduce the numbers of false positives and negatives; this method is termed the Honeypot. In this chapter, section 2.1 introduces and discusses web-based malware as well as its sources, types, behaviours and tools. Section 2.2 defines a Honeypot and gives an overview of the technology. Section 2.3 describes the active Honeypot (client Honeypot), its types and its tools.

2.1 Web-based Malware

Most attacks today can be successfully defended by using security technologies such as Firewalls, IDS and anti-virus software, so the attackers have developed new methods to spread their malicious code by using web pages, which can affect many more victims. The attackers now use these websites to threaten users without the user's knowledge or

permission. The defence of websites is less effective than traditional security products. Therefore, the attackers have the advantage of targeting a greater number of users.

Provos et al. [81] define a malicious webpage as one that attempts to cause the automatic installation of software without the owner of the computer or authority knowing. The malicious web page tries to launch an attack on the user's system, which only a highly skilled user is able to detect.

Attackers have preferred client-side attacks in recent years because server-side attacks have been made difficult by the improvement of a wide range of security measures, such as Firewalls and IDS. Therefore, attackers (Blackhats) are attracted to client-side attacks because of the general lack of security measures. The goal of the attacker is to place malware into a user's environment and then collect important information, such as credit card data. The attackers are able to detect vulnerabilities in a user's application by just a single visit by the user to a malicious web page. They then exploit this vulnerability to get control of the user's machine. This type of attack is also called *drive-by-download*. Polychronakis et al. [82] define the drive-by-download attack as "a malicious web page exploiting a vulnerability in a web browser, media player or other client software to install and run malware on the unsuspecting visitor's computer." This kind of attack uses web application technology to pose its threats. There are several ways to deliver malicious code, but first a vulnerability must be found within the user's machine. Examples are a vulnerability within the web browser (Internet Explorer, Firefox), or in client software which can interact with the web browser, such as media players (Windows Media Player, Real Player) and office applications (Microsoft Office), or in other applications, such as Adobe Acrobat reader or WinZip. The malicious web page can affect end-users in a number of ways. A user can click on a link of spam email that will allow the attacker to exploit his machine. Drive-by download attack is a method to attack the user via HTTP/HTTPS requests in a number of possible ways. Hence, drive-by-download refers to a series of possible ways in which a client machine can be attacked from a remote server. For example, it can redirect the user from a benign web page to an exploit server where the malicious code is sent to the victim's machine. According to OWASP [139], the second of the top 10 web security risks is Cross-site scripting (XSS), which can be used by the attacker to redirect the user to malicious web page in order to affect his machine by drive-by download attack.

After the attacker identifies a vulnerability within the user’s client application, they can exploit this in order to install malicious software, run new processes, change existing files or add new files to the system. Fig 2-1 below shows that a survey conducted by the Verizon RISK team in cooperation with the United States Secret Service (USSS) [130], which in 2010 contributed a few hundred of their breach cases to the study, found that the web application attacks were responsible for 54% of breaches by hacking and for 92% of records breaches.

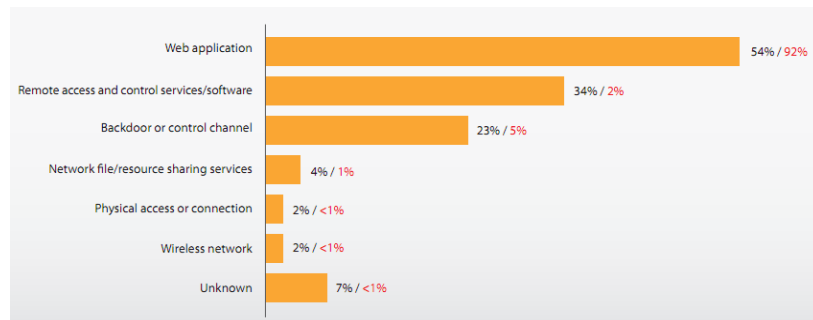


Fig 2-1: the attack pathways from the 2010 data-breach investigations report [130].

2.1.1 Malware sources

Attackers will try any possible way to deliver their malware onto a user’s machine. However, there are some common techniques that lead to attacks on larger numbers of people than others. According to Provos et al. [81], an experiment was conducted to analyse web-based malware, which found that there are four main aspects that could result in attacks.

- **Advertising:** A site’s webmasters are always looking for ways to optimise the benefits of their website and earn funds for the owners. The webmaster has to put their trust in the company advertising to display non-malicious code within the banner. Meanwhile, the company advertising will try to display non-malicious code in their banners with the aim of getting more websites subscribed to their services without damaging their reputation. The implementation of the advertisement is by the webmaster adding JavaScript code, from the company advertising, into their web pages. That JavaScript code generates a banner every time the page is reloaded, so that each visitor will see different banners. This method could be used in a bad way; for example, the website or company advertising could be attacked by malicious code being placed by the attackers within the banner to adversely affect large numbers of visitors without anyone ever knowing.

- **Third-party widgets:** The idea of third-party widgets is that a website's owner will try to add more functionality from a third-party developer such widgets may be free or may be paid for. The widgets could include a weather data or internet speed tests. The website owner embeds these widgets in their website using JavaScript. These widgets can be targeted by attackers, and their own malicious code can be embedded within the widgets, resulting in an attack on all the website's visitors by attacking the third party's widgets which are embedded in the registered website.
- **User contributed content:** In recent years, user-contributed content has been growing. It connects people with one another within a given website, such as a social network, forum or blog. These websites target more people and visitors, which might lead the attacker to focus on them, as they have a large number of people using them. For example, the attacker can perform a phishing attack through Facebook which can then easily spread to other members of the Facebook community through sharing or liking.
- **Web server security:** Web servers are an important target for attackers. Web servers, such as Apache HTTP server, and their security, along with web programming languages, such as PHP or ASP, can all be vulnerable to direct attack. In the case of attacking the website from the server, the attacker can attack the visitor's website easily as they have full control of the server and are able to embed any malicious codes or files into their web pages.

2.1.2 Malware types

Malware code and scripts affects programs as well as operating systems. Malware can be defined as "a set of instructions that run on your computer and make your system do something that an attacker wants it to do" [83]. Once a malicious web page attacks the user's system, then it is able to download malware including the following, as described by Provos, et al. [81] and [84]:

1. **Virus:** This is the traditional kind of malware and takes many different forms. However, it requires user interaction to download, and it is not able to propagate through a network by itself. Viruses are able to harm a system in many ways, from changing a user desktop wallpaper to formatting hard drives.

2. **Worm:** A worm is similar to a virus but it is also able to self-propagate, and can attack other systems through the network and removable devices.
3. **Rootkit:** A rootkit is the most dangerous type of malware. It attempts to modify the operating system files and applications and to perform activities on behalf of the attacker, such as hiding malware files, processes and network traffic.
4. **Trojans:** The unique aspect of a Trojan is that the user is required to download benign software that has malicious code embedded in it without the user's knowledge. When the user installs the software, then the malicious software is installed as well.
5. **Adware:** Displays advertisement banners on a user's machine. These advertisements might be displayed on the user's desktop as a pop-up, or redirect the user's web browser to unwanted websites.
6. **Spyware:** Spyware software tries to collect information about a user's browsing behaviour, such as their web browsing history, cookies, their most-visited websites or even their search keywords. These are then sent to the attacker.

2.1.3 Web-based malware tools

Web-based malware can also include tools that give the attacker more advanced control and wider options. Web-based attack tools are easy to use and can have significant effect upon the victim's machine. According to Seifert [85], there are at least three different web-based exploitation tools, which are not available for general download but can be bought on the underground market.

1. **WebAttacker:** This was one of the first web-based exploitation tools. It originated at the beginning of 2006.
2. **MPack:** The second web-based exploitation tool was developed by the Dreamcoders team. The tool was released in June 2006. Fig 2-2 shows an Mpack screenshot [85]. It shows the attacker has the ability to view how many users are affected, and provides such statistics as their web browser, operating system, country and the previous referrer's URL.

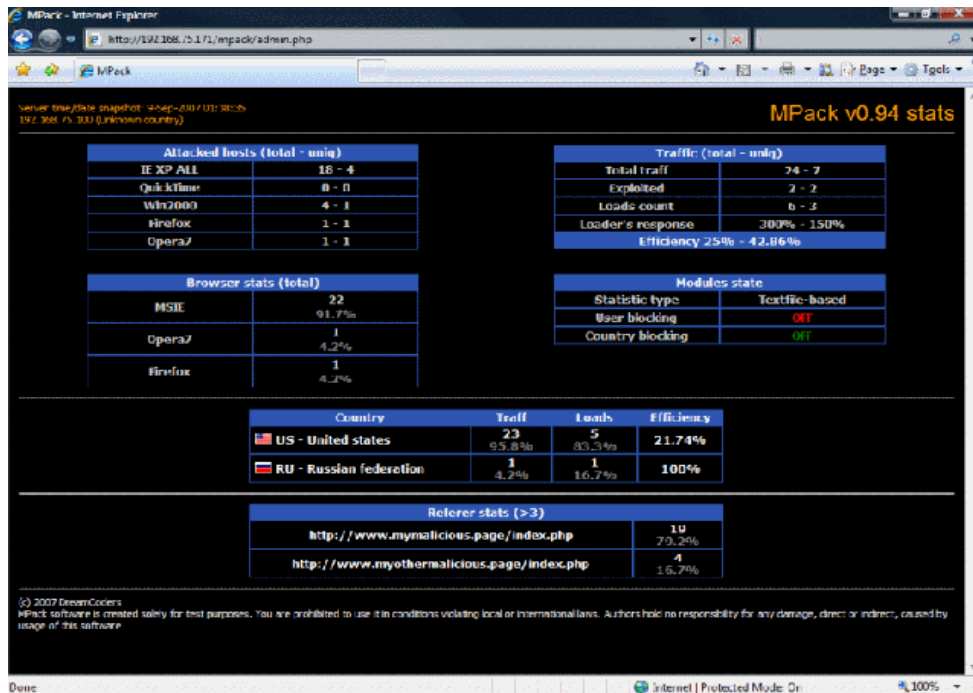


Fig 2-2: Screenshot of MPack [85].

- IcePack:** The IcePack tool was written by the IDT group and was released in July 2007.

2.2 HoneyPot

A HoneyPot is defined as a “security resource whose value lies in being probed, attacked or compromised” [5]. The aim of the HoneyPot is to collect high-value data by monitoring the state of an existing operating system or services. Alternatively, it can emulate them to detect traffic that could come to a server. Because the HoneyPot is a provision made by the owner of the system, the HoneyPot should not receive any internet traffic from the network or the outside world. Therefore, any traffic that comes to the HoneyPot could be an attempt to attack or scan the system. The log files for this traffic have high value for the security team and owner of the network because they reveal the traffic about the attacker without any false positives. Using the HoneyPot can reduce the effort required to analyse the log files from other security techniques. According to the HoneyNet project [86], the information collected by HoneyPots is explained and used differently by each organisation. For example:

- Academic institutions:** The data collected by a HoneyPot can help them with their research into viruses, worms and attackers’ techniques.

- **Security organisations:** These organisations use the Honeypot to detect malicious activity and to try to understand them in order to build security defences for them by, for example, creating signatures for new viruses.
- **Government organisations:** A Honeypot can help them halt attacks and discover where they originate.

The Honeypot has advantages and disadvantages which are summarised in the following points. The advantages of using Honeypots are:

- Collecting and logging high value data for the security team and owner of the network because they reveal the traffic of the attacker.
- Few false positives and false negatives compared to a firewall or an Intrusion Detection System [136] [137].
- High value of the data to avoid the effort required to analyse the log files from other security techniques.

On the other hand, the disadvantages of using Honeypots are:

- The risk of compromising the system, leading to attacks on other systems on the network or beyond it. The Honeypot is an easy target for the attackers and therefore they can focus on compromising the system and this will result in new attacks to the network and the internet.
- The need for highly skilled people and plenty of time to operate and analyse the data. Our work operates without the use of highly skilled people and is able to analyse the collected data from Honeypot automatically by using our three models, state machine, clustering and prediction.

Honeypots can be implemented in different ways on a real computer, as a service, operating system emulated, service emulated or unused IP address on the network. This technique can bring some risks to the network as well as benefits, so it should be used carefully. Generally, the Honeypot approach covers two main types: passive Honeypots and active Honeypots (Client Honeypots) [87].

When implementing a Honeypot system on a production network, such as a company network, the implementation team should think about the different issues in implementing the Honeypot in order to mitigate the risk of Honeypots, as mentioned earlier in section 2.2.

As Mokube and Adams [88] suggest, when a Honeypot is being used the legal and privacy issues need to be looked at as some countries and cities have their own laws about implementing Honeypots in production networks as they might cause a risk to networks. Examples of the legal issues that must be considered while implementing a Honeypot are: entrapment, privacy and liability. The attacker could take over the Honeypot system and then use it as a starting point to attack other systems on the network or internet. This action by attackers is called uplink liability. Therefore, there should be a focus on the legal and liability issues when implementing a Honeypot as well as strategies for how to respond in the case when the Honeypot system is compromised.

Most of the Honeypot's disadvantages and risks can be mitigated and avoided by the correct configuration and implementation of a Honeypot in the network. Experts suggest separating the Honeypot from the production network to avoid risks such as uplink liability [88]. In this work, we are focusing on active Honeypots, not passive Honeypots; therefore, our scanning engine does not suffer from all these risks as described in the next section. We will only monitor the web browser while it is interacting with the target malicious web page and then block or limit the outgoing connections. Furthermore, we can benefit from cloud technology by implementing the Honeypot in the cloud and pushing any monitoring activities from the production network to the cloud while having the scanning on the network. The amounts of data transferred to the cloud are only the changes made by the web based malware on the victim's system where the analysing and detection models are in the cloud.

2.3 Active Honeypots (Client Honeypots)

In recent years attackers have mainly been targeting client applications, such as web browsers or media players, whereas before they used to threaten servers. Therefore, there have to be ways to determine methods of attack and to capture malicious scripts and tools to improve security. The active Honeypot is a new concept and is quite different from the passive Honeypot. Instead of building a Honeypot and passively waiting for attackers, the active Honeypot will go and search for attackers. The client Honeypot acts as a client and interacts with the server to study it and determine if an attack has happened. The client Honeypot can

interact with websites using the HTTP protocol, as well as other protocols, such as SMTP and FTP. The active Honeypot has convincing names that are accepted publicly as client Honeypots and Honeyclients.

2.3.1 Client Honeypot architecture

The client Honeypot architecture is divided into three main parts: the queuing system, the client and the analysis engine. Each part is necessary for the smooth operation and correct performance of the client Honeypot. The following points below help to describe each part:

1. **The queuing system:** is responsible for creating the list of servers for the client to interact with. There are several techniques used to create the server list:
 - **Search Engine:** there are common malicious websites that use popular keywords or website content to target large numbers of visitors, such as those containing Music, Adult and News content.
 - **Blacklists:** a number of organisations publish malicious websites; examples of these organisations are www.mvps.com and www.stopbadware.org.
 - **Gather links from phishing and spam messages:** there are public archives of spam messages on sites such as www.untroubled.org/spam.
 - **Typosquatting domains:** typosquatting refers to mistakes in domain names such as goole.com instead of google.com, the real domain name.
 - **Monitoring instant messaging:** most instant messaging services such as MSN Messenger suffer from spam. Malware infects the victim and sends malicious URLs to their contact list. Therefore, we can monitor these services and collect malicious URLs for scanning and analysing.
2. **The client:** the client is the component that is able to make requests and interact with the servers collected by the queuing system.
3. **The analysis engine:** the analysis engine is responsible for checking the state of the client Honeypot system to determine if an attack has occurred or not. Fig 2-3 shows the client Honeypot architecture.

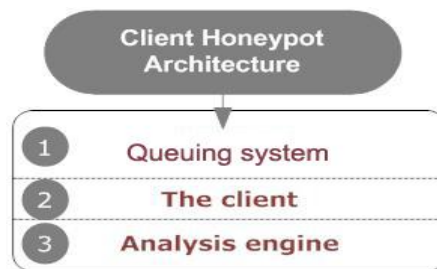


Fig 2-3: Client Honeypot architecture.

Client Honeypots are divided into two main types based on their level of interaction: low-interaction and high-interaction client Honeypot.

2.3.2 Low-interaction client Honeypot

The low-interaction client Honeypot simulates a client used for interacting with a server without using a real system, and instead using a lightweight simulation. The aim of this is to discover malicious code present within the server by interacting with it and then examining the reply that is sent to the client, which ultimately results in the detection of malicious code within the page. The examination of the reply is made by checking the string in the source page for attack-related code. The disadvantage with this type of client Honeypot is that it cannot detect unknown malware because it uses a signature database to match them with the response from the server. However, the low-interaction client Honeypot also has some advantages, such as its ease of use and deployment, and the speed of the generated report.

2.3.3 High-interaction client Honeypot

On the other hand, the high-interaction client Honeypot is the most advanced type and uses a real operating system to examine malicious servers. The basic concept behind this is that the operating system itself acts as a client, browsing a server which has been identified previously via the queuing system, just like a human browsing the website. After a visit to the website has been made by the client, the analysis engine checks for the presence of malicious code. Table 2-1 shows an example of Mpack malware state changes [85]. Several operating system components need to be monitored while visiting and scanning each web page:

- Processes will be monitored for new processes being run or any default processes being killed.
- Windows Registry will be monitored for any added, deleted, or modified registry values.

- File system will be monitored for any changes, such as adding, deleting or modifying files or folders.
- TCP and UDP ports will be monitored to detect if any ports are opened or closed, as these ports could be used as a backdoor by the attacker.

Monitor	Action	Actor	Action parameter
file	Write	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
process	Created	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
file	Write	C:\syswcon.exe	C:\WINDOWS\system32\drivers\uzcx.exe
process	Created	C:\syswcon.exe	C:\WINDOWS\system32\drivers\uzcx.exe
process	Terminated	C:\Program Files\Internet Explorer\IEXPLORE.EXE	C:\syswcon.exe
registry	SetValueKey	C:\WINDOWS\system32\drivers\uzcx.exe	HKCU\Software\ewrew\uzcx\main\cid
file	Write	C:\WINDOWS\system32\drivers\uzcx.exe	C:\Documents and Settings\cseifert\Local Settings\Temporary Internet Files\Content.IE5\OPUJWX63\benupd32[1].exe

Table 2-1: Example of Mpack malware state changes [85].

There are several advantages and disadvantages of using high-interaction client Honeypots. The advantages are:

- More effective in detecting unknown attacks.
- The ability to use different clients to interact with the server, such as different web browsers (e.g. Internet Explorer, Firefox and Safari).
- Using real operating systems and services to interact with the target server, allowing more resources to understand and analyse threats.

On the other hand, there are also disadvantages of using high-interaction Honeypots:

- The performance of the operating system could be reduced after the effects of the attacks.
- The analysis time and the examination of the state of the operating system and applications are longer.

- The attacker might detect the presence of the virtual environment if the Honeybot is run under such an environment.

2.3.4 Low-interaction client Honeybot tools

A number of tools exist for low-interaction client Honeybots. They are listed below, along with their descriptions:

- **HoneyC:** is an open source low-interaction client Honeybot developed by Christian Seifert [89]. The tool is written in the Ruby scripting language. This tool uses the three components mentioned above: first, it creates a list of websites by using the search engine Yahoo!; secondly, the website is identified and the page downloaded to be examined; finally, the web page code is analysed with Snort, an IDS tool that contains a database of signatures of malicious strings.
- **SpyBye:** is a low-interaction client Honeybot written by Niels Provos [90]. The tool looks superficially like HoneyC, but, instead of using the Snort signature database to check the web page, the ClamAV anti-virus engine is used.
- **Monkey-Spider:** is a low-interaction client Honeybot developed by Ali Ikinici et al. [91] at the University of Mannheim. This tool is available under a GPL licence and is free to use. Their approach detects malicious web pages on the internet, by crawling and collecting web pages from different sources such as search engines or spam emails, and then scans them to detect any malicious behaviour in the web page code. They use signature-based detection such as Clam-AV, F-Prot and Avast, and behaviour based detection by using CW-Sandbox. They do not analyse the behaviour of malicious web pages, but their approach only detects malicious code.

2.3.5 High-interaction client Honeybot tools

1. **Capture-HPC:** is an open source high-interaction client Honeybot developed at Victoria University, Wellington [36]. The tool is capable of interacting with servers through different types of HTTP applications, such as Internet Explorer, Opera and Firefox, in addition to media players and office applications. Capture-HPC has several advantages, for example, the tool is fast at detecting any changes in the operating system, since it monitors processes, the registry and the file system. In addition, the tool uses client-server methods so that it has server control over clients and saves all the log

data within the server. Lastly, the tool allows the creation of more than one client Honeypot and controls all of the clients from the server. Figs 2-4 and 2-5 show the Capture-HPC architecture and the Capture-HPC client inside the operating system [36]. Capture-HPC uses VMWare virtual machine [45] to create the clients and control them from the host system. It creates a number of virtual machine clients and controls them by using VMWare API as shown in Fig 2-4, which can help to start or restore the system as well as getting the malicious behaviour from the clients in real time. Fig 2-5 shows how Capture-HPC is installed inside the operating system of the client, where it can monitor the system calls in the kernel mode and log any changes made to I/O Manager, Process Manager or Registry Manager.

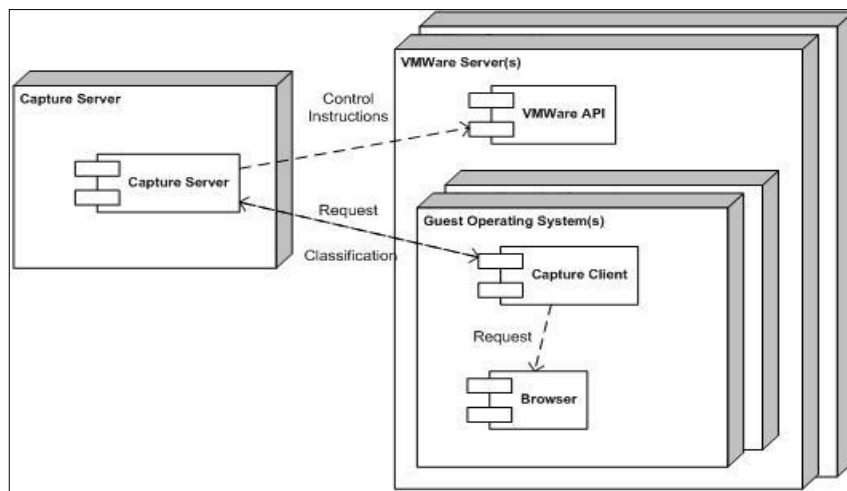


Fig 2-4: The Capture-HPC architecture [36].

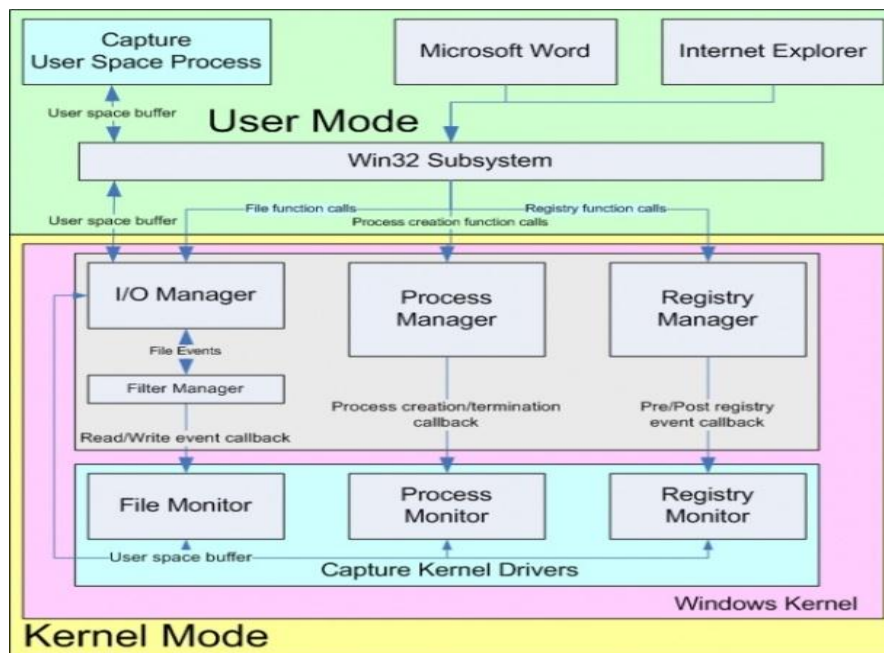


Fig 2-5: The Capture-HPC client inside the operating system [36].

2. **HoneyMonkey:** is a high-interaction client Honeypot produced by Microsoft [92]. This tool simply uses Internet Explorer to visit servers and monitors processes, the registry and files. It is focused on finding 0-day attacks against Internet Explorer. HoneyMonkey is not available for download and is only used by Microsoft. The function of this tool are summarised below: :
 - First, the tool utilizes a vulnerable version of Internet Explorer to visit servers on the internet.
 - Secondly, if there are any attacks or attempts on the system, the system stops the current interaction and re-visits the previous website with a fully-patched version of Internet Explorer.
 - Finally, after visiting the malicious website with the fully patched Internet Explorer, the attacks and attempts will be saved for further analysis.
3. **UW Spycrawler:** is a high-interaction client Honeypot developed at the University of Washington by Moshkchuk et al. [93]. It is based on the Mozilla web browser. As with other high-interaction client Honeypots, the UW Spycrawler monitors the state of the operating system by monitoring the file system, processes and the registry. The tool also detects other changes such as crashes by the web browser. This tool is not available for download.

2.3.6 High-interaction client Honeypot challenges

The technology and ideas behind the high-interaction client Honeypot help security researchers and other organisations to study web-based malware, their techniques and motivations. There are still challenges left to solves listed below.

- **IP tracking:** This technology is used widely in web-based tools to track the IP addresses of visitors; one major exploitation tool using this is the MPack web-based application. According to Seifert [85], MPack provides the capacity to track a visitor's IP address and affects the visitor only after a number of visits have been made. If the client Honeypot visits the malicious website running Mpack for the first time, then no malicious behaviour will be detected and it will assume the site is clean. However, if the client Honeypot makes a number of visits, then malicious behaviour will be seen.

Therefore, the first challenge is to solve this problem to achieve a better result without false negatives.

- **Geolocation-dependent:** This feature affects only visitors from certain countries and behaves normally with visitors from others. This feature is provided by malware web-based malware tools.
- **Obfuscation:** This involves hiding code within the web page. The malicious code is encoded in such a way that the signature database will not detect it. Fig 2-6 shows an example of obfuscating malicious code [85], it is able to avoid detection as it is encrypted and it will decrypt itself in runtime and effects the visitors of the web page

```
<script
language=javascript>document.write(unescape("%3CScript%20Language%3D%27JavaScript%27%3Edocument.write%
28
%20unescape%28%27%253C%2573%2563%2572%2569%2570%2574%253E%25
...
%252A7Khtzsyjw%252A7Knsij%257D3umu%252A77%252A8J%252A8H%252A7Knkwrj%252A8J5%27%29%3B%
3C/Script%3E"))</script>
```

Fig 2-6: Example of obfuscating malicious code [85].

Fig 2-7 summarises the two types of client Honeypots along with their advantages, disadvantages and tools that support each type.

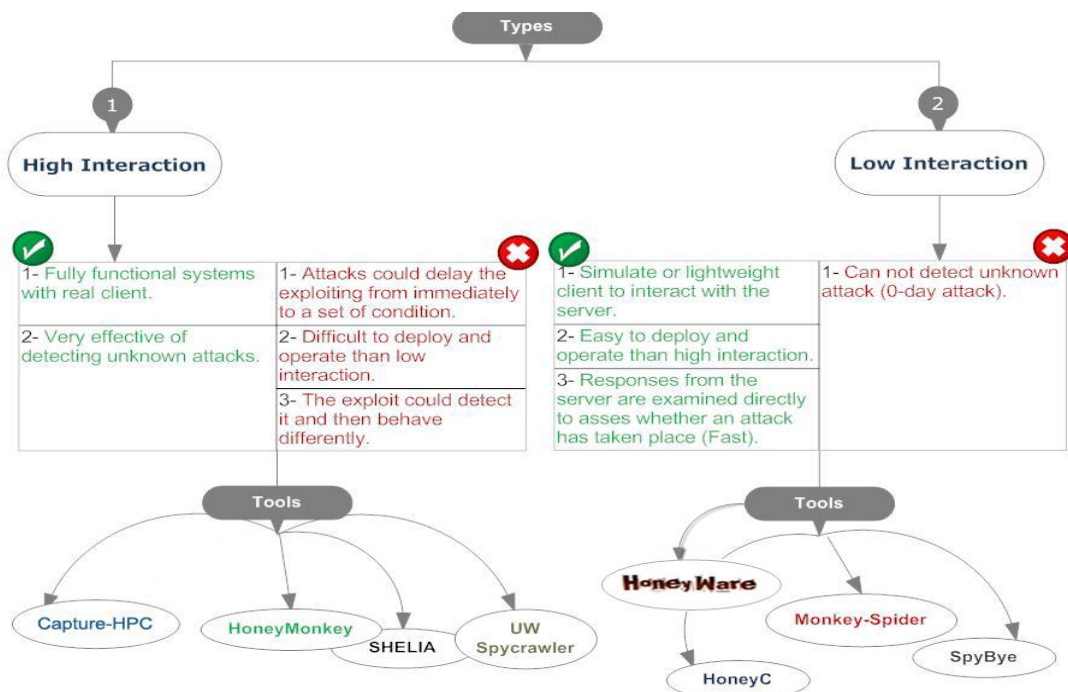


Fig 2-7: the two types of client Honeypots, their advantages, disadvantages and tools.

Chapter 3

Literature review

In this chapter, we discuss and present other approaches in the area of malware analysis and detection, we also describe how our approach differs from them.

3.1 Malware analysis

Malware is a short form for malicious software and “malicious code is any code added, changed, or removed from a software system in order to intentionally cause harm or subvert the intended function of the system” [7]. The goal of the attacker is to place malware into a user’s environment and then collect important information, such as credit card data, usernames and passwords. The malware analysis needs to understand how the malware under consideration works and then develop a patch in order to protect against that malware. There are two main types of malware analysis: static and dynamic analysis.

3.1.1 Static analysis

This type involves analysis of the malware without executing it. The analysis starts firstly by collecting information from the malware such as disassembling the code and collecting header and body data from the malware portable executable format (PE). This will give a complete view to the malware code while analysing it. One of the first approaches to detect malware is to compare an infected application against a clean copy of the same application or a formal specification to detect the malicious code [8] [9]. Lo et al. [10] developed a tell-tale sign that will be raised when the target application matched pre-defined actions such as a

malware writing an EXE file, which they stated that linkers and compilers could only perform such action. However, the attackers use encryption and obfuscation to hide their malicious code and make it harder to analyse it by automated applications or security experts.

Christodorescu and Jha [11] proposed a framework and built a tool called SAFE (a Static Analyzer For Executables) that is able to find malicious patterns in EXE files. They have also developed an obfuscator for executable code which supports transformations such as code transposition or instruction substitution. Christodorescu et al. [12] have developed their approach further by building the tool on top of IDA Pro [13] disassembler to get the flow graphs and produce an intermediate representation which produces a template for the code to pass it to the malware detector algorithm, and then developed their approach further [14] [15].

Other approaches used data mining with static analysis to detect malware. Matthew et al. [16] proposed a framework based on data mining that found malicious patterns in known malware by using automated static analysis techniques. They then detected new malware based on the previously discovered malicious patterns. The data mining techniques with static analysis are divided into four main categories [17] [18]:

- **N-grams:** it involves a sequence of hexadecimal strings extracted from an executable file. Various literature reports such work, e.g. [19] [20] [21].
- **Assembly Instructions:** Involves assembly instructions extracted from the executable disassembled file [22].
- **API/System Calls:** the application programming interface (API) is a specific method that a service provides to communicate with other applications. In this type, the system calls by the application are collected and then determine any malicious actions by looking for patterns. There are various research works in this area such as [23] [24].
- **Hybrid features:** involves using more than one feature to collect data about the code such as: assembly instructions, DLL usage or n-grams, such as [25] [26]

There are a number of publications that show the obfuscation of executable files make it harder to disassemble them such as [27] [28] [29]. Furthermore, malware are able to change

the application by using polymorphic and metamorphic techniques such as [30] [31]. As Kaspersky [32] stated that recent malware uses server-side polymorphism which changes some sections of the code whenever pre-defined executions are reached. Finally, based on these limitations of static analysis, we base our work on dynamic analysis which monitors the behaviour/actions of the malware on the system without a need to decrypt the file.

3.1.2 Dynamic analysis

This type is different from static analysis as the code is analysed while executing. The code is executed in a controlled environment and then its behaviour is monitored in real-time. These changes and behaviour are then analysed and not the program code. This is one of the benefits of dynamic analysis as it is able to monitor the code even if it used obfuscation or self-modifying techniques.

The scenario of dynamic analysis is that it firstly executes the program in a virtual machine that emulates a real operating system and then monitors the system in real-time to detect any changes caused by the program such as File system changes, running or killing a process or establishing a new network connection. After a pre-defined time-out, the system will shut down and restore to its clean state. There are two types of controlled environments: virtual machine and PC emulator. A virtual machine only simulates some parts of the operating system and shares some of the host resources, such as CPU, whereas a PC emulator simulates the whole hardware architecture and operating system. There are a number of Honeypot systems that are able to execute malware and then monitor its behaviour, for example CWSandbox [33], Norman [34], Threat Expert [35], Capture-HPC [36] and Anubis [37]. We have used Capture-HPC to scan the most critical part of the system such as File system, Process, Registry and Network. However, our malware behaviour analysis tools can be applied to any existing client Honeypot system and can generate state machine signatures and then pass these to our clustering and prediction algorithms. However, most Honeypot tools suffer from a limitation in that malware can take advantage of them and behave differently while scanning. One way they can do this is through a single path of execution, which means that some malware behaves differently in different environments or at different times; this cannot be identified as most Honeypot systems scan and extract the malware behaviour just once. Therefore, Moser et al. [38] proposed multi-path executions which explore different execution paths while scanning the malware. Furthermore, virtual machine-based Honeypots

suffer from anti-VMware techniques [39] [40] [41] [42] by which attackers can detect malware running under a virtual machine environment, and so change their behaviour to a benign one. This can be solved by the correct implementation and by configuring the virtual machine environment to hide its fingerprint. In addition, Sun et al. [43] introduced an automated technique to discover the malware trying to hide detection from a virtual machine, even if the malware is packed. In this work, we make the assumption that our scanning engine, Capture-HPC, is already implementing those techniques, as Sun et al. [43] who proposed a solution to extract the real behaviour of the malware.

One of the first approaches to use dynamic analysis in a controlled environment was that of Bailey et al. [44]. They firstly scan the malware in a VMware virtual machine [45] with a BackTracker system [46], a system developed by King and Chen which is able to trace malware activity in the system along with its activities in a dependency graph and most importantly differentiate it from the benign activities undertaken by the operating system or other applications. Furthermore, they then represent malware behaviour activities in a special representation that groups the total number of processes, file system, registry and network activities by malware. Finally, they use a hierarchical clustering algorithm to group similar malware together based on their behaviour to build up behaviour groups for malware. Our work differs from this approach by demonstrating how malware behaviour is encoded and the clustering model used. We used state machine, as described in chapter 4, which is able to encode more malware behaviours such as the behaviour action, source and destination of the change. This data is essential in order to understand what malware actually changes on the victim's system, as well as to help other algorithms such as clustering to group malware based on the behaviour collected and encoded. The clustering model we used is based on FP-Growth, as discussed in chapter 5, which is able to group malware based on its repeated behaviour. Therefore, it can group more malware as evaluated in chapter 5 comparing it to the hierarchical clustering approach.

Rieck et al. [47], scan malware behaviour in a sandbox environment called *CWSandbox* [33], a Honeypot environment which is able to execute malware on a real operating system, and then monitor its system calls and produce a log file for the malware behaviour. After receiving the report from *CWSandbox*, they developed a malware representation called MIST (Malware Instruction Set) that converts the *CWSandbox* to its hash code, which includes a collection of hashes which represent malware behaviour with the source and destination of

the activity. Furthermore, they passed this malware behaviour in MIST representation to a classification and clustering model to group them into behaviour groups using the same clustering algorithm as Bailey et al.'s [48] hierarchical clustering algorithm with complete linkage as the distance measure, where the grouping is based on the maximum distance between items of each cluster.

Lee and Mody [49] introduce one of the first approaches of dynamic analysis, they introduce a behaviour-based classification model centred on machine learning. They scan malware in a sandbox environment and then produce an ordered sequence of malware behaviour; this includes the activity state, kernel function and action parameters. They use classification and clustering algorithms to analyse malware and group them based on the malware's behaviour. Their classification is based on CBR (Case Based Reasoning) as well as the addition of a clustering algorithm to scale it. The clustering is based on K-medoid algorithm which produces k clusters, where k is the number of output clusters.

Predisci et al. [50] presented a framework that is able to analyse malicious HTTP traffic and then automatically generate network signatures to detect activities of the malware. They differ from the previous dynamic analysis approaches as they collect more features from the HTTP traffic such as the total number of POST and GET requests and the average length of URLs. They use a single-linkage hierarchical clustering algorithm to group similar malicious HTTP traffic together in behaviour groups. One issue that faced Predisci et al. [50] as well other researchers using hierarchical clustering algorithm is that the clustering algorithm builds up a tree that starts from each cluster with one malware behaviour and then groups similar clusters together until they merge into one cluster. They need to adopt a *cluster validity* analysis which can determine the right number of clusters and then stop the clustering to output the most compact and well separated clusters [50]. They used Davies–Bouldin (DB), a clustering evaluation algorithm presented by Davies and Bouldin [135], as the cluster validity index.

Christian et al. [51] proposed a new method for clustering malware based on dynamic analysis. The motivation behind their clustering approach was to analyse local malware in Indonesia to help them gain a better understanding of the current local malware types and behaviours. They used two algorithms to cluster similar malware based on behaviours/characteristics. The clustering algorithms they used are K-means and Self-

Organizing Map (SOM). Their approach starts by scanning malware within a sandbox environment and then extracting malware features as XML output from the sandbox. Then they used the Weka data mining framework [52] to apply the clustering algorithm to the malware and to visualize the results using the self-organising map algorithm.

These data mining algorithms have been in use for more than 10 years to analyse malware based on dynamic and static analyses; they provide automatic methods for a better understanding of malware behaviour, types, classes and detection. An approach that used data mining with a dynamic analysis was taken by Gurutxaga et al. [53], who proposed using an unsupervised learning method to cluster malware through Hierarchical Agglomerative Clustering (HAC). They tested different distance measures to cluster malware based on dynamic analysis and found that average-linkage is the best distance.

Wang et al. [54] suggest using a clustering approach to cluster worms by using dynamic program execution. They use two classification algorithms: Naive Bayes and Support Vector Machine. First they scan a large number of malware executables (worms) and benign executables on a virtual machine and then they record and trace the system calls for each worm. They then apply the classification algorithms to the data collected from the system calls made by each worm to learn and understand the malicious behaviour of the worms. They then used the classification knowledge to detect the worms online.

Another approach for clustering and detecting malware is to use an Artificial Neural Network (ANN) algorithm. Stopel et al. [55] proposes a new classification method for worms based on ANN. Their approach starts by scanning the worms in a sandbox environment where they gather data such as calls to the TCP/IP or UDP/IP layers. Then they use ANN to cluster the worms based on their behaviour as well as using two other data mining algorithms, K-Nearest Neighbours and Decision Tree for comparison. They identified that the ANN approach is faster when detecting real-time worms and also had the ability to detect unknown worms.

Another study that uses ANN is conducted by Linda et al. [56], who propose an intrusion detection system using ANN for critical infrastructures such as SCADA (Supervisory Control And Data Acquisition) and nuclear plants. Their approach is to cluster the normal behaviour using two neural network learning algorithms – the Levenberg-Marquardt and Back-Propagation algorithms – and then to use these normal behaviour clusters to detect abnormal behaviour. They stated that the ANN clustering approach used overcame most of the common

problems of other clustering approaches, such as how to choose the maximum number of clusters or how to initialise the centres of gravities of clusters [56].

Sequeira and Zaki [57] developed a new system called Anomaly-based Data Mining for Intrusions (ADMIT) which acts as an IDS system. First they monitor the UNIX shell command data to create user profile behaviour and then use K-means as the clustering algorithm to group user behaviour. Their system aims to detect intrusions in real time with host-based data collection and processing.

Another approach using data mining in malware analysis is to create profile behaviour for malware which are then fed into an Intrusion Detection System (IDS). Generally, the types of IDS are misuse and anomaly detections. In misuse detection, the IDS builds a data set of known attack signatures and then compares with other data to detect attack signatures. Anomaly detection defines the behaviour of normal traffic and detects any abnormal behaviour. The IDS is able to detect malware if it knows its signature, and these signatures or normal behaviour need to be manually fed into the IDS in order for it to be able to detect malware. Therefore, data mining is used to automatically analyse new malware and then generate a behaviour profile to update the IDS database to detect new malware. For example, Sequeira and Zaki [57] use K-means algorithm to detect abnormal behaviour based on default user profile behaviour. The IDS is able to detect malware through its signatures, and then alert the network administrator to react, or through another method called an Intrusion Prevention System (IPS), which acts the same as an IDS but is also able to block malware automatically without the need to alert the network administrator. Fig 2-1 shows an overview of malware analysis approaches, the numbers in Fig 3-1 show the reference number for each approach.

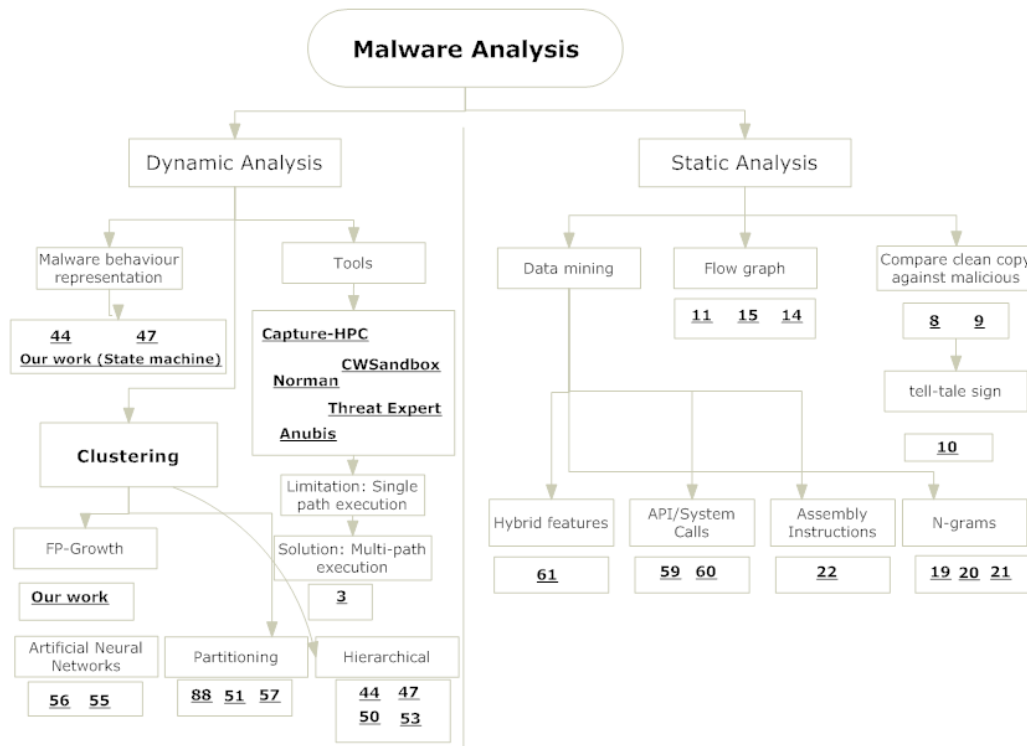


Fig 3-1: an overview of malware analysis approaches.

The right side of Fig 2-1 shows the different approaches used in static analysis, while the left shows dynamic analysis, which is what this work is based on. Dynamic analysis needs to be analysed by behaviour scanning tools such as Capture-HPC and CWSandbox, as shown in Fig 2-1. In order to cluster and analyse malware behaviour, we need to encode it in a defined manner. It can then be passed to different advanced algorithms, such as clustering, to analyse the results without needing to re-encode the data for each algorithm separately. Each scan engine such as Capture-HPC, as shown in Fig 4-7, or CWSandbox have their own representations and we need to have defined encoding that shows most of the behaviour characteristics. Therefore, we can use different algorithms to analyse the same defined encoding which is created from different scan engines. Our representation is based on the state machine, as described in chapter 4; other approaches, including MIST (Malware Instruction Set) [50] and Bailey et al. [44], are described in the next section. Finally, FP-Growth clustering is compared with the other clustering approaches used to analyse and group behaviour of malware using dynamic analysis. Table 3-1 shows a comparison between different clustering algorithms and the FP-Growth approach.

Name	Analysis type	Algorithm
Bailey et al. [44]	Clustering - Hierarchical	Single-linkage

Predisci et al. [50]	Clustering - Hierarchical	Single-linkage
Gurutxaga et al. [53]	Clustering - Hierarchical	Average-linkage
Lee and Mody [49]	Clustering - Partitioning	K-medoid algorithm
Christian et al. [51]	Clustering - Partitioning	K-means
Sequeira and Zaki [57]	Clustering - Partitioning	K-means
Linda et al. [56]	Clustering - Artificial Neural Networks	Levenberg-Marquardt and Back-Propagation
Stopel et al. [55]	Clustering - Artificial Neural Networks	Levenberg-Marquardt
Wang et al. [54]	Classification	Naive Bayes and Support Vector Machine
Our work	Clustering	FP-Growth

Table 3-1: a comparison between different clustering algorithms and FP-Growth approach.

Our approach based on FP-Growth clustering differs from all existing approaches for dynamic behaviour analysis. Most of the previous clustering methods used HAC, we have tested it previously with state machine and found that it suffers from a number of limitations. One of the limitations is that malware might make some changes amongst some of its behaviour in order to mislead the analysis engine (clustering algorithms) thereby group is similar malware in different behaviour groups.

By using FP-Growth we can detect similar state machine signatures and group them based on the common sequence behaviour states which enables us to detect similar malware behaviour, even in different parts of the state machine signature. FP-growth has the ability to mark each behaviour group with a label and can deal with an arbitrary length state sequence. The label represents frequent states within each group. By using this label, we can identify similarity between groups and then use the label to match with new data. Each sub-group will have a label to identify the common states within each group, which indicates why each sub-group state machine is similar. By using this clustering approach, we can add new behaviour groups to the already discovered groups. The benefit is we will only need to re-cluster and match the new and old group labels, and not have to re-cluster the entire data set.

Furthermore, most of the clustering approaches identified previously in Fig 2-1 are good at grouping similar attack sequences that have one or a few malware behaviours. However, our clustering model is able to identify several aspects of the state machine, such as the behaviour

sequences that are responsible to exploit a web browser by using a particular vulnerability, or a series of states that are similar from one web-based malware to another. While developing the clustering model for malicious web pages, we know that this type of attack is different from traditional attacks such as virus or Trojan horse, as they first need to exploit a vulnerability on the web browser or operating system to gain permission to access the system and enable the attacker to install and execute different malware. The state machine representation helps by detecting these sequences of changes in real-time and in order and can therefore distinguish between the different types of malicious behaviour, such as exploiting web browser vulnerability, installing and running one or more malware on the system or monitoring the traces of malware behaviour. Furthermore, the clustering model uses these benefits of the state machine representation by grouping them according to common sequences, and can therefore detect practical web browser exploits, Trojan horses, viruses or other malicious techniques being used by a 0-day exploit. We describe and discuss the benefits and methodology of the clustering model in detail in chapters 5.

3.2 Malware modelling and representations

Schneier [58] stated:

“What we need is a way to model threats against computer systems. If we can understand all the different ways in which a system can be attacked, we can likely design countermeasures to thwart those attacks. And if we can understand who the attackers are — not to mention their abilities, motivations, and goals — maybe we can install the proper countermeasures to deal with the real threats”.

The security community has been researching the most suitable method to identify attacks in order to show their exploit methods and behaviour for many years. The basic idea behind modelling an attack is to show the attack trace, such as methods that the attack uses to exploit the system, including the use of network, application and operating system. In general, there are two main types of attack models: Tree and Graph structures [59] [60] [61]. Furthermore, there are a number of modelling techniques which fall under these two types; these are discussed below. However, some researchers have not used modelling to encode and model

the attack behaviours and methods [44] [47]. We believe that the benefits of using attack modelling within the framework for analysing attacks is an important and efficient way to understand the structure of the attacks. It also helps to have pre-defined coding from different scan engines which can then pass to other algorithms by different formats, compared to textual representation of attack steps that does not offer such advantages and benefits. The main types of attack models are shown below:

- 1. Attack tree:** this is a hierarchical structure in which the upper level of each stage is responsible for actions below it; the top level of the tree is the start of the attack and then the tree draws the steps of the attacks under the cause of that action. Every tree starts with one action, and then each node of the tree represents one step in the attack. Furthermore, OR or AND logical operations can be attached to each node, as well as a value or probability [59]. This approach was first presented by Schneier [58], who wanted to build a model that could show all the steps in an attack as well as measure the risk and cost from a practical attack. After the idea was proposed by Schneier, many researchers adopted the attack tree methodology and it has since been used widely. One of the approaches is conducted by Ten et al. [62], who used the attack tree to model the supervisory control and data acquisition (SCADA) to detect vulnerabilities in an embedded system. The attacker is able to control the electrical power system remotely using supervisory privileges, and they use these to attempt to detect these kinds of vulnerabilities. As the previous approach shows, the use of an attack tree is able to define a complex system, such as an electrical power system, and has also been used to test the security of nuclear digital instrumentations by Khand [63]. He presented a methodology that uses the attack tree to counter the measure of the security adopted by the nuclear digital Instrumentation and Control (I&C); any breaches of this could cause catastrophic impacts to the employees and to the environment [63].
- 2. Attack graph:** this presents attacks in a sequential order that begins with the start state representing the exploit of a vulnerability and attack. Then, each state in the series represents a step in the attack; the attack transfers from one state to the next through a transition that is caused by the previous state, leading to the effect seen in the next state; the final state shows the end of the attack. The attack graph orders the attack steps based on the time at which they were caused. Many studies have used and extended this method as it shows attacks in a simple way. Most malware can be presented using this method. An

example of this method is the use of the state machine model to show web-based behaviour as discussed in chapter 4.

Finally, there are other approaches that analyse and detect malware and vulnerabilities using their own unique representations of attacks. Bailey et al. [44] propose a representation that shows malicious behaviour, their encoding involves combining the total number of changes on the test machine such as the total number of changes to processes, file system, registry and network. The representation is mainly focused on the number of total changes on the system but not the events and transitions between the states of malware activity.

Another approach which represents malicious behaviour is called MIST (Malware Instruction Set) proposed by Rieck et al. [47]. They have used a hash representation that show malware behaviour in a sequence of hashes which represent the system calls as shown in Fig 3-2. The hash representation needs to be specified by the algorithms in order to group them correctly based on the meaning and order of the encoding, while the human analyst cannot read and understand what malware behaviour is without linking between the hashes and the system call terms. Our state machine encoding is suitable for use in automatic algorithms as well as the human analyst to read the behaviour sequence and understand what the malware intends to change without explanation of system calls.

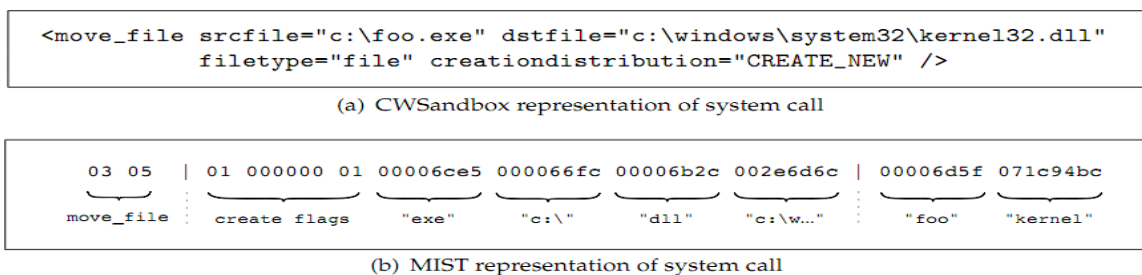


Fig 3-2: MIST representation [47]

Previous approaches of encoding and representations of malicious behaviour involve a sequence of strings that represent the actual changes such as writing a new file or inserting a new value to the registry. However, our state machine model is event driven as at any time during the interaction with web based malware it will be in a state and event. Furthermore, the model will save the change details in each state and when a change occurs it will trigger an event, which will lead to a state change. Also, the automated state machine encodes malicious activity in real-time and builds the state machine signature for the web-based malware, making it easy for the end-user to get an overview of the behaviour as well as the

ability to use it with a complex algorithm such as clustering and prediction algorithms. There are several intrusion detection systems that use the state machine as a way to encode behaviour of attacks [64][65][21].

The state machine is different to a simple string comparison. The state machine is an encoding model that converts the client Honeypot log file to the state machine graph representation that shows the behaviour of the web-based malware. The state machine itself is driven by the malicious web page causing a state transition (through an event). Our model is able to represent changes caused by web-based malware in an ordered sequence of malicious behaviour as shown in chapter 4.

3.3 Malware detection

The malware detector helps by preventing malware from the end-user system. Malware detectors differ in their ability as some are able to detect both known and unknown malware, while others are only able to detect known, previously-seen malware. Generally, there are three main types: Signature, Anomaly and Specification based detection [66] [67]. Each type of malware detector offers Static, Dynamic or Hybrid detection. Static detection mainly involves identifying malware based on the properties of the malicious file, such as assembly instructions or n-grams (as discussed in section 2.1). On the other hand, dynamic detection is based on the behaviour of malware, such as the activities and changes it makes to the file system or network. In our study, we focus on the dynamic approaches for each detection method as we think that it provides better detection than signature, such as detecting unknown malware, and most of the static techniques can be overcome through obfuscation and encryption which makes it hard to unpack and analyse malware. The three types are explained below.

1. **Dynamic anomaly-based detection:** The detection engine starts in the training phase in order to learn the behaviour of the program and to build normal behaviour. It can then monitor the program during execution and compare it to the normal behaviour; through that it can detect abnormal behavior, which might be malicious behaviour or activities.

There are different approaches concerning which parts of the program are monitored to build a normal behaviour template for the anomaly based detection system. One of the first approaches was conducted by Hofmeyr [68], who built an anomaly detection

approach that is able to detect abnormal behaviour in privileged processes. He stated that using privileged processes is a better method of detection compared to the user-mode, as most privileged processes have a short and predictable sequence of system calls, unlike the unpredictable behaviour of user-mode processes. He examined short sequences of system calls from the privileged processes in order to determine normal behaviour. Furthermore, in order to detect similarities between the new sequences and the pre-defined ones, he measured them using the minimal Hamming distance [68]. A similar approach to this was taken by Wang and Stolfo [69], who proposed using payload-based anomaly detection which involves building normal behaviour by scanning the application payload of the network traffic. Their detection tool is called PAYL (payload-based anomaly detector). In their unsupervised training phase they create a “normal profile” from the byte frequency distribution of application payload on the network, and then they use the Mahalanobis distance [69] to measure the similarity between the incoming data (unknown) from the application during detection against the pre-defined profiles and then flagging an alert if it exceeds the threshold. Furthermore, they use a clustering approach to find two similar profiles using the Manhattan distance.

Mahoney [70] developed a Network Traffic Anomaly Detector (NETAD) that is able to identify malicious behaviour in network traffic. He stated that the first few packets are important in order to detect most attacks against systems and so filters incoming IP packets using the first 48 bytes from the packet, starting with the header. First the model uses the most common network protocols, such as FTP, TCP and HTTP and then it compares these to the incoming 48 bytes to detect any unusual behaviour and will raise an alarm if necessary.

- 2. Dynamic specification-based detection:** this detects malicious behaviour by determining similarity during execution against a pre-defined set. It differs from the previous detection method (anomaly) as it is able to discover malicious behaviour once it matches a pre-defined set of behaviour rules. Masri and Podgurki [71] developed a tool called Dynamic Information Flow Analysis (DIFA) to detect intrusions and attacks against application software. They specified flow signatures for known attacks and found that the tool is able to detect these attacks. Furthermore, they also used automatic cluster analysis alongside their tools in order to detect unknown attacks. Their tool is designed to detect attacks against Java applications.

Another approach is used by Sekar et al. [72] who introduced a new translation language called ASL (Auditing Specification Language). They converted known vulnerability and attack behaviour to ASL and then into C++ class definitions; this C++ class is then added to the detection system in order to monitor the application system calls just before they are passed to the operating system kernel. These are then compared against the pre-defined ASL specification to detect any malicious behaviour.

Another specification-based detection was developed by Ko et al. [73], who proposed a method to detect exploitation of vulnerabilities found in privileged processes. Their detection method monitors privileged processes through audit trails and then compares them to known specifications of vulnerabilities within such processes. They have demonstrated their detection method on a Unix environment and proved that their approach works well in detection of these kinds of malicious behaviours, except for some privileged processes that use authentication, such as login and rlogin.

Finally, there is another specification-based detection used in cross-site scripting (XSS). Krida et al. [74] presented a new tool called Noxes, which is a Windows-based service that runs in the background of a user's system. It acts as a web proxy and so will receive all the HTTP connections to the web browser and then pass or block the connections based on the policy of the tool. The policy is simply a specification of pre-defined XSS attack behaviours. Noxes aims to protect against XSS on the client-side, as most current solutions to protect against XSS are server-side, which might suffer from delays in fixing them and so might harm the users (visitors).

- 3. Dynamic signature-based detection:** This detection method relies on using real signatures of malicious applications in order to detect them. Signature-based detection is one of the oldest methods used to detect malware, and used by the majority of security products. Ilgun et al. [75] presented a detection system called State Transition Analysis Tool (STAT) which is able to detect malicious behaviour by comparing the monitoring behaviour to a pre-defined STAT (known attacks in the format of a STAT). Table 3-2 below shows a comparison between the three malware detection systems [76] [67].

Detection type	Detect unknown attacks	Regularly updates	False alarm
Signature	no	yes	low
Specification	no	yes	low
Anomaly	yes	no	high

Table 3-2: a comparison between the three malware detection systems [76] [67].

3.3.1 Hidden Markov Model (HMM) for malware analysis and detection

HMM has been used in malware analysis and detection research tools for the last few years. Some approaches use it for clustering similar malware, while others use it as a classification tool to match the new pattern (i.e. from applications or algorithms) with pre-defined patterns. One of the first approaches using HMM was proposed by Le [64] who presents an anomaly-based detection system able to learn normal behaviour through a Markov chain model using a Basic Security Model (BSM). The BSM is an extension for monitoring activities on the Solaris operating system. The BSM generates 284 different audit event types while monitoring the system and so a Markov chain model can be built of the temporal behaviour by building a transition probability matrix from the normal audit events generated previously [64]. Le conducts an experiment to distinguish between normal and attack behaviours and stated that his approach successfully distinguishes between the two types of traffic with a rate of 100%.

An interesting study was conducted by Radosavac and Baras [65], who proposed a detection model based on HMM classification that encodes attack behaviour in a finite state machine (FSM), in the same way as our approach represents web-based malware, as described in chapter 4. Their model is only able to detect known attacks and so they encoded several buffer overflow attacks into their finite state machine encoding and classify them using HMM. They use the BSM on the Solaris operating system to monitor the behavior and record this in a log file. Furthermore, they filter the log files to get malicious behaviour by dividing each attack into 100 sequences and then using a sliding window approach to filter out normal behavior from malicious one. As stated previously, their approach is only able to detect known attacks they have modelled previously and so their algorithm cannot detect unknown attack behaviour in the BSM log file. In the classification phase, they use the HMM to classify buffer overflow attacks FSM representation into a group. If the classification

algorithm is not able to find a full match between the current attack and previous attacks then it will calculate the highest likelihood for the whole attack behaviour in order to find a relevant group. They assume that the attacker cannot change their behaviour by adding normal behaviour between malicious actions in order to avoid detection. However, we disagree with them as there are many methods the attacker can use to mislead their classification approach, as described in section 2.1; the attacker can mislead the classification and clustering behaviour using obfuscation, such as changing the order of malicious behaviour, adding unwanted behaviour between actual malicious behaviours or adding normal or abnormal behaviour to mislead the classification process. We use the FP-Growth clustering approach which is able to identify malicious behaviour even when the attacker uses some obfuscation techniques. Furthermore, our work differs from their approach in that our HMM model is used for prediction, as discussed in chapter 6, not for classification. The benefit of this is that the prediction algorithm is able to expect the likely next activity of the attacks, even if the attacks are unknown, while their implementation of HMM is only able to classify known malicious buffer overflow attacks.

Another interesting approach that uses HMM with a finite state machine is developed by Radosavac et al. [21]. They propose modelling the buffer overflow attacks in a FSM and then using HMM to detect such attacks. Their approach is only able to detect buffer overflow which have similar behaviours between attacks. The method of a buffer overflow attack is to pass extra data to the maximum buffer value [21], and so they face the problem of misclassification of buffer overflows, as most of the attacks share similar behaviour and their classification model has a very low rate for detecting attacks which share similar behaviour. The basic idea of the main and sub behaviour groups in our approach is that we are able to have a main group for similar buffer overflows and then use sub-groups that are able to distinguish between similar attacks. Furthermore, their approach is only able to detect known buffer overflow attacks; it cannot detect unknown or obfuscated buffer overflow attacks, a limitation not found in our approach.

A recent work that is close to our research is proposed by Lee et al. [77], who propose an intrusion detection system based on multi-stage determination. The basic idea behind their approach is that after each stage of the intrusion signal, the HMM algorithm attempts to identify the current malicious sequence against the pre-defined ones in its database. They apply the HMM determination algorithm to the anomaly intrusion detection to detect network

intrusion. Furthermore, their determination algorithm, based on the HMM, is similar to our prediction algorithm; however, our prediction algorithm is different because it predicts the next activity from the current state, and is not restricted to identifying similar sequences only. An intrusion might have a different sequence from a pre-defined one and so we calculate the probability of the next state of the intrusion behaviour. Therefore, we are able to base predictions on similar state machines in the database, even if they are at different positions in the state machine sequence. We believe that by comparing such representations using HMM, our prediction model provides better identification of similar state machine signatures in the database. Moreover, if the prediction detects an unknown signature, it will try to identify its behaviour group by passing it to the FP-Growth clustering model which is able to find greater similarity between the state machine signatures or build a new behaviour group in case it is new malware to the database

Xiuqing et al. [78] proposed a novel intrusion detection technique that uses misuse and anomaly detection and is based on HMM. They use a Fast Adaptive Clustering Algorithm (FACA) to update the HMM dynamically with new attacks in the case of misuse detection and then update the misuse detection engine with normal and abnormal signals. After clustering using FACA, the Davies-Bouldin (DB) index is used to measure the distance between the clusters and then merge them if they are small and similar, which represents normal behaviour, or detect abnormal behaviour if the distance is larger. The system uses the clustering output to build the HMM and uses it with the anomaly detection system to detect abnormal behaviour.

Another approach towards using FACA algorithm is used by Xiuqing et al. [79], who propose an intrusion prevention system for network attacks based on the HMM (HMM-IPS). The system is different from the previous approach of Xiuqing et al. [78], as it is a prevention system and so is able to block abnormal behaviour. Furthermore, the system uses the FACA clustering approach to add new attacks to the HMM and the Kullback Libler (KL) distance measure is a measure between two determined probability distributions [128], to combine clusters if the minimum distance is reached. Finally, HMM is used in the prevention detection system to detect abnormal behaviour using the KL distance measure and then calculate the size of the sliding window to detect if the incoming sequence is malicious or not.

Another approach to using HMM was used by Tsai et al. [80]; they proposed an intrusion detection system that sniffs from multi-agent sensors on the network and analyses the normal, intrusion and misuse signals with a multi-dimensional HMM. In their system they have three different main models: the detection, expert and console models. The detection model is responsible for detecting intrusions using a fuzzy inference rule; the expert model is focused on analysing the data detected based on HMM and using cross-correlation and autocorrelation measurement to detect the normal, intrusion and misuse traffic; finally, the console model is responsible for controlling the whole system performance and controlling the agents and sensors for monitoring and generating reports.

An improvement to the previous approach is proposed by Tsai and Hung [19] who presented a similar intrusion detection system which was based on the multi-dimensional HMM, and had the same three models: the monitor, track and analysis models. However, in this approach they have used a Honeynet deployment for their system instead of the multi-agent sensors used in the previous approach. They stated that the Honeynet adds benefits of tracking and recording the exact behaviour and accessing trails of the intrusions. They then used the ant colony algorithm with the deposit of different pheromones to analyse the intrusion traversal. In their deployment they collected all the data from security monitoring tools, such as Firewall, antivirus and SQL queries, and then passed them to an analysis model. They used the multi-dimensional HMM and used cross-correlation and autocorrelation measurement to integrate and advance analysis. Finally, as in the previous approach, they used a fuzzy inference rule to identify and recognise intrusions.

In our work we are using the HMM to predict the next activities of a given malicious behavior sequence. This section has examined the different uses of HMM in malware detection such as [65] [21], and shown that most of them use HMM as a detection model to compare between a given attack sequence and pre-defined attack sequences in a database. Lee et al. [77] proposed using multistage detection and run HMM in every stage of attack to determine similar attack sequences. In our use of HMM, we employ an algorithm for prediction of future behaviour of a given state machine sequence based on the previously known information from the state machine sequences in a database. Therefore, it is able to predict future activities of a given web-based malware attack sequence in each stage of the attack.

Moreover, our HMM prediction model uses an FP-growth clustering algorithm to help detect web-based malware. The HMM will pass the unknown state machine signature to the clustering model to find more relationships between the given web-based state machine and the others in the database. It can therefore group it into a similar main or sub-behaviour group, or create a new behaviour group if it is a new attack sequence; this will improve detection in the future. Finally, it re-trains the prediction and classification models each time it adds a new state machine to the database in order to improve its performance.

Chapter 4

The web-based behaviour system has been developed based on malicious behaviours by generating a state machine signature from a malware's behaviour and then grouping these signatures into main and sub-behaviour groups. The future behaviour for a running web page is then predicted, based on the relationships and knowledge of behaviour groups, and how each malicious web page is similar to others. Our objectives from this model can be summarised as follows:

1. Monitor the web-based malware behaviour and generate a signature for its behaviour in the victim's system.
2. Visualise the behaviour of the malicious web page in a way that is easy for humans, as well as the ability to use it in automatic or data mining algorithms to reveal more information and allow more understanding about malicious activities.
3. Group the web-based malware into behaviour groups based on the similarities in their malicious behaviour on the victim's system.
4. Get more similarities and relationships for each behaviour group by creating sub-groups that will show a greater number of relationships between the malicious web pages that cause similar malicious behaviour.
5. Predict future behaviour for any given malicious web page, and then block it from causing risk to the end-user's system. In addition, use the prediction state to get more knowledge about malicious behaviour by using the clustering model to reveal greater understanding, such as in which behaviour group it belongs or what the other possible risks it might cause to the end-user's machine by comparing it to malicious web pages in the same main and sub-behaviour groups.

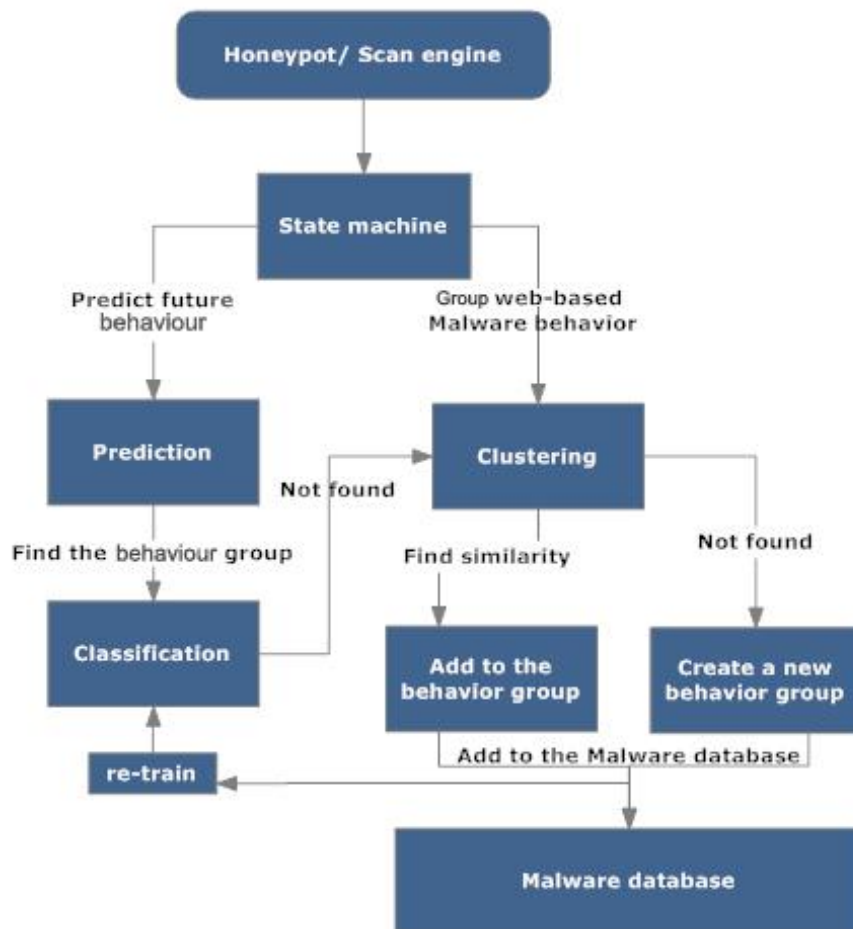


Fig 4-1: Web-based behaviour system.

Automated State Machine

This chapter describes the automated state machine that was built to encode malicious web page behaviour as a sequence of activities. Client Honeypots visit and interact with suspect web sites in order to detect and collect information about malware. Furthermore, each malicious web page is able to perform a number of activities on the victim's system. This chapter will show the benefits of using an automated state machine in conjunction with a client Honeypot. The automated state machine, when applied to a client Honeypot as described here, adds to the Honeypot technology a number of benefits when analysing malware, and tracing the details of malware steps. The state machine provides a powerful framework for analysing malware activity and encoding the results.

The state machine creates a unique signature for each malicious web page behaviour. These states describe malware behaviour in detail, from the state, such as File system, Registry or Process, to the source and destination of the change. By using the state machine signature, different malicious web page behaviours can be compared and analysed to find their similarities and differences.

Section 4.1 describes the state machine concept and the theory of how it is used. Section 4.2 discusses the use of the client Honeypot with the state machine and how they can be integrated. Section 4.3 shows the monitoring and detection tools. Section 4.4 provides an overview of the drive-by download attack, while Section 4.5 gives an example of such an attack. Section 4.6 gives an example of the automated state machine, and how a state machine signature can be created for any malicious web page. Section 4.7 evaluates the automated state machine.

4.1 web-based attack behaviour models

The attacker's aim is to compromise the victim's machine and steal valuable data, or use it as a machine to attack others on the local network or the Internet. One of the most successful methods of attack nowadays is a web-based attack. As Wüest [104] stated, there are more than 600 known vulnerabilities identified per year across all browsers; this opens the door for attackers to compromise the end-user's system through drive-by download attacks. Drive-by download attacks increased by 93% in 2010, and in my opinion, they will increase further in the next few years, because many of the new production tools are web-based, and this increases the use of web browsers, which have exploitable plugins and extensions. The drive-by download attack scenario is more effective and easier to carry out, as the attackers only need a way to send malicious web pages to the victims. When the victim accesses the malicious web page, the attacker's code checks his web browser, plugins and operating system vulnerabilities and reports these to the attacker. Section 4.2.3 shows an example of a drive-by download attack that exploits the IE vulnerability, CVE-2005-0553 [105].

4.1.1 The motivation of web based attacks

The attacker targets the web browsers and if successfully exploited, takes control of the system and may change the state of the system to a malicious state. In general, the web-based attack is divided into a series of steps and actions, and in the end the attacker will be able to

take control of the victim's system. Our aim in modeling a web-based attack is to build a structure that demonstrates the steps of malicious web pages as states, and the order in which the states occurred. Furthermore, within each state, the model will save the state's behaviour, demonstrating, for example, the source and types of changes. There are a number of advantages to this model:

- 1- It provides a standard format to describe the actions of web-based attacks and their timelines.
- 2- It distinguishes the states responsible for exploiting the web browser vulnerability from the payload and the malicious activities after the system is compromised.
- 3- It supports the ability to store each behaviour type and action within the corresponding state, as well as the ability to adopt new system states to the model.
- 4- It provides a comparison between two or more web-based attack models, which reveals new relationships between them.
- 5- It helps in understanding the state's sequences and which ones are likely to be the next state, in real time, using a prediction algorithm.

4.1.2 The state order

A malicious Web page represents an ordered set of behaviour sequences, which performs specific actions on the users system. This behaviour sequence is ordered by time and usually starts with steps to exploit a vulnerability on the victim systems. It then conducts exploit payload, the attack actions, to affect and change the system state. However, there are a number of times that the order of the behaviour sequence can possibly be changed:

- Software fault: The malicious Web page is monitored by a high-interaction client Honeypot system. At some time, the monitoring system cannot detect all changes in real time or log some of the changes in a different order, causing errors in the analysis.
- Avoid detection: The attacker can change the order of the exploit to avoid detection from a signature-based software, so the changes can cause the same actions but in a different order. Further, they can cause unwanted behaviour between the exploit steps to avoid detection.

It is not easy to determine if changes to the state order would result in the performance of the same behavioural actions. This adds a level of challenge for malware analysis that aims to determine whether there are states that perform the same malicious behaviour (impact on the victim's system) on different orders. We focus on finding the key malicious states involved in the attacks, in order to determine which states are exploiting vulnerabilities in order to affect systems, and which states are responsible for delivering attack payloads. Attackers could potentially change the order of a state to interfere with the analysis of an attack, which would add limitations to our model. In our work, we base our monitoring and logging on a high-interaction client honeypot system (Capture-HPC), and assume that the monitoring software logs all of the behavioural steps in real time without any errors, and that the state order is correct.

4.1.3 The challenges

The modeling of web-based attacks is useful for understanding attack types and methods. However, there are a number of problems and challenges facing the modeling of such attacks. The web-based attack that we are aiming to model is the attack that compromises the victim's system by exploiting the operating system, web browser and its plugin, through the web browser. One of the challenges is that web-based attacks contain a number of activities, and each series of steps and activities is responsible for a different malicious behavior. These malicious behaviours might begin by exploiting the system through a known or unknown vulnerability, and then carry out other malicious activities, and might install malware on the victim's system. Furthermore, the web-based attack affect not only one part of the system, such as the file system or the network, but it attacks different parts of the system, and these parts need to be monitored and scanned. In case of buffer overflow vulnerability, the attack will first exploit the vulnerability from the memory, and then it might move to the processes in order to launch a malicious file or insert malware into the file system. Therefore, we have to understand which parts of the system need to be monitored in order to detect any web-based attacks. Moreover, in each part of the system, there are a number of actions that might occur, for example an attacker might insert, delete or modify a file in the file system.

Finally, we need to understand how the malicious web page operates and behaves, and therefore detect and predict its malicious activities before or while it is attempting to attack the victim's machine. Huang et al. [107] have summarised the general steps of attacking the

user through drive-by download attacks in Fig 4-2. It shows four main steps of attacking visitors: Firstly, they generate the malicious URL for visitors on different web sites for example Facebook or Twitter, secondly, when the visitor visits the malicious web page this will exploit their web browser from the attacker's exploiting server. Thirdly, the attacker inserts malware into the visitor's system and finally the attacker is able to take over the visitor's system.

The next sections describes the methods of modeling web-based attacks, as well as how we solve the previous challenges and problems to build a model that describes and demonstrates web-based attacks in a simple but detailed manner.

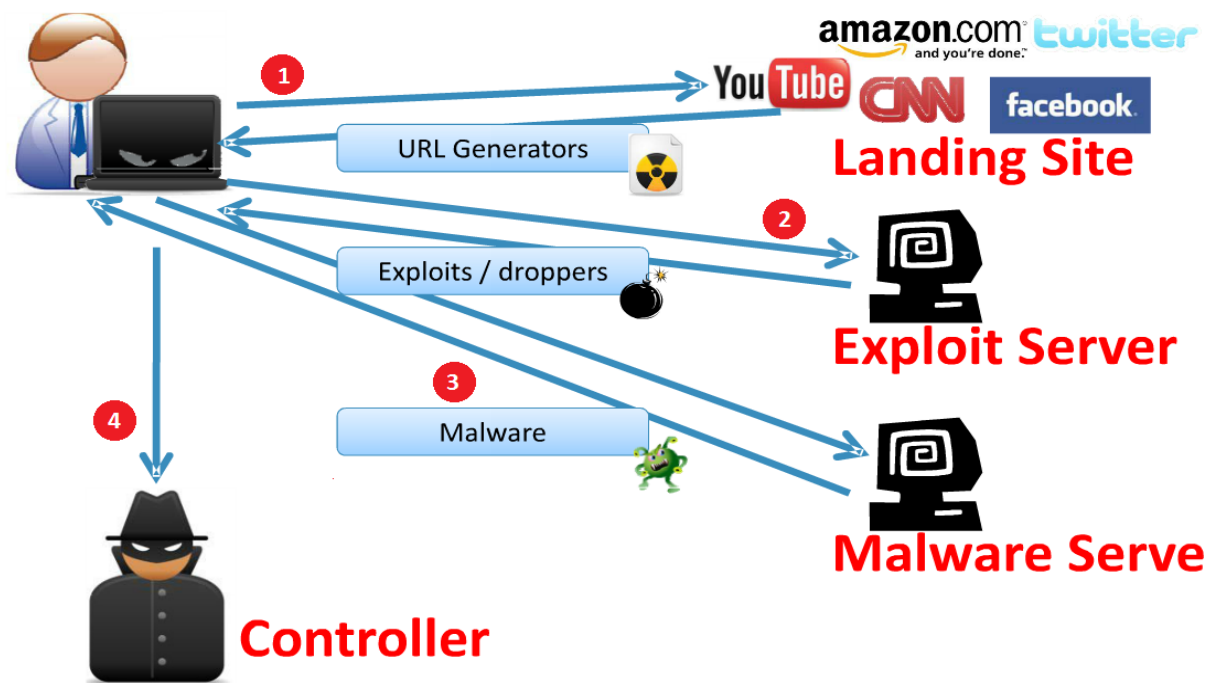


Fig 4-2: the drive-by download attack scenario [107].

4.1.4 web-based modeling methods

There are a number of modeling methodologies that assist modeling the structure and methods of web-based attacks. These models help to understand the possibility of the threats, as well as measuring the risk, cost and attacks. As mentioned earlier, in Section 2.2, there are different modeling methods, such as attack trees, Petri nets and UML, and most of them can help to present and model complex attacks that involve modeling different attack paths and scenarios. In our work, we are going to model web-based attacks, and therefore we need to

fully understand what we are going to model in order to apply and select the best modeling technique to model our target, which is malicious web page behaviour. As discussed before, the malicious web page attacks the end-user's system through a web browser in stages, and each stage causes a transition from one stage to another.

Furthermore, the malicious behaviour steps should be ordered over time, in order to understand how it works and which state is responsible for causing the change to the next activity. Therefore, we needed to choose an attack graph, as discussed in section 2.2 in chapter 2 to enable us to visualise the steps in each web-based attack scenario. Furthermore, we based our model on a state machine, as this provides a simple way to represent and save the behaviour data in states, starting with the first state, which is the beginning of the attack, and from then it is event driven; any change should trigger an event within the current state and cause it to transfer to the next node, which represents another malicious activity on the system. The state machine provides a simple and easy way to model the activities of malicious web pages in sequential order. In addition, it helps us to easily extract the behaviour from the node for use with automatic analysis techniques, such as clustering and prediction algorithms. The state machine model has already been used widely as an attack and vulnerability model, as we discussed in Chapter 2.

Finally, our state machine model is a defined format of malicious web page behaviour and it provides the base for other complex algorithms and processes in order to derive more relationships from the database of the state machine signature, such as clustering and prediction algorithms.

4.2 State Machine

To summarise our automated state machine model, as described in Chapter 3; first, the model is divided into pre-defined malicious states, such as file system, registry and network. For each state, the model will save the action of the change, such as adding a new value to the registry, as well as the source and destination of the change. The transitions between the states are driven by malicious activity; for example, the state can be changed from the process state to the network state when the malicious web page opens a new network connection.

As mentioned earlier, there are several issues to discuss concerning the finite state machine, as well as the client Honeypot, in order to gain a better understanding. Some of these are: the

system states to be monitored, the transitions from one state to the next, the information saved with each state, and the tools for monitoring changes in each system state. The next section discusses the types of state machines and why we chose the finite state machine to model web-based attacks.

4.2.1 State Machine types

The state machine has been used widely to model different data, such as hierarchical finite state machines with multiple concurrency models [107], abstract State Machine Language for Attack Scenario Specification [108], and detecting Denial of service (DOS) attacks on VoIP systems by state machines [109]. There are different types and methods of using state machines, as mentioned in Chapter 3. However, we are going to model web-based attacks, therefore we need to select the appropriate type to model and present all of the steps and paths of malicious web page behaviours.

Web-based attacks compromise the system and change the state of the system via web browsers. The parts of the system are limited and finite and therefore they need to be monitored in real time in order to detect any change that may occur due to a malicious web page. Therefore, we are going to use a finite state machine, as it models a finite number of states and these states are known to us.

4.2.2 Finite state machine modeling for web-based attacks

A state machine is divided into a series of states and transfers from one to another through transitions generated by events. An example of an event is a user turning on the system: the system transfers from the “off” state to the “on” state. A state machine cannot be in two states at the same time, which makes it easy to query the state of the state machine. The basic idea of this model is to generate the state machine automatically according to the system state, and to build up a sequence of the system’s states from the first interaction between the Honeypot and the server until the end of the session.

The benefits of using an automated state machine with a high-interaction client Honeypot are:

1. The state machine is generated automatically according to the system’s states while it is working.

2. The state machine generated includes each action performed by any malware and saves the action along with its parameters.
3. For each client Honeypot scan, the state machine is generated afresh.
4. Each state machine contains all the system states while interacting with the external server, which helps to review the system states at a later time.
5. The map of the state machine for each interaction helps in analysing the malware and identifying its processes in order to develop a patch for it.

We used the finite state machine to encode finite and limited parts of the system. These are the four main states that are monitored. A diagram of the state machine is shown in Fig 4-3. This finite state machine can also be illustrated with a 5-tuple as:

$$A = \{S, \Sigma, \delta, S_0, S_5\}$$

Where

- The Σ alphabet input includes the finite number of events $\{e1, e2, e3 \dots, e18\}$.
- S is the non-empty set of finite states, $S = \{S_0, S_1, S_2, S_3, S_4, S_5\}$.
- S_0 is the initial state.
- S_1 is the File system state.
- S_2 is the Process state.
- S_3 is the Registry state.
- S_4 is the Ports state.
- S_5 is the final state.
- δ is the transition function to states, $\delta: S \times \Sigma \rightarrow S$

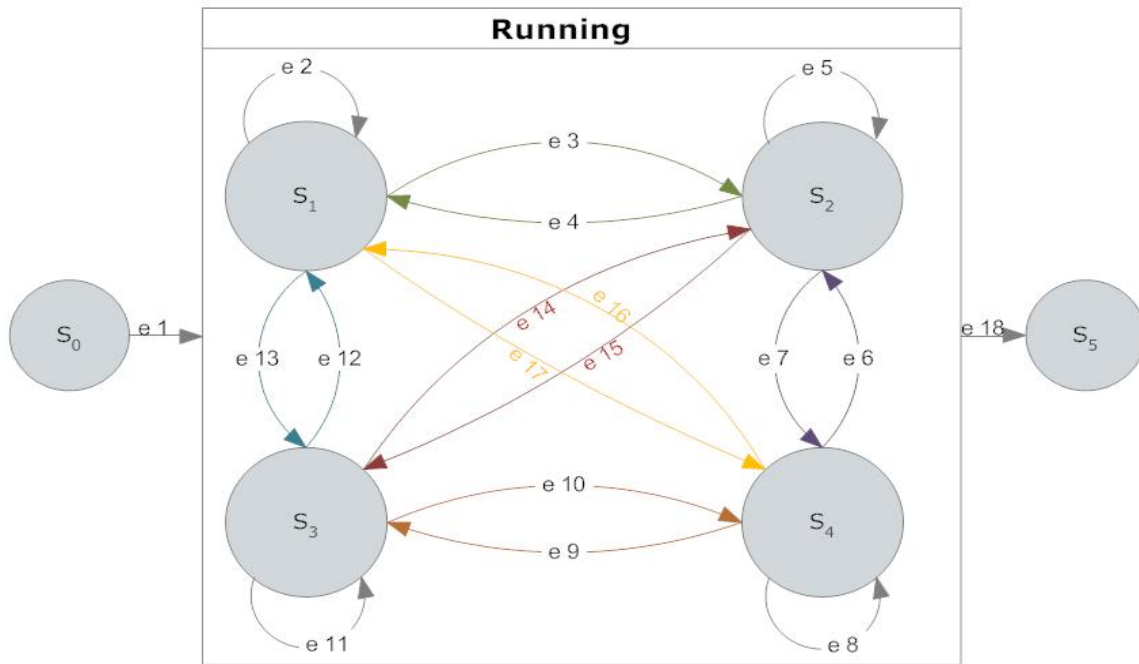


Fig 4-3: the state machine diagram. (Circles indicate states and arrows represent transitions)

The state machine includes four main states that will be monitored through the 18 possible transitions between the states. The finite state machine can be described by a state table, which shows the possibility and direction of each transition to/from a state, these transitions are shown in Table 4-1.

	<i>e 1</i>	<i>e 2</i>	<i>e 3</i>	<i>e 4</i>	<i>e 5</i>	<i>e 6</i>	<i>e 7</i>	<i>e 8</i>	<i>e 9</i>
S0	S1, S2, S3, S4, S5	θ	θ	θ	θ	θ	θ	θ	θ
S1	θ	S1	S2	θ	θ	θ	θ	θ	θ
S2	θ	θ	θ	S1	S2	S4	θ	θ	θ
S3	θ	θ	θ	θ	θ	θ	θ	θ	θ
S4	θ	θ	θ	θ	θ	θ	S2	S4	S3
S5	θ	θ	θ	θ	θ	θ	θ	θ	θ
	<i>e 10</i>	<i>e 11</i>	<i>e 12</i>	<i>e 13</i>	<i>e 14</i>	<i>e 15</i>	<i>e 16</i>	<i>e 17</i>	<i>e 18</i>
S0	θ	θ	θ	θ	θ	θ	θ	θ	S5
S1	θ	θ	θ	S3	θ	θ	θ	S4	S5
S2	θ	θ	θ	θ	θ	S3	θ	θ	S5
S3	S4	S3	S1	θ	S2	θ	θ	θ	S5
S4	θ	θ	θ	θ	θ	θ	S1	θ	S5
S5	θ	θ	θ	θ	θ	θ	θ	θ	θ

Table 4-1: the state transitions table of the finite state machine in Fig 4-1.

Table 4-1 explains the transitions between states, because the state machine is not like a sequential workflow where you can predict which transition will flow from/to which state. In a state machine, it is necessary to make a transition for every possible event that may occur. For example, Table 4-1 shows that the transition (e 14) will transfer the state machine from state S3 to S2, which can also be seen in Fig 4-1. In other words, the system will remain in state S3, which is the state representing the Registry, until the malware tries to add a new process or kill an existing one. In this case, using transition (e 14), the automated state machine will transfer from the Registry state (S3) to the Process state (S2). From this it is possible to identify all malware actions. All 18 transitions cover all the possible actions that can be performed by a malware.

4.2.3 A client Honeypot with a state machine

In simple terms, a state machine is a model that indicates the different states of a task being performed. By planning a strategy in advance such that each state gets defined by its time and resources, the probability of completing a task on time increases and workflows become smoother. A workflow is defined as “a series of steps, decisions, and rules needed to complete a specific task” [83]. There are essentially two types of workflow: Sequential and State Machine Workflow.

We will use the state machine to encode the attacks from malicious sources. The main advantages of the state machine are that the transition from one state to another is not predictable, and that movements are driven by outside events (malicious activities). It also encodes malicious behaviour into a state machine signature format, which we can use.

The previous section described the state machine and its benefits for modelling web-based malware behaviour. In order to use a state machine with a client Honeypot to encode its logged behaviour into a state machine signature, we first have to answer some questions to make it clear how they can be integrated. The first question is: “What are the monitored system states?” To answer this question, the system states during infection by malware, as well as any other benign states, have to be understood. The system state can be changed by adding a registry value, running a new process, or inserting a new cookie. The tools that can be used to monitor each change also have to be looked at. The system states that will be monitored while interacting with a target server are described in section 2.3.3.

The second question is “How does each state transfer to the next state?” Each of the four states listed above will have transitions to another state. Each transition is responsible for transferring from the current to the next state, or even returning to the same state. For example, malware could add a new file, which would transfer the system state to the file system state. If Malware adds another file while it is in the file system state then the state machine remains in the same state, but logs the new information such as action, source and destination of the new change to build the state machine signature.

The third question is: “What information will be saved for each state?” In each state, there are several pieces of information that can be saved to show what the web-based malware changes and activities on the victim system are. The data saved with each state are as follows:

1. **Type:** This field saves the action, which can be different for each state. Table 4-2 shows each state with its types.
2. **Action:** This field saves the action parameters, such as the target file created or the registry value inserted.
3. **Source:** This field describes the source causing the changes.
4. **Destination:** This field saves the file, process or value that has been changed.
5. **Time:** this field holds the time for each change to make up the state machine map ordered by time.

State	Type
File system	Add, modify or delete
Registry	Add, modify or delete
Processes	New or kill
ports	Open or close

Table 4-2: the state types.

4.2.4 Monitoring and detection tools

Open source and free security tools have been developed over the last few years for the Windows environment and have led to the improvement of Windows security. Consequently, there are many security tools now available for Windows, which will be the host operating system for client Honeypot. Table 4-3 shows each monitored activity along with the relevant tools. Involved the Capture-HPC client Honeypot [36], for instance can be used to monitor most of the system parts of the system as presented in section 2.3.3. This could be used as a

basis and combined with other tools to capture activity that Capture-HPC does not support, such as monitoring of ports.

activity	File system	Registry	Process	Ports
tools	FileMon [94]	RegMon [95]	Process Monitor [96]	
	FileSystemWa tcher [97]			
	Capture-HPC [36]			Winsock Control [98]

Table 4-3: each monitored activity along with the relevant tools.

4.2.5 An example of drive-by download attack

It is necessary to understand how this attack works, because it is the main type that a client Honeypot will face when interacting with servers.

1. **The vulnerability:** is in Internet Explorer, which is the default internet browser on the Windows operating system. The vulnerability is a DHTML Object Memory Corruption, caused by race conditions on the memory management routines in DHTML objects, in IE version 5.01, 5.5 and 6. The result of this vulnerability is that an attacker is able to execute malicious code remotely, via his malicious web page, as well as sending a hazardous email with HTML capability. The vulnerability is called CVE-2005-0553 labelled as CVE ID [105] and MS05-020 [110] in the Microsoft security bulletin. According to Microsoft [110], the vulnerability is a remote code execution, “If a user is logged on with administrative user rights, an attacker who successfully exploits this vulnerability could take complete control of an affected system. An attacker could then install programs; view, change, or delete data; or create new accounts with full user rights. Users whose accounts are configured to have fewer user rights on the system could be less impacted than users who operate with administrative user rights.”
2. **The experiment:** The vulnerability was tested in a virtual environment, to reduce the risk of a successful and damaging exploitation. The test operating system was Windows XP Service Pack 2, running Internet Explorer 6 with default settings. The experiment was monitored manually with different security tools available. The tools used were:

- Active Ports 1.4 [111]: this tool displays the open ports on the machine and their operating software in a graphical user interface (GUI).
 - Process Monitor v2.03 [96]: this tool is a real-time monitor, which can show each process and its operation. The processes can be filtered to check Internet Explorer (IE) activities. The tool also shows all the changes of IE from the registry or file system.
 - Process Explorer v11.33 [112]: this tool is an advanced graphical interface used to monitor real-time processes. It helps to monitor the state of the processes exploiting the vulnerability to see if there are any new processes or even run some new ones within IE.
3. **The exploit:** The scenario is centred on the user making a single visit to a malicious web page with their vulnerable Internet Explorer (5.01, 5.5 and 6). The user will not notice anything unusual; instead, they will just see a blank page. The malicious code will exploit the vulnerability in the background; a shell code is responsible for opening a backdoor for the attacker. The exploit is publicly available at [113]. The shell code can do much more than just open a backdoor; it can also install malicious code on the victim's machine, such as a virus or rootkit.
4. **The result:** By monitoring the state of the processes while opening the malicious web page, it is possible to identify a process that runs for just a few seconds before killing itself. That process has the name rundll32.exe, which is a part of the Windows operating system and helps to execute some command line functions. Fig 4-4 shows the process monitor in action.

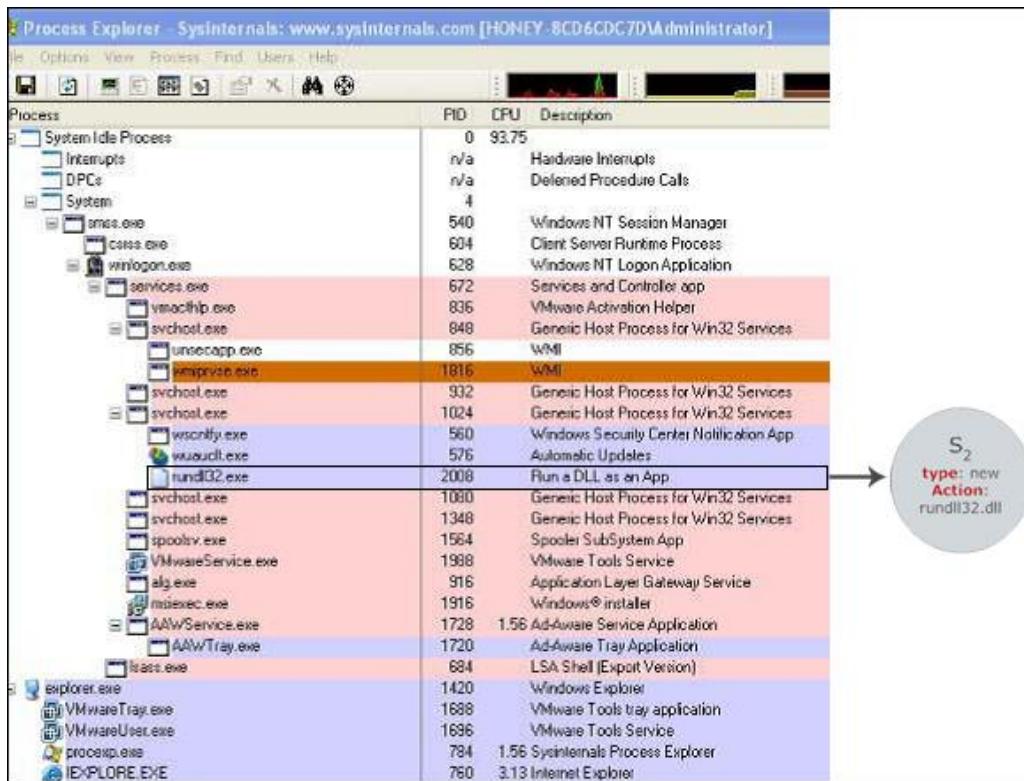


Fig 4-4: the processes monitor in action.

After a successful exploitation from the attacker’s point of view, the users will notice no difference on their computer or Internet Explorer. However, port 28876 will be open for the attacker to connect remotely to the user’s machine. The use of the Active Port scanner shows that the port was opened by `iexplore.exe` which is the process name for Internet Explorer. Fig 4-5 shows the port in action after the exploitation of the vulnerability on the user’s machine.

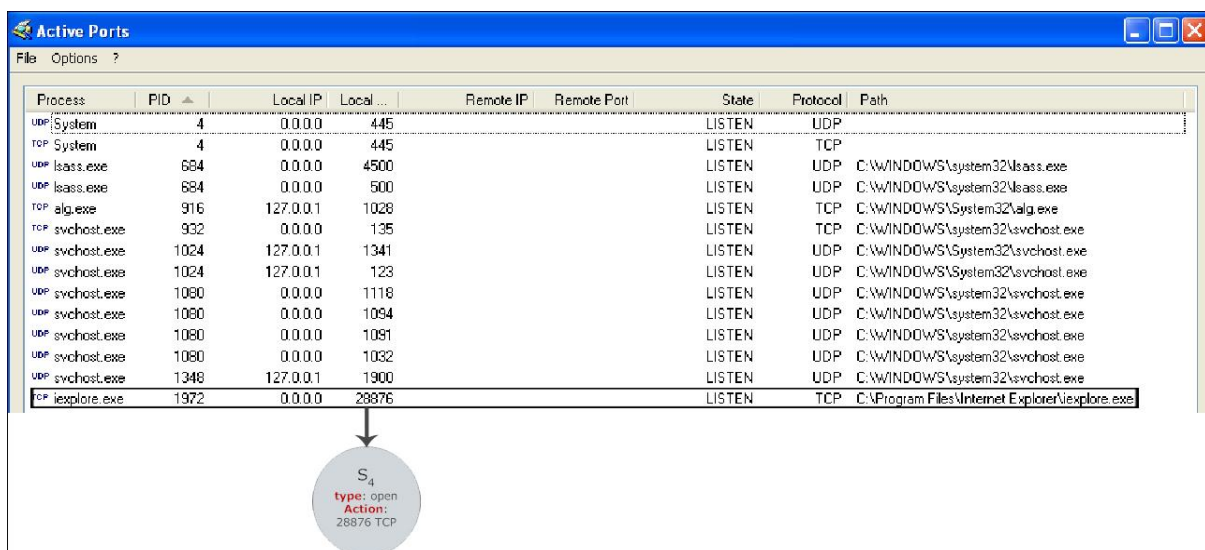


Fig 4-5: Port activity after the exploitation of the vulnerability on the user’s machine.

The result of this attack scenario can also be shown in the state machine diagram, to show clearly how the automated state machine could be applied in the high-interaction client Honeypot. To summarise the previous attack, in order to understand its stages so that we can use it in the state machine, the steps are shown below in time order. Fig 4-6 shows the automated state machine of the CVE-2005-0553 exploit.

- The exploit is transferred with the source page to the user’s vulnerable browser.
- The attack is initiated by starting a new process called rundll23.exe, which was not run from the default processes.
- The TCP port 28876 is opened from the new process running process.

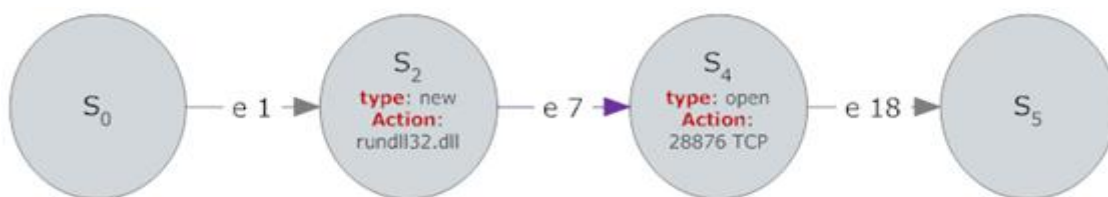


Fig 4-6: the automated state machine of the CVE-2005-0553 exploit.

The diagram shows the automated state machine applied to the CVE-2005-0553 exploit. There are four states that summarise the exploit and the machine’s states. The states’ steps are illustrated below:

1. **S₀**: The first state is the initial state, starts with (e 1) transition, which is responsible for transferring the machine state from the first state (default state) to the S₂ state by adding a new process.
2. **S₂**: The “Processes” state, according to the automated state machine in Fig 4-3, shows the new action, which is to run the ‘rundll32.dll’ process. Transition (e 7) transfers the machine to the S₄ state.
3. **S₄**: The “Ports” state, as described in Fig 4-3. This state is the result of the transition (e 7) and indicates that there is a new port open: TCP port 28876. Transition (e 18) caused a change to the S₅ state.
4. **S₅**: This is the final state, which ends the exploit activity and its effects on the system.

As mentioned above, using the automated state machine allows us to save the whole sequence of changes for an exploit and malicious software within the client Honeypot. The automated state machine saves each exploit with its state machine sequence for further

analysis. One such effect is that most malicious activity is started by adding a new process and editing a particular file and finishes by opening a particular port.

4.2.6 Implementation and Evaluation

The evaluation section focuses on testing the state machine concept, as applied to a real client Honeypot, to see the benefits of this idea in a real situation and to determine how it can be improved in future. The evaluation section is divided into three sub-sections, which are:

The client Honeypot: describes the client Honeypot that will be used with the state machine and how it can be integrated.

The state machine: describes the state machine specification and its implementation.

The evaluation experiment and result: shows the results of the state machine as applied to the client Honeypot.

The state machine will build on Capture-HPC log files. There is a need to understand the structure of these files so that the state machine may be built. Fig 4-7 shows a snapshot of a Capture-HPC log file.

```
file", "Nov 11, 2007 2:50:50 PM", "C:\Program Files\Internet  
Explorer\IEXPLORE.EXE", "Write", "C:\WINDOWS\TEMPmbroit.exe"  
"process", "Nov 11, 2007 2:50:50 PM", "C:\Program Files\Internet  
Explorer\IEXPLORE.EXE", "terminated", "C:\WINDOWS\TEMPmbroit.exe"  
"registry", "Nov 11, 2007 2:50:50  
PM", "C:\WINDOWS\TEMPmbroit.exe", "SetValueKey", "HKLM\SYSTEM\ControlSet001\Control  
\Session Manager\PendingFileRenameOperations"
```

Fig 4-7: snapshot of Capture-HPC log file.

The snapshot shows that three changes have been detected: write a file, terminate a process and insert a new registry value. For each change there are five elements which have to be detected:

- **The state:** the location of the change: File system, Process or Registry.
- **The time:** the time of the change.
- **The source:** the source of the change.
- **The type:** the type of change, which differs from one state to another. File change types are: Write, Delete or Create. Process changes are: Created or Terminated. Registry changes are: Set or Delete value.

- **The destination:** shows the destination of the effect caused by the source file, which can be a file, process or registry value.

The state machine is built using Windows Workflow Foundation [114]. The architecture shown in Fig. 4-8 represents the states and transitions as presented in Fig. 4-3 and Table 4-1. Each box represents a state, such as S0 as the start, S1 as the file system, S2 as the process, S3 as the registry, and S4 as the complete state. Within each box, there is a driving activity that transfers from one state to another as we have described in detail in Table 4-1.

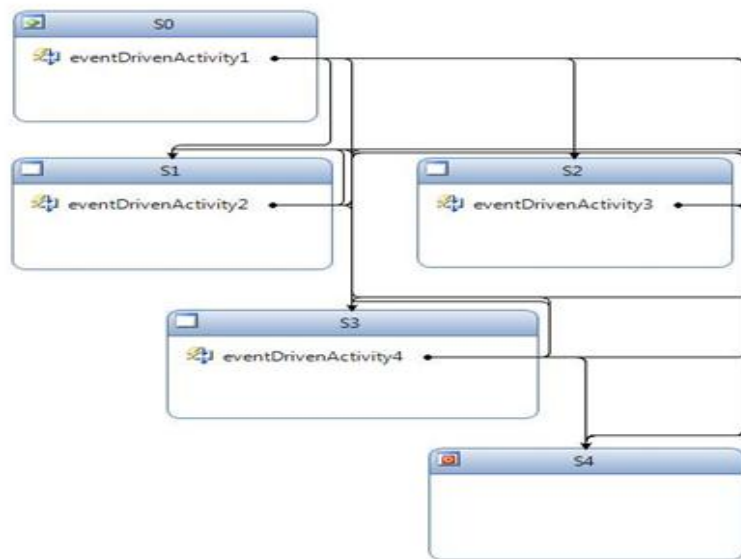


Fig 4-8: the state machine states in Windows Workflow Foundation.

After building the state machine and choosing Capture-HPC to be our base monitoring tool, the tool must now be tested and used to gather results. Each of the 26 different log files received shows a scan result for a different malicious web page. The state machine tool can be configured from an external configuration file to convert each Capture-HPC log file to a state machine version; the configuration file can also set the folder name for the conversion process. Fig 4-9 shows the state machine result files.



Fig 4-9: the file produced by the state machine tool.

The state machine file is output.txt, which contains all the state changes. At the end of the file are two state machines; the first shows the states of the log file with repeated values, while the second shows the states without repetition, which makes it easier to determine malware changes to the system. The machine signature without repeated values is the state_det_short.txt file in Fig 4-8. Most malware will try to make changes more than once to ensure it achieves its aims. The state machine signature also includes the change type, as well as the source and destination. A state machine signature example is shown in Fig 4-10 below:

```
S0--->S1W1=2--->S1W1=3--->S2N1=3--->S3S3=4--->S2K1=3--->S4
```

Fig 4-10: example of a state machine signature.

The state machine above shows the state and type of change in each stage, the states listed above are the same states described in Fig 4-4, while the characters after each state name are described in Table 4-4 below:

State	Character / Describe	
File system (S1)	W	Write
	D	Delete
	C	Create
Process (S2)	N	New
	K	Kill
Registry (S3)	S	Set
	D	Delete

Table 4-4: the state's characters and descriptions.

The evaluation experiment was to convert all the client Honeypot log files into state machine files and then to use an analysis engine to process the log files. All 26 log files were converted to the state machine version as shown in Fig 4-9. The Weka data mining software was then used [52], to get an overview of the 26 state machine files in a graphical format and to understand some properties of the data to help the analyser. The Weka software supports formats such as ARFF (Attribute-Relation File Format) files. The state machine tool produced a Weka state machine signature that was fed to a pre-defined ARFF template. Fig 4-11 shows the graphical output of all 26 state machine files, with some modifications to make the results clearer. It demonstrates that there are six stages for the 26 state machine signatures; the first three stages can be described as follows:

1. First stage: the malicious web pages start writing files to the file system.
2. Second stage: twenty-three web-based attacks involve writing another file to the file system where another three are running a new process.
3. Third stage: two malicious web pages write a new file, twenty-three run a new process and one web-based attack adds a new value to the registry.

The analysis shown below is useful in understanding how a group of Web-based attacks is performed and what the common types of behaviour for such groups are. The analysis can be used as a way to group similar Web-based attacks that affect the end-user system similarly by using clustering algorithms. Further, it can be used to determine the malicious Web pages

attacking a particular network, and thus develop corresponding security protection to safeguard against the highest system parts attacked by such exploits. For example, by using the above analysis, network security can determine that most of the attacks in the first and second steps write new files to the file system. Therefore, there is a need to install file system protections tools and to trace the path of the Web browser vulnerabilities that cause this type of exploit.

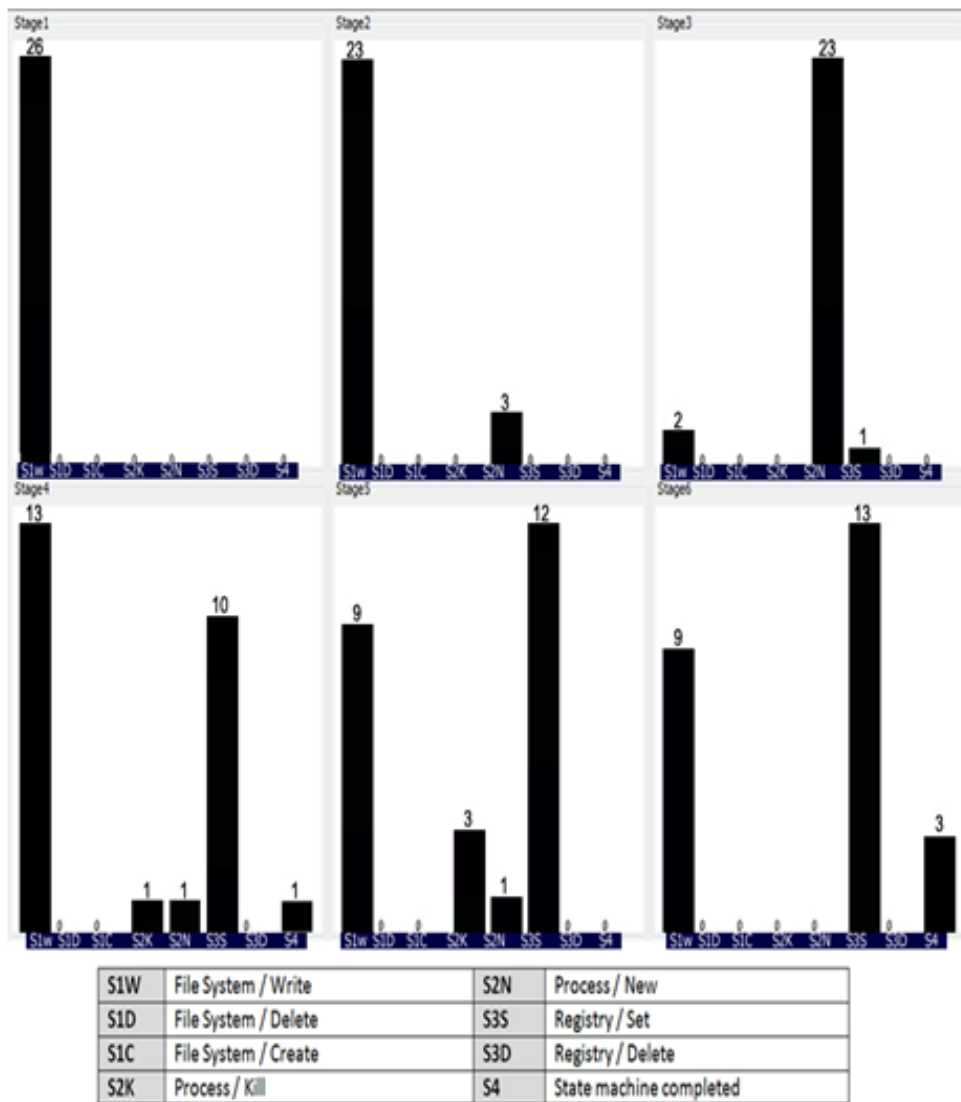


Fig 4-11: Weka graphical output of the 26 malicious web pages.

4.3 Summary and Discussion

In this chapter we discovered how web-based attacks work to exploit the victim's machine. The attack scenario requires the user to visit a malicious web page, which in turn exploits the vulnerabilities of the web browser, its plugins and the user's operating system. We discussed how we could monitor web-based attacks from the client's side, as the attack scenario changes the behaviour for some aspects of the system, such as the file system or processes.

We also introduced the state machine in order to model the behaviour of web-based attacks. We described the different approaches for modeling web-based attacks, such as attack trees, Petri nets and UML. As we have previously seen, a malicious web page attacks the end-user's system through a web browser in stages. The steps in the malicious behaviour should be ordered over time, in order to understand how it works and which state is responsible for the transition to the next activity. Therefore, we chose an attack graph to enable us to visualise the steps in each web-based attack scenario. We based our model on a finite state machine, as this provides a simple way to represent and save the behaviour data in the states, starting with the first state, where the attack begins, and from that point tracing the event-driven process; any change should trigger an event within the current state and cause it to transfer to the next state, which represents another malicious activity. The model was based on the FSM because the parts of the system that needed to be monitored were limited. The state machine signature allows security researchers and automatic tools to understand and follow the steps of the web-based attacks.

This chapter introduced the first part of our behaviour model, which shows the state machine modeling the behaviour of web-based attacks. In the next chapter, we explore the clustering model and how it can group similar malicious web page behaviours together into one main behaviour group, as well as creating sub-groups for those malicious web pages that have a greater similarity between them.

Chapter 5

Clustering model

This chapter discusses how we group web-based malware based on their behaviour by using clustering algorithms. Client Honeypots visit and interact with suspect websites in order to detect and collect information about malware. Interactions with malicious web sites may cause changes to the properties of a victim's system, often without the system owner being aware that this is occurring. The activities performed by web-based malware can be used to group them according to their effects on a victim's system. We use a state machine based representation to encode log data from a client's system, as discussed previously in Chapter 4, subsequently grouping this data based on the "types" of activities that were performed on the machine (referred to as malware behaviour).

The structure of this chapter is as follows: Section 5.1 describes the idea of using the clustering algorithm with the state machine signature; Section 5.2 discusses the use of the FP-Growth to group the behaviours of web-based malware.

5.1 Clustering model overview

As previously outlined, our state machine encodes the sequence of events that occur during an interaction between a client and a web server. The state sequence derived from multiple interactions can be clustered to identify common patterns between them. Table 5-1 shows examples of some state machine signatures, the state's characters and descriptions are presented previously in table 4-2 in chapter 4.

S1W1=13,S1W1=14,S2N1=14,S1W1=15,S3S14=16,S1W14=17
S1W1=18,S1W1=14,S2N1=14,S1W1=15,S1W14=17,S3S14=16
S1W1=2,S1W1=3,S2N1=3,S2K1=3,S3S3=4
S1W1=13,S1W1=14,S2N1=14,S1W1=15,S3S14=16,S1W14=17

Table 5-1: example of state machine signatures.

One way to solve this is to group the state machine signatures using data mining techniques, particularly clustering algorithms. The term clustering covers a number of different techniques used to group similar items together and can be defined as "the process of organizing objects into groups whose members are similar in some way" [99]. Clustering can help us with our state machine files by:

- 1) Grouping state machine files that are the same, which can be used to group different URLs that use the same web-based malware.
- 2) Grouping similar state machine signatures to build behaviour families of web-based malware, containing malware that use similar techniques or attacks, which could lead to a better analysis of the relationships between the state machine activities.
- 3) Reducing the analysis time, thus providing the researcher with groups of web-based malware to analyse; previously, each log file would have to be analysed separately to understand the behaviour of the web page scanned.

5.1.1 The motivation of web based attacks clustering

In the first three months of 2011, there was an average of 73,000 new malware strains found daily according by PandaLabs [104], an increase of 26% from the same period in 2010. Meanwhile there is still a large amount of old malware affecting thousands of computers worldwide. These numbers raise the challenge of how to analyse these attacks automatically, particularly as the traditional method involves a human analysing the malware, which clearly is time consuming and requires a lot of effort. Client Honeypots visit and interact with suspect web sites in order to detect and collect information about malware. Malicious websites may cause a number of activities to be performed on a victim's system; each activity is performed in different stages. We use a state machine, as discussed in the previous chapter, to represent the activities performed by the malicious web page into pre-defined states. These states can be used to summarise interactions with malicious web pages using the same state machine structure. The states are then passed to a clustering algorithm to group similar

malicious web page exploits in order to better understand how software can be developed to better respond to such attacks. The outputs of the clustering algorithm are categorised to build up groups of similar states that represent the malicious activities performed on the victim's system. The benefit of using this process is to build families of malicious web pages with similar behaviours (behaviour families) leading to the development of common approaches to deal with such exploits.

5.1.2 Clustering methodology types

As discussed in Chapter 2, there are many different approaches used to cluster and group attacks that exploit browser and operating system vulnerabilities. A summary of the previously mentioned approaches is given next. HAC has been widely used to cluster similar attacks. Rieck et al. [47] use HAC with complete linkage as the distance measurement for malware clustering based on MIST format; MIST is a string that represents malware behaviour in a hash file format. While Gurutxage et al. [115] use the same clustering approach, but with average linkage as the distance measurement instead of complete linkage. Moreover, Sequeira and Zaki [57] use still another clustering approach, based on K-means, which clusters the user behaviour in order to use it with an anomaly-based detection technique for detecting abnormal behaviour in real time in a UNIX shell. Christian et al. [51] use a hybrid approach, which combines K-means and Self-Organising Maps to cluster and visualise malware behaviour. Finally, there are more approaches that can be used for malware clustering based on behaviour, such as the artificial neural network used in the work conducted by Linda et al. [56] and Stopel et al. [55].

Our clustering approach is based on FP-Growth clustering, which differs from all of the existing clustering algorithms for dynamic behaviour analysis. Most of the previous clustering approaches have used hierarchical clustering. We have used HAC with a state machine, and found that it suffers from a number of limitations. One of them is that malware might make some changes to some of its behaviour in order to deceive the analysis engine (clustering algorithms) into grouping similar malware into different behaviour groups. By using the FP-Growth algorithm, we can detect more malicious web pages and group them based on their common behaviour states by comparing them to HAC; this then allows similar malware behaviour to be detected, even in different parts of the state machine's signature.

Furthermore, most of the clustering approaches identified previously present acceptable performance when grouping similar attack sequences that have one or several malware behaviours. However, our clustering model is able to identify several aspects of the state machine, or the behaviour sequence, of web-based malware. While developing the clustering model for malicious web pages, we already knew that this type of attack is different from traditional attacks, as they first need to exploit a vulnerability in the web browser or operating system to gain permission to access the system and open the door for the attacker to install and run different malware. Therefore, the state machine representation helps by detecting the sequence of changes in real-time and in time order, and can therefore distinguish between the different types of malicious behavior, such as exploiting a web browser vulnerability, installing and running one or more malware codes on the system, or monitoring the traces of malware behaviour. Furthermore, the clustering model uses these benefits of the state machine representation by grouping them according to common states and can therefore detect the practical web browser exploits, Trojan horses, viruses or new malicious techniques being used by a 0-day exploit.

5.1.3 Clustering model based on FSM

The previous chapter discussed our web-based behaviour model, the finite state machine, and how it represents the behaviour of web-based attacks in a complete and easy way. The clustering model used our finite state machine model to understand web-based attacks and then group similar attacks based on the behaviours. The state machine model represents web-based attacks as a sequence of changes that make it easy for data mining algorithms, such as FP-Growth, to read the states and group similar states. In the case of our clustering model, which is based on the FP-Growth algorithm, it groups similar web attacks based on the common states from each state machine sequence. Therefore, the use of the state machine model assists the clustering model in processing and clustering the behaviour of malicious web pages quicker and easier, while also making it readable for a human. Furthermore, the clustering model creates sub-groups that show similar states within each main behaviour group. The next sections discuss the use of the FP-Growth clustering algorithm and how they create the behaviour groups based on the state machine model that represents malicious web page behaviour.

5.2 Clustering model

To summarise the benefits of the clustering model, HAC provides a good clustering algorithm that can group the state machine signatures into behaviour groups; However, some limitations exist within the HAC clustering approach, such as: (a) each state machine has to be of the same length, requiring insertion of null states, (b) the web-based behaviour groups produced by the clustering algorithm do not provide a label for each group. The label for each group shows how the state machine signatures are grouped (the common state sequences are repeated within each group's files). For example, Table 5-2 below shows three state machine signatures and the group's label, which is the underlining statement: S1W1=2, S1W1=3, S2N1=2.

<u>S1W1=2,S1W1=3,S2N1=2</u> ,S362=4,S2N1=2
<u>S1W1=2,S1W1=3,S2N1=2</u> ,S2N1=3,S2N1=2,S3R2=4
<u>S1W1=2,S1W1=3,S2N1=2</u> ,S3R2=4,SSN2=5

Table 5-2 shows three state machine signatures and their label underlined.

Therefore, each group has an identifier that represents a behaviour group, and can subsequently be compared with other groups; (c) to add additional behaviour groups, it is necessary to re-cluster the entire data set, which is a time consuming process.

5.2.1 FP-growth clustering model

The frequent pattern growth (FP-growth) based clustering approach uses an FP-tree (extended tree) to store the database and groups data according to the frequency of occurrence of a specific behaviour. By using the FP-growth algorithm we can detect more similar malicious web pages and group them based on their common behaviour states. Through this we can detect similar malware behaviour, even in different parts of the state machine signature. FP-growth has the ability to mark each behaviour group with a label and can deal with arbitrary length state sequences. The label represents frequent states within each group. By using this label, we can identify similarities between groups and then use the label to match with new data. Each sub-group will have a label to identify the common states within each group which indicates why each sub-group state machine is similar. Using this clustering approach we can add new behaviour groups to the already discovered groups. The benefit of this is that we will

only need to re-cluster and match the new and old group labels, and not have to re-cluster the entire data set. The next section will describe the clustering algorithm in detail, as well as discussing the benefits of using the FP-growth clustering algorithm as the basis for our clustering model.

5.2.2 FP-growth clustering methodology

FP-growth has the ability to mark each behaviour group with a label and can deal with a state sequence of arbitrary length. The label represents frequent states within each group. Using this label we can identify similarities between groups and then use the label to match new data. Each sub-group has a label to identify the common states within each group, which indicates how each sub-group is similar. Using this clustering approach we can add new behaviour groups to the already discovered groups. The benefit of this is that we will only need to re-cluster and match the new and old group labels, and will not have to re-cluster the entire data set in case we have similarities in the labels. The FP-growth based clustering approach is shown in Fig. 5-1.

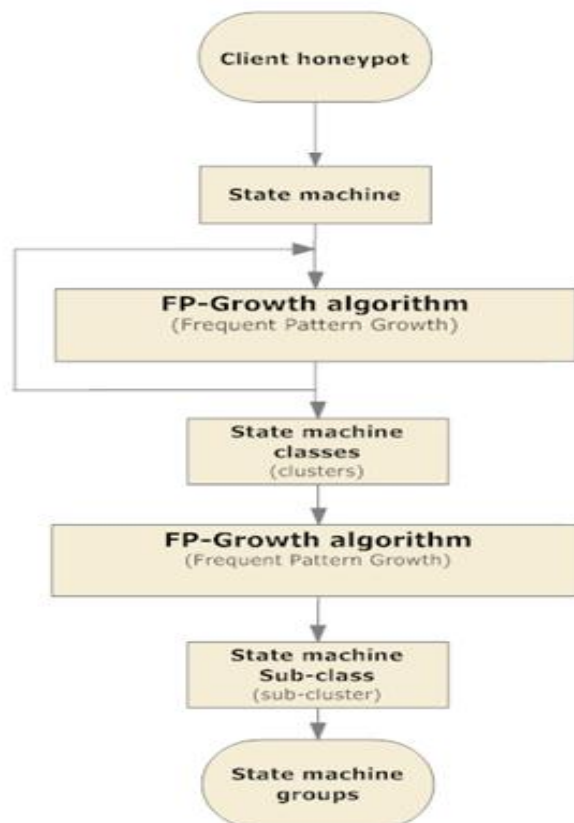


Fig. 5-1: FP-growth clustering

The FP-growth clustering algorithm receives the web-based malware behaviour from the Capture-HPC as log files and encodes them into a state machine representation, as mentioned in Chapter 4. Then the model loads the state machine signatures and runs FP-growth algorithms to find the main behaviour group. The process starts by creating a list that will have all the state machine signatures and then attempts to find the most commonly occurring behaviours across the various state machine representations. The longest common represents frequent states among the state machine signatures which identifies the similarities between them. It then eliminates those state machine signatures that have the longest common state sequences from the list and groups them into a main behaviour group. Furthermore, the algorithm will re-cluster the other state machine signatures from the list until it groups all of them. The second stage of the model is responsible for creating sub-groups in each main behaviour group. This process is the same as the first FP-growth clustering process, but it will create sub-groups within each main behaviour group and then move to the next one. The benefit of creating the sub-groups is to find more similarities within each main behaviour group's files. The next sections describe the FP-growth algorithm and groups in detail.

5.2.3 FP-growth clustering algorithm

The algorithm benefit is to find the similar state machine signatures. We work with a set of state machine signatures $N = \{N_1, N_2, N_3, \dots, N_n\}$, where each N_i identifies a set of states, such as $N_i = \{S_1, S_2, S_3, \dots, S_m\}$. It is important to note that each state $S_i \in N_j, i \leq j$ represents a malicious behaviour in the victim system. The set N can work a number of states where each state represents a behaviour that might either be similar to or differ from other containing states. The clustering algorithm pseudo-code is shown in Listing 5-1.


```

1. Load(Ni)
2. Build(A,Ni)
3. While A != 0
4.   Begin
5.     For i=0 to n
6.       Begin
7.         For j=1 to m
8.           begin
9.             L=max(Ni,Nj)
10.          End for
11.        End For
12.       Group(Np,L)
13.      Eliminate(Np)
14.    End While

```

Listing 5-1: the clustering algorithm pseudo-code.

The algorithm loads all n states N_i and builds an array A that includes all n states N_i to create groups of similar states. Furthermore, the algorithm finds the longest similar states S among the set of states N_i ; $i < n$. A number of state machine signatures with the longest sequence L are identified by creating a behaviour group for all the N_i states with the longest similar states S ; L identifies the longest similar states among N_i . In the last stage the algorithm eliminates L from the data set A by deleting those N_i states with the longest similar states from A . An example of applying the clustering algorithm is giving in the next section by grouping three of Trojan Apher [118] family.

5.2.4 FP-growth clustering groups

The clustering model outputs two behaviour group types: main and sub-groups. The main group contains similar state machine signatures that share the same sequence of states. This is helpful for grouping similar web-based attacks together, such as grouping attacks that download executable files and change specific registry values or attacking the web browser using a specific exploit. However, within each main group some of the web-based behaviour attacks might share other similarities. The sub-group is responsible for further refining each group. This stage can provide more information within each group, such as finding additional similarities in each behaviour group, or showing that these sub-groups can help better

characterise each piece of recorded behaviour. An example of the main and sub-group is a Trojan called “Trojan.Apher” [118]; it downloads files via the Internet without the user’s knowledge or consent. The Trojan’s size may vary in the family, but once executed it downloads other files from the Internet and then deletes itself. Table 5-3 shows the scan results of the three Trojans from VirusTotal [131]. For example, Trojan-Downloader.Win32.Apher.ad has been detected by 39 anti-virus products out of 43.

Virus name	MD5 Hash	No of detections
Trojan-Downloader.Win32.Apher.ad	e7d4ab448872b52b3d52d29c4ff5949d	39/ 43 (90.7%)
Trojan-Downloader.Win32.Apher.am	812a95a8fa42d7f5fb086f7d35bf517b	36/ 43 (83.7%)
Trojan-Downloader.Win32.Apher.ab	ce7e908614c5bf549a2c3094ee35c459	33/ 43 (76.7%)

Table 5-3: Scan result of the Trojan.Apher family.

We scan these three Trojans by using Capture-HPC to monitor and log their behaviours. Then we encode their log files as state machine signatures, as presented in Table 5-4. It shows that all the three Trojans start with the same 18 states, which are responsible for changing the browser proxy settings to download other files from a pre-defined malicious server.

Name	State machine signature
Apher.ad	<u>S2N1=2,S2N2=3,S3S2=4, S3S2=5, S3S2=6, S3S2=7, S3S2=8, S3S2=9, S3S2=10, S3S2=11, S3S2=12, S3S2=13, S3S2=14,S3R2=15, S3R2=16, S3R2=17,S3S2=18,S3S2=19</u>
Apher.am	<u>S2N1=2,S2N2=3,S3S2=4, S3S2=5, S3S2=6, S3S2=7, S3S2=8, S3S2=9, S3S2=10, S3S2=11, S3S2=12, S3S2=13, S3S2=14,S3R2=15, S3R2=16, S3R2=17,S3S2=18,S3S2=19, S3S2=20, S3S2=21, S3S2=22,S4T2=23</u>
Apher.ab	<u>S2N1=2,S2N2=3,S3S2=4, S3S2=5, S3S2=6, S3S2=7, S3S2=8, S3S2=9, S3S2=10, S3S2=11, S3S2=12, S3S2=13, S3S2=14,S3R2=15, S3R2=16, S3R2=17,S3S2=18,S3S2=19, S3S2=20, S3S2=21, S3S2=22, S4T2=23,S1W2=24,S4T2=25, S2N2=24,S2K1=2</u>

Table 5-4: three Trojans state machine signatures.

We then use our clustering model to group the three Trojans based on the methodology described earlier in Fig 5-1. The clustering algorithm finds one main behaviour group that has all three Trojans, because all of them share the first 18 states, which are the longest similar state sequences found. The algorithm also finds one sub-group, which contains the Trojan.Apher.am and Trojan.Apher.ab signatures and their longest similar states are in red in Table 5-3. Fig. 5-3 shows the main and sub-behaviour groups of the Trojan.Apher family.

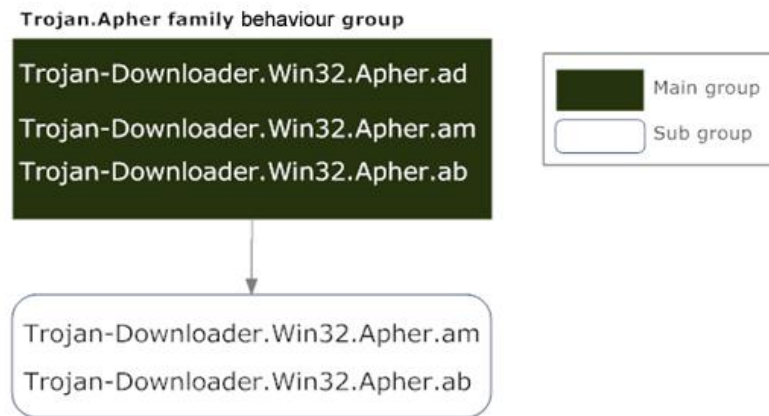


Fig 5-2: main and sub-behaviour group of the Trojan.Apher family.

5.2.5 FP-growth evaluation

To validate our clustering model that is based on the FP-Growth algorithm, we conducted two experiments that both used the same data set of 110 state machine signatures collected from public malware blacklists [116] [117]. The experiment's methodology is to pass the 110 state machine signatures to the FP-Growth and HAC clustering algorithms so that they can be grouped and group behaviour can be outputted. As mentioned in Chapter 2, most of the current malware clustering approaches use HAC, so we also chose HAC clustering approach to validate FP-Growth.

The first step is to group the experiment malicious Web pages by using FP-Growth, as mentioned earlier. FP-Growth can cluster based on the common sequence of states. The experiment uses the same methodology as described previously in Fig. 5-1, in which it starts to bypass all experiment state machine signatures to FP-Growth and then output the main behaviour. Further, in each main behaviour group, the algorithm attempts to find sub-groups. The result of the first experiment that used FP-Growth is shown in Table 5-5. The findings show that FP-Growth found 27 behaviour groups in total, with the largest group having 34 state machine signatures.

Group	Group label	Number of State machines
1	S1W1=5, S1W1=19,S2N1=19	34
2	S1W1=70,S2N1=71	15
3	S1W1=31,S2N1=31,S3S31=32	8
4	S1W1=8,S2N1=8	9

5	S1W1=115,S2N1=115, S3S115=9, S2N115=61, S1W115=116,	5
6	S1W1=51,S2N1=51	4
7	S1W1=3,S2N1=3	4
8	S1W1=64,S2N1=64, S3S64=40,S3S64=41,S3S64=42,S3S64=43	3
9	S1W1=54, S1W56=57, S1W56=58	3
10	S1W1=104,S2N1=104	3
11	S1W1=45,S2N1=45,S3S47=49, S3S47=50	2
12	S2N1=71	5
13-27	1 in each group	1

Table 5-5: the behaviour groups output from FP-Growth clustering.

The second experiment grouped the same data set from the first experiment by using the HAC clustering algorithm. The hierarchical agglomerative clustering (HAC) algorithm can group clusters (each state machine signature is one cluster) into one major cluster by combining them together in different stages. The technique used in the hierarchical agglomerative algorithm to group similar clusters is called ‘distance measures’, which has three different types:

- Single linkage clustering: the minimum distance between items in each cluster
- Complete linkage clustering: the maximum distance between items in each cluster
- Average linkage clustering: the mean distance between items in each cluster

Using the above three distance measures, the hierarchical agglomerative algorithm can calculate the distance between clusters in order to merge similar clusters together and to build a truss of similarity. The purpose is to help identify similar state machine signatures and combine them to build an automated analysis engine for state machine signatures, as well as to build families of similar Web-based malware (behaviour families).

The experiment starts by passing the 110 state machine signatures to HAC. We used the average distance measure as it shows from our tests that it is the best distance measure in grouping the files into the behaviour group. Table 5-6 show the experiment results. HAC found 77 behaviour groups, with the largest group having 8 state machine signatures.

Group	Log files	Group	Log files	Group	Log files
1	3	4	8	7	6
2	3	5	6	8	4
3	7	6	2	9	3
10	2	11-77	1 in each group		

Table 5-6: The experiment behaviour groups by HAC.

We tested both FP-Growth and HAC clustering algorithms and applied them to the experimental data discussed here. The experiment shows that the HAC algorithm identified 77 groups. The FP-Growth algorithm had 27 behaviour groups, with the largest group having 34 state machines. Hence, FP-Growth found additional similarities between state machines. Table 5-7 compares the results of the HAC and FP-Growth clustering algorithms.

	HAC	FP-growth
Number of behaviour groups	77	27
Highest behaviour group items number	8	34
Sub group option	no	Yes
Highest sub group items number	none	11
show the frequent state repeated	no	Yes

Table 5-7: a comparison between FP-Growth and HAC.

5.3 Summary and Discussion

The previous chapter introduced the state machine model, which is a way to encode and model the behaviour of web-based attacks in an ordered sequence of states. In this chapter, we focused on the second part of the web-based malware behaviour, which is the clustering model. This model was introduced to a group of similar web-based attacks based on their behaviour. These behaviour groups showed that similar behaviour has been used in different or identical, web-based attacks. This will help security companies and researchers study the different tricks used by the same exploit, and develop a security patch or detection mechanism

against this type of attack by identifying the common behaviour among malicious web pages in the same behaviour group.

This chapter began with an explanation of the benefits of using the clustering model to group similar behaviour of web-based attacks and discussed many of the advantages of applying it. One of the advantages is that we need a way to group similar web-based attacks into one group, which makes it easy for security researchers to investigate and study this particular type of attack, and this knowledge can be applied to anti-malware software in order to build static or dynamic signature for detecting this type of attack.

In Chapter 2, and at the beginning of this chapter, we mentioned the other approaches used to group attacks in general and we have selected the most familiar approach, HAC, which has been used widely to cluster attacks and exploit security vulnerabilities. However, some limitations exist within the HAC clustering approach, such as: (a) each state machine has to be of the same length, requiring the insertion of null states; (b) the web-based behaviour groups produced by the clustering algorithm do not provide a label for each group; (c) to add additional behaviour groups, it is necessary to re-cluster the entire data set, which is a time-consuming process. Therefore, we use the FP-Growth clustering approach to overcome the previous limitations. The use of FP-Growth has been discussed in Section 5.2.2 and showed that we were able to acquire the behaviour groups, as well as sub-groups, from the state machine signatures.

In the next chapter we are going to discuss how we can extend the use of clustering and state machine models by using a prediction model. It works with the clustering and state machine models to build a complete web-based behaviour system. The prediction model enables us to discover the potential future behaviour for a given state machine signature, and using the classification and clustering models, we will be able to decide whether to allow an interaction or to block it and add it to the malware database

Chapter 6

Prediction model

The last two chapters discussed the state machine and clustering models. The prediction model aims to determine the future behavior for a given state machine signature, based on the state machine modeling of previously scanned web-based malware.

A summary of malware detection approaches as presented in Chapter 2 is given next in order to present what are the other approaches of detecting malware compared with our model. The dynamic malware decoction types are anomaly, specification and signature. The dynamic anomaly detection starts in the training phase in order to learn the behaviour of the program and to build normal behaviour. It can then monitor the program during its execution and compare it to normal behaviour; by doing this it can detect abnormal behaviour, which might be malicious behaviour or activities. Hofmeyr [68] used an approach that detects abnormal behaviour in privileged processes based on the system calls. Wang and Stolfo [69] created PAYL (the payload-based anomaly detector), which involves building normal behaviour by scanning the application payload of the network traffic. They used the Mahalanobis distance [69] to measure the similarity between the incoming data (unknown) from the application during detection against the pre-defined profiles and then flagging an alert if it exceeds the threshold.

The dynamic specification detects malicious behaviour by determining the similarity during execution against a pre-defined set. It differs from the previous type as it has a pre-defined set of rules where it detects the malicious behaviour if it matches these rules. Masri and Podgurki

[71] developed Dynamic Information Flow Analysis (DIFA) to detect intrusions and attacks against Java applications by specifying flow signatures for known attacks and blocking the intrusions when they matched. Krida et al. [74] presented Noxes, which is a Windows-based service that acts as a web proxy and receives all the HTTP connections and then passes or blocks the connections based on the policy of the tool. The policy is a specification of pre-defined XSS attack behaviours. Finally, dynamic signatures rely on using real signatures of malicious applications in order to detect them. Signature-based detection is one of the oldest methods used to detect malware, and is used by the majority of security products. An example of this type was presented by Ilgun et al. [75] and is called the State Transition Analysis Tool (STAT), which is able to detect malicious behaviour by comparing the monitoring behaviour to a pre-defined STAT (known attacks in the format of a STAT). Radosavac et al. [21] proposed modelling the buffer overflow attacks in an FSM and then using HMM to detect such attacks. Their approach is only able to detect buffer overflows that have similar behaviours between attacks. On the other hand, Lee et al. [77] proposed an intrusion detection system based on multi-stage determination. Their method states that, after each stage of the intrusion signal, the HMM algorithm attempts to identify the current malicious sequence out of the pre-defined ones in its database.

Xiuqing et al. [78] proposed the use of the Fast Adaptive Clustering Algorithm (FACA) to update the HMM dynamically with new attacks in the case of misuse detection and then update the misuse detection engine with normal and abnormal signals. After clustering using FACA, the Davies-Bouldin (DB) index is used to measure the distance between the clusters and then merge them if they are small and similar, which represents normal behaviour, or detect abnormal behaviour if the distance is larger. The system uses the clustering output to build the HMM and uses it with the anomaly detection system to detect abnormal behaviour. Tsai and Hung [19] presented an intrusion detection system based on the multi-dimensional HMM, and had three models: the monitor, track and analysis models. They used a Honeynet deployment for their system instead of the multi-agent sensors used in the previous approach. They stated that the Honeynet adds the benefits of tracking and recording the exact behaviour and accessing trails of intrusions. They then used the ant colony algorithm with a deposit of different pheromones to analyse the intrusion traversal. In their deployment they collected all the data from security monitoring tools, such as Firewall, antivirus and SQL queries, and then passed them to an analysis model. They used the multi-dimensional HMM and used cross-

correlation and autocorrelation measurement to integrate and advance the analysis. Finally, as in the previous approach, they used a fuzzy inference rule to identify and recognise intrusions.

In our work we are using the HMM to predict the next activities of a given malicious behaviour sequence. Lee et al. [77] proposed using multistage detection and running HMM in every stage of attack to determine similar attack sequences. In our model of HMM, we employ an algorithm to predict the future behaviour of a given state machine sequence based on the previously known information from the state machine sequences in a database. Therefore, it is able to predict future activities of a given web-based malware attack sequence in each stage of the attack. Moreover, when the malicious web page is new (0-day exploit) then we can predict potential future behaviour based on similar state machine sequences in the database.

Furthermore, our HMM prediction model uses an FP-growth clustering algorithm to help detect web-based malware. The HMM will pass the unknown state machine signature to the clustering model to find more relationships between the given web-based state machine and the others in the database. It can therefore group it into a similar main or sub-behaviour group, or create a new behaviour group if it is a new attack sequence; this will improve detection in the future.

6.1 Prediction model overview

As we have discussed previously in section 2.3, there are different methods of using our prediction algorithm, HMM, in order to detect the attacks. Some of the approaches used HMM as a matching method between the attack behaviour against the HMM models of previous attacks in the database, whereas other approaches used HMM as a way of clustering approach to cluster similar attacks behaviour. In our work, we used it as a prediction and detection algorithm to detect web-based malware.

Before describing our model in detail, it is important to summarise our web-based behaviour system which used different models to process malware behaviour before passing it to the prediction algorithm. Firstly, the state machine model, as presented in chapter 4, monitors the client Honeypot state while interacting with target malicious web pages and then generates a state machine signature which shows all the web-based malware behavior ordered by time.

The state machine signature has rich data about the relationships of the behaviour states. Furthermore, it then passed the state machine signatures FP-growth clustering which clusters them based on the common behaviour between the state machine signatures. Therefore, it builds the web-based behaviour groups and includes the similar state machine into one behavior group. Finally, the web-based behavior model monitors the state of the client Honey pots and generates state machine signature in real-time, and sends it to the prediction model in real-time in order to predict possible malicious future behaviour.

The prediction model works by attempting to predict the next missing value of the given state machine based on entries in the database. In every stage of prediction, it will search for similar states between current malicious web page behaviour against the pre-scanned malicious behaviour and then calculate the highest possibilities for the next activity.

6.1.1 Prediction model methodology

The prediction algorithm will scan the current web page and then try to predict future behaviour to determine if it is malicious or benign. The prediction algorithm uses its knowledge based on previously-seen web-based malware behaviour to understand how they operate and which types of malicious web pages are related to which behaviour group. The prediction model is able to operate in two different modes:

- A. Predict the next state: in this method, the prediction algorithm will try to predict the next state based on the most frequently seen next state. This will calculate all related state machine signatures and then select the highest probability for the next state.
- B. Predict the behaviour signature: the algorithm will predict the whole state machine – using successive use of approach (A) above.

We have built a model that can be used with a client Honey pot to predict web-based malware behaviour. Fig. 6-1 provides an overview of the prediction model.

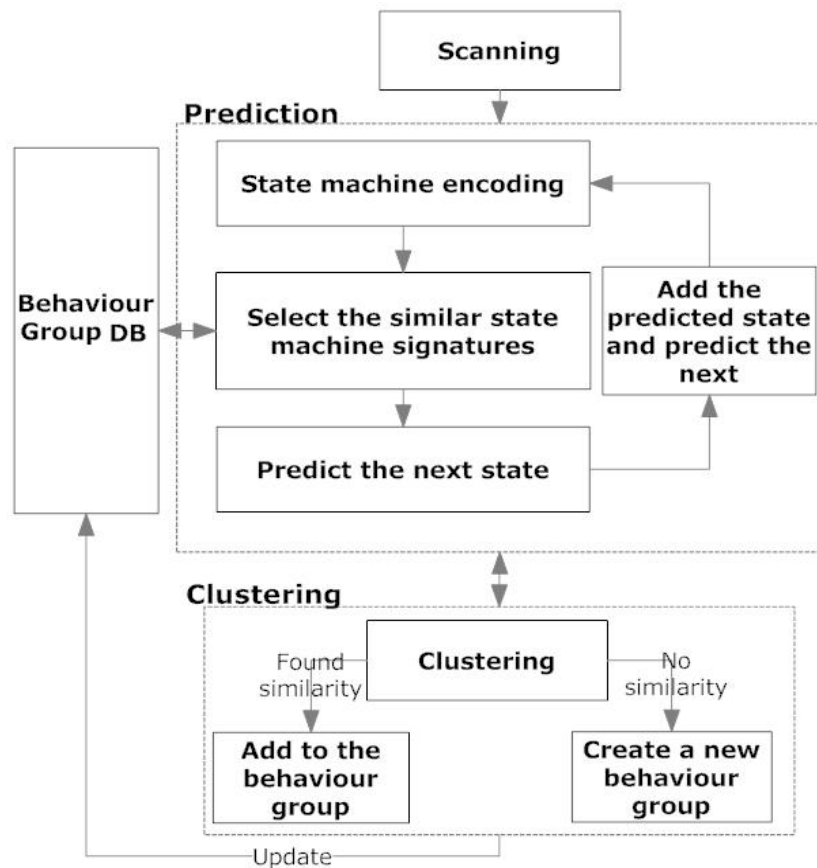


Fig 6-1: the prediction model overview

The prediction model starts by encoding web-based behaviour into a state machine as described in chapter 4. The malware database contains a number of main and sub behaviour groups which have been scanned and grouped together previously. Each behaviour group contains state machines with similarities.

The next stage is to select similar state machine signatures from our malware database according to the current interaction. Listing 6-2 shows how we select the similar state machine signatures from the malware database. Similar state machines are selected in accordance with the initial states of the input state machine. Therefore, the prediction algorithm uses the selected state machines to identify and predict the next state for the input state machine. The highest probability for the next state is calculated and selected as the predicted state. In Listing 6-1 we explain how the similar state machines are selected. In the first step, similar state machines with associated change operations are identified (source and destination of the change), while the second step finds similar state machines excluding change operations (only using the state of change i.e. file system or registry).

$$A = \{A_1, A_2, A_3, \dots, A_n\}, A_i \in A, A_i = \{S_1, S_2, S_3, \dots, S_m\}, S_j \in A_i, S_j \rightarrow S_i O_k = x,$$

1. First step: $\forall A_i \in A, A_i = \{S_1, S_2, S_3, \dots, S_m\}, \exists A_{sim} = A_i \cap A_j, A_{sim} = \{S_1, S_2, S_3, \dots, S_p\}, P \leq m, S_k \rightarrow S_i O_j = x,$
2. Second step: $\forall A_i \in A, A_i = \{S_1, S_2, S_3, \dots, S_m\}, \exists R_{A_i} = A_i - A_i/j = x, R_{A_i} = \{S_{s_1}, S_{s_2}, S_{s_3}, \dots, S_{s_p}\}, S_{s_i} \in R_{A_i}, S_{s_i} = S_{i_o}$

Listing 6-1: how we select the similar state machine signatures from the malware database

An example of selecting the most likely next state is shown in Fig. 6-4 using the example of a web-based state machine for which the signature is “S1W1=2, S2N1=2”. The algorithm needs to predict the third state or activity the web-based malware might perform. In order to predict the next activity, similar state machine signatures are selected from the malware database, and then the highest probability for the next state of the similar state machine is calculated. Therefore, the predicted next state in our example is “S3S2=3” because both similar state machines have this as the next state, as shown in Fig. 6-2 below.

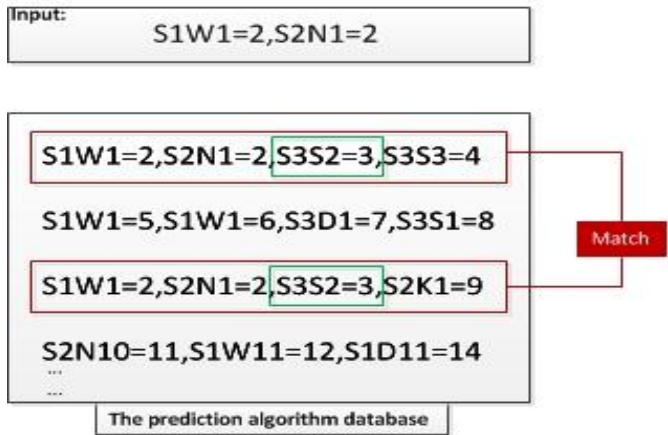


Fig 6-2: an overview idea of the prediction algorithm.

After selecting the next state machine, the algorithm is able to add the predicted next state to the state machine and then predict the next state, until the whole state machine signature has been predicted. Subsequently, it is passed to the clustering algorithm based on the FP-growth algorithm (as discussed in section 5.2 in chapter 5). After predicting the behaviour signature of the current malicious web page, the prediction model attempts to link it to pre-defined behaviour groups using our clustering model. The prediction model has two options: (a) find similar behaviour activities between the current malicious behaviour signatures and others in the same behaviour group; or (b) create a new behaviour group if no similarities are found.

The malware database contains the web-based malware behaviour group by using the FP-growth clustering model. The output of the behaviour families need to be stored in an organised database. The database contains all the state machine signatures, path table, the groups' labels and the behaviour groups obtained from the clustering algorithm. The path table contains the absolute path of the source and destination of malware changes in the victim's system. When a file system changes, it saves the MD5 of the files sent to the system. It is used by prediction and classification models to identify state machine signatures or predict future behaviour from similar state machine signatures. The database is updated regularly by FP-growth in case it adds a new state machine or creates a new behaviour group from new attack sequences.

6.1.2 Prediction algorithm

In this algorithm we work with a set of malicious behaviour activities encoded within various state machines. These behaviour activities are associated with a malware M. Using a mechanism of probabilities we try to identify the next possible behaviour activity that a malware M can perform. We work with a set of state machines $N=\{N_1,N_2,N_3, \dots, N_n\}$, where each N_i identifies a set of states such as $N_i=\{S_1,S_2,S_3,\dots, S_m\}$. It is important to note that each state $S_i \in N_j, i \neq j$ represents malicious behaviour in the victim system. The prediction algorithm pseudo-code is shown in Listing 6-2.

```

1.For i=0 to n
2. Begin
3.   capture(b,M)
4.   encode(b,S,N)
5.   state_prediction(S,L)
6.   For S=0 To L
7.     Begin
8.       retrieve(Si)
9.       predict(Si,P)
10.      add(P)
11.    End For
12. End For

```

Listing 6-2: the prediction model pseudo-code.

The algorithm starts capturing a number of behaviour activities from malware M by generating the state machine N while monitoring the malware in real-time. The set $N = \{N_1, N_2, N_3, \dots, N_n\}$ contains various state machines where each N_i identifies a set of states such as $N_i = \{S_1, S_2, S_3, \dots, S_m\}$. At the next step we convert all the behaviours from the malware to the state machine version S and set the number of state prediction L. By looping through the states we retrieve similar state machine sequences and obtain similar state machines from the database that share similar states and calculate the highest possibility of the next state P. Furthermore, the predicted state P is added to the state machine N.

We make use of the Hidden Markov Model (HMM) to fit the data within a certain tolerance threshold and the Baum-Welch learning algorithm for discrete density. The HMM uses the initially defined sequences as the input variable and returns a number of symbols. We work with a transitions matrix for this model which contains the number of transition probabilities as value. Furthermore, we calculate the most likely sequence of hidden states that produces the given observation sequence in relation with a sequence of observations and the optimized state probability. More specifically, given the HMM $M = (A, B, \pi)$ and the observation sequence $O = \{O_1, O_2, O_3, \dots, O_k\}$, the most likely sequence of hidden states S_i that produced this observation sequence O can be calculated. In the final stage the algorithm predicts the next states that will occur after a given observation sequence.

The Baum-Welch learning algorithm is used to compute the discrete density Hidden Markov Models. The Baum-Welch learning algorithm computes the forward and backward probability matrices for a given observation referenced by its index in the input training data. The index identifies the index of the observation in the input training data and the output resides in a matrix containing the computed backward probabilities. The Baum-Welch learning algorithm works with an array of observation sequences to train the model and returns the average log-likelihood for an observation after the model has been trained. More specifically, given some training observation sequences $O = \{O_1, O_2, O_3, \dots, O_k\}$ and the general structure of HMM (numbers of hidden and visible states), it determines the HMM parameters $M = (A, B, \pi)$ that best fits the training data. In the final stage the observations in the training data and the probability matrix are used to update the probability distributions of the next states. Listing 6-3 shows how we calculate the possibilities based on forward and backward algorithm.

$$\forall A_i = \{S_1, S_2, S_3, \dots, S_n\}, S_k \in A_i, S_k \rightarrow S_i O_j = x, \exists I = \{I_1, I_2, I_3, \dots, I_m\}, I_i \rightarrow S_i O_j = n, m < n.$$

1. Forward/Backward algorithm: $P(I_k/S_i)$
2. Forward: $P(I_k/S_i) \forall S_i \in A_i$
3. Backward: $P(S_{k+1:n}/I_k), \forall I_k \in I$

Listing 6-3: the forward and backward probability algorithm.

6.1.3 Implementation

The prediction algorithm was developed using the .NET programming language and used the Accord.NET Framework [119]; Accord.NET is a C# framework that includes many algorithms, such as the Bayesian regularisation for Neural Network training, K-Means and Hidden Markov Models. Fig 6-3 shows the prediction model interface.

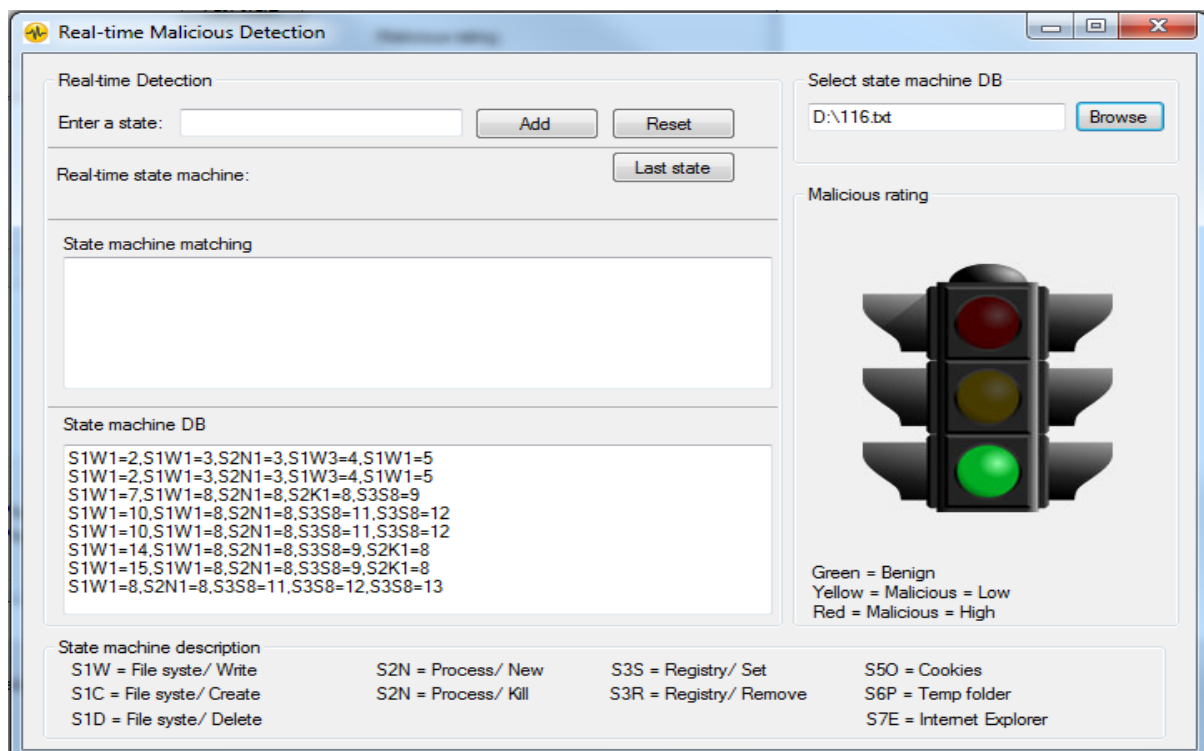


Fig. 6-3: the prediction model interface.

The interface provides a ranking for each state machine signature based on its impact on the end-user's system. The ranking system has three different categories:

1. **Green:** when the web page does not attack or harm the end-user's system and only performs benign actions such as inserting cookies.

2. **Yellow:** when the web page tries to attack the end-user's system by installing files on the system but does not execute them. In this case, the ranking system will give it a yellow colour as it does not attack the end-user's system immediately, but might do if it executes certain malicious behaviour (e.g. changes such as changing the user's files or edits the registry).
3. **Red:** shows that the web page exploits the end-user's system by installing and executing malware, deleting files, establishing network activities or editing the registry by adding or deleting values without the user's permission.

As shown in Fig 6-6, the application is invoked with a known first state: "S1W1=2". The application then finds a similar state machine signature from its malware database. It passes the input state (S1W1=2) along with similar state machines, of which there are three in our example to the prediction algorithm. The prediction algorithm then calculates the highest probability for the next state. From Fig. 6-4 there are two possibilities ordered according to their likelihood, starting with (S1W1=3) and followed by the most likely next state (S1W1=6).

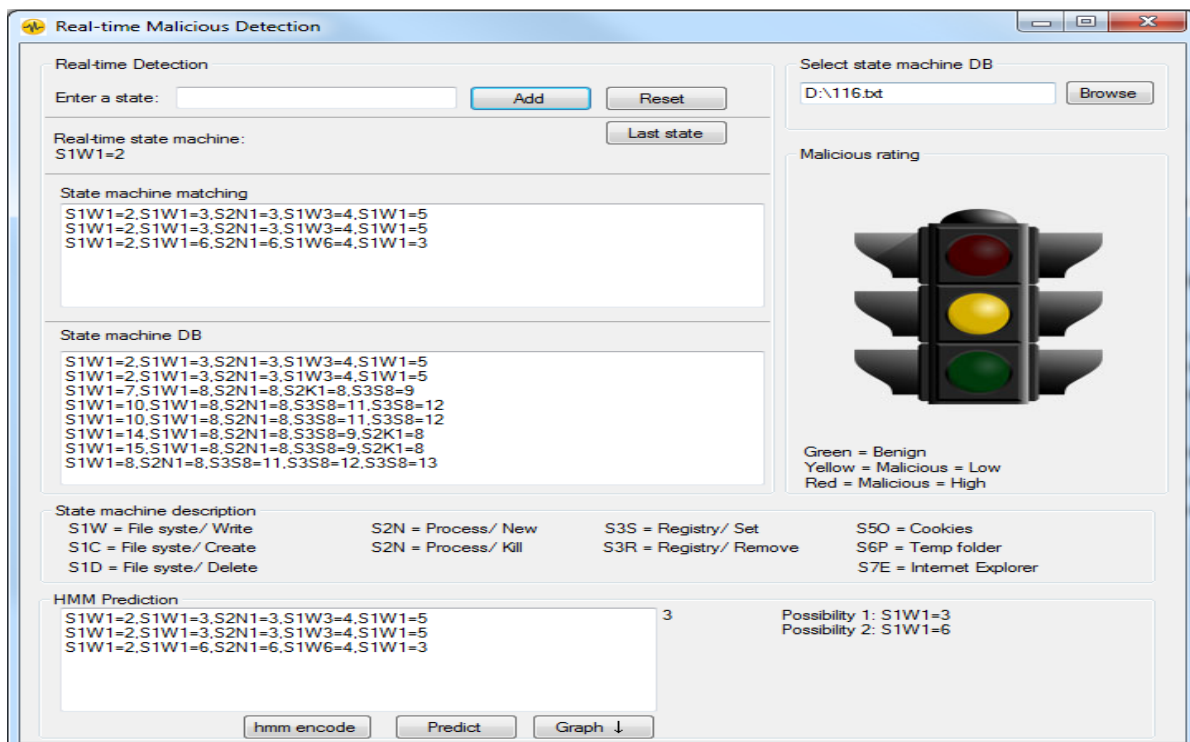


Fig 6-4: the user interface showing prediction outcomes

6.2 Classification

A classification model is used to match a given behaviour to a group formed using the clustering approach discussed previously. The classification model can be used with an initial set of states as interaction with a web server is progressing, or by passing a whole state machine signature to find a similar behaviour group after predicting it, in case it cannot find the behaviour group; then it will pass it to the FP-growth clustering algorithm, which is able to find a similar group or create a new one in case it has new attack sequence.

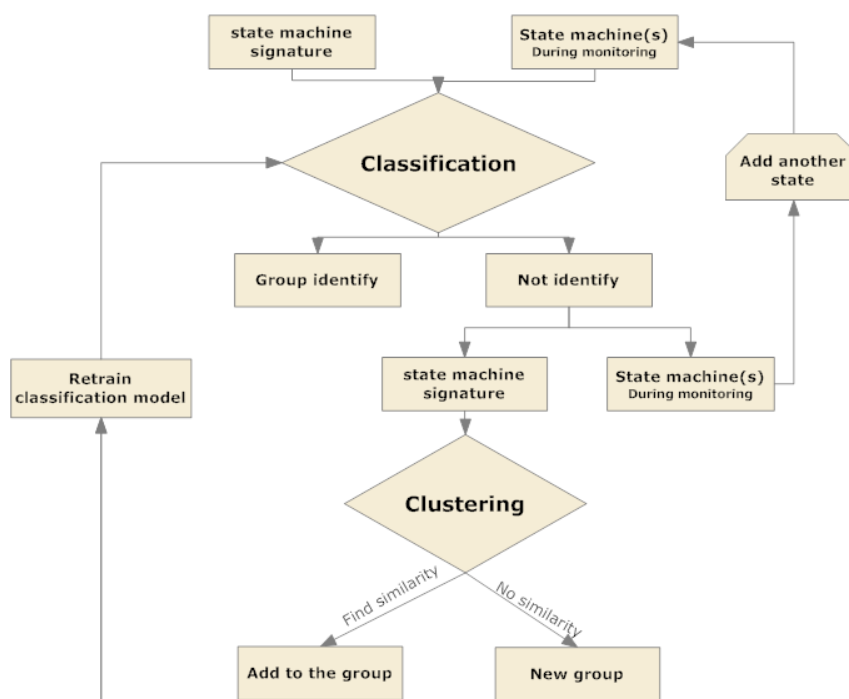


Fig. 6-5: classification model.

The classification model, as shown in Fig. 6-5, is the one we use to apply to the state machine groups. First, the model will receive the state machine groups from the clustering algorithm, based on the FP-growth clustering algorithm. Then a state machine is passed to the classification model from the prediction model, which is used to identify the state machine behaviour group. The model can operate in two modes:

1. State machine signature: to identify the malware group for the malicious web page after scanning is completed; that is, after the entire interaction with an external web page has been completed.

2. During monitoring in order to identify the malware group while scanning the malicious web page: that is, while the interaction is in progress. The model will add one state at a time to the state machine and then try to identify the group.

If the classification model cannot identify a relevant group, the state machine is forwarded to the clustering algorithm, although it takes more time to find any similarities between it and our state machine group's database. If it cannot find any similarities then it will be identified as a new group.

6.3 Summary and Discussion

We have completed our web-based behaviour system which we believe is one of the first of its type that is able to analyse and predict drive-by download attacks based on malware behaviour. The system is able to encode web-based malware behaviour from the client Honeypot, by the state machine model and then detects any changes by dynamic anomaly detection as it profiles all benign behaviour of the operating system and applications, and then detects any malicious behaviour based on the pre-scanned web based malware behaviour. When it detects malicious behaviour from the target web page, it will then generate the state machine corresponding to the actual change on the client Honeypot system. Therefore, the system will use the prediction model, HMM, to predict the possible malicious future behaviour from the initial malicious states in multi-stage processes, therefore, the model will predict the possible next state while monitoring malicious web pages in real-time. Moreover, when it predicts future behaviour successfully, by matching the predicted state with the monitored state from the malicious web page in multi-stage processes, then it will attempt to predict the state machine signature and pass it to the clustering model. Our clustering model is based on FP-growth which is able to find the common states between the given state machine signature against the previously discovered one in the database. When the state machine finds the behaviour group of the given malicious web page, it will include it within its behaviour group and perform the sub-group clustering phase which aims to find more similarities between state machine signatures of the same group. On the other hand, when the clustering model fails to find a similar behaviour group for the given malicious web page behaviour, then it assumes it is unknown behaviour and creates a new behaviour group for it.

The next chapter presents the evaluation to test our web-based behaviour system to detect, analyse and predict real web-based malware that causes risk to users just by visiting those web pages. Furthermore, we demonstrate a hybrid system that can combine dynamic and static malware analysis methods together. We propose a system that uses our web-based behaviour system which is able to detect malware dynamically with a low interaction client Honey-pot tool called Honeyware, which is able to detect malicious web pages by using static signature-based detection.

Chapter 7

Evaluation and Usage

This chapter evaluates the proposed web-based behaviour system which contains three main parts: state machine encoding, clustering and prediction models. The system has two main methods: the first method scans the malicious web pages offline and models their behaviour using a state machine as described in chapter 4. These state machine signatures are then fed into the clustering model which is able to build behaviour groups which contain similar web-based attack behaviour. The second method is the online mode which monitors the system state while interacting with a web page in real time, and detects any present and future malicious behaviour using the malware database which contains the previously-scanned web-based attacks organized into behaviour groups.

This chapter has two main sections. The first section discusses the use of the low-interaction client Honeybot by creating a tool called Honeyware. The main reason for developing this tool is to improve the accuracy of malicious Web page detection as our scanning engine, Capture-HPC, cannot detect all types of malware. Therefore, we used Honeyware as a signature-based detection scheme to find malicious codes within the target server. The high-interaction client Honeybot which we used can detect attacks, such as drive-by download attacks. However, it cannot detect some malicious Web pages, such as when there is a need to interact with a malicious Web page in order to trigger the malicious behaviour or to download an executable file in order to affect the end-user system. Therefore, we propose the use of both low- and high-interaction client Honeybots to detect Web-based attacks. The second section shows the experiments used to evaluate Web-based behaviour system by

scanning and monitoring real Web-based attacks. Finally, we demonstrate how the proposed hybrid system works by combining both low- and high-interaction client honeypots to improve the accuracy of detecting web-based attacks by signature- and behaviour-based detection.

7.1 An overview of Honeyware

Honeyware is a low interaction client honeypot; the aim is to combine it with our web-based behaviour model in order to build a hybrid system that is able to detect malware based on its behaviour and to use anti-malware products to detect the malicious code statically. It combines the benefit of web-based tools, which run on local or remote servers, with the ability to access the tool from a web browser, which is important for many different devices, such as PCs or mobiles. Another important function of this tool is to give the user the ability to test the target server with many of the available web browsers and to scan the target with five different scan engines. Honeyware is written in PHP and is an open source tool. It was developed to solve some of the problems and challenges in low-interaction client Honeypots. Its features are:

1. **Web browsers:** Honeyware has the ability to simulate different web browsers including Internet Explorer, Firefox, Opera, Chrome, Safari and Konqueror.
2. **The Scan engine:** Honeyware has the ability to scan target files with different scan engines to determine whether the target has malicious code or not. Some scan engines are able to detect viruses and malware that other products cannot. The scan engines that Honeyware supports are: AVG 7, F-Prot 5.2.1.4252, Avast 1.3, ClamAV 0.95.2 and AntiVir 7.8. An example of the benefits of using different scan engines is the exploit CVE number CVE-2007-5601. This vulnerability is in RealPlayer software [121]. This vulnerability was scanned with five different scan engines but only AVG and Avast out of the five scan engines detected it.
3. **Crawling:** Honeyware supports search engine crawling to obtain URLs to scan. Honeyware supports two main search engines: Yahoo! and MSN.
4. **Honeyware Client:** Honeyware implements an important client tool that can interact with the target server to return the files downloaded to Honeyware and examine them. This feature is very important for solving geolocation-dependent problems.

7.1.1 The motivation for using Honeyware

Client Honey pots visit and interact with suspect web sites in order to detect and collect information about malware to protect users from malicious websites or to allow security professionals to investigate malicious content. We present the idea of using web-based technology and integrating it with a client Honey pot by building a low interaction client Honey pot tool called Honeyware. Honeyware is able to detect known malicious web pages and it is able to detect more known attacks than the high interaction client Honey pot as shown in the experiment section 7.3. It checks all the HTML files returned and can detect the presence of malicious code or malware which high interaction systems cannot, because such attacks require interaction with the users, such as downloading a file or clicking on a web page. In a hybrid system, the idea is to pass all URLs to a high interaction client Honey pot, which is the basis of our web based behaviour system, then pass suspicious URLs to Honeyware to validate the results and to provide another layer of analysis. In theory, this technique will be quite slow, as it needs it download the Web page files first and then scan them by using a number of anti-virus products. However, it will detect more malicious Web pages at a cost of the detection speed.

7.1.2 Overcoming client Honey pot challenges

Honeyware was built to solve some of the challenges inherent in a low-interaction client Honey pot, as discussed in section 2.4.6. Attackers take advantage of these problems to prevent their detection by client Honey pot tools and so deliver malicious binaries to a wide range of people. There are already some tools available for low-interaction client Honey pots, but it is hoped that Honeyware will improve upon these.

The first challenge is the use of IP tracking to delay malicious code according to the number of website visits. There are two possible ways to solve this.

1. Visit the website multiple times to give the malicious web page an appearance of usual human visitor behaviour. This way, the client Honey pot will need to download the target files and delete them. This method is not reliable as it requires more time to perform and causes more traffic.
2. Visit the target web page as a web spider, which connects to the target web page and makes an HTTP request to the target but does not save the target response files to the

Honey pot machine. This would be much faster than the first method and it could also send multiple web spider requests simultaneously.

However, with both methods the malicious web page will detect the multiple requests to the web page and could block the Honey pot or behave as if it were benign. One way to solve this is to use a client with a different IP address, to make multiple requests and then to use the Honeyware tool to scan the target machine again. In this scenario, Honeyware would be able to compare both results and check if the web page used any techniques to mask its malicious behaviour. This scan would take some time to complete as it would need to request the web page twice, once through Honeyware and once through the client. The client would need to request the web page multiple times using a Wget tool in spider mode, using the web spider method of interaction with the target and not saving any response to the local machine. Fig 7-1 shows the IP tracking scenario using Honeyware.

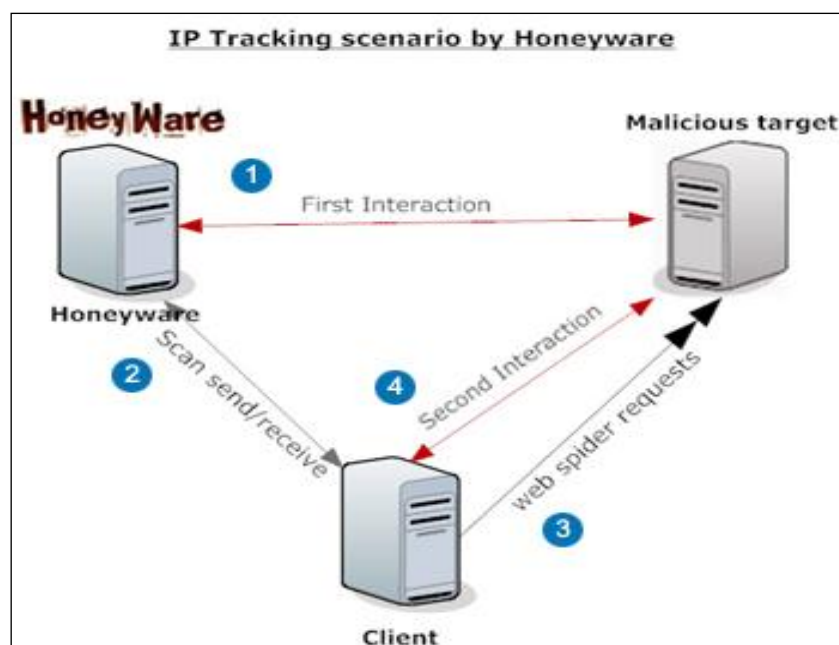


Fig 7-1: IP tracking scenario using Honeyware.

The second challenge is the use by attackers of geolocation to target visitors from certain countries. The Swiss security blog [122] conducted an experiment using a Honey pot to find sites that collected FTP credentials from different victims by attacking them first from a malicious website and then exploiting vulnerability on the victim's PC to capture the FTP credentials and send them to a control server. The results showed the top country attacked was the USA. The Honeyware operator could therefore install a client in the USA, even if the main Honeyware server is elsewhere.

Using this technique, Honeyware can successfully detect websites that use geolocation-dependent features provided by tools such as MPack [123]. It does this by placing Honeyware clients in locations that appear to be of high value to the attackers and then linking them to the Honeyware tool to perform different scans from different locations, comparing the results to check if there are any changes and then detecting whether a malicious web page used a geolocation-dependent feature.

7.1.3 Honeyware architecture

Honeyware is able to scan any target server with a valid URL. The user can choose the web browser that Honeyware will simulate by using user agent strings as described in Table 7-1. The benefit of this is that some attackers use web-based tools such as MPack [123] to attack only visitors using a certain browser, or even a certain version of a browser. For example, Internet Explorer version 6 suffers from a remote code extension vulnerability (CVE-2006-5579 and MS06-072) [124], [125]. The attacker can therefore check the visitor's web browser version and then only infect them if they use the same vulnerable web browser version.

Honeyware is able to scan any target server with a valid URL by file or by crawling. The file feature lets the user take advantage of blacklists produced by security or non-profit organisations such as *www.stopbadware.org*, or the user can make a file with valid URLs on each line and then choose from it. Honeyware will upload the blacklist file (.txt) to a temporary folder and then extract all the URLs to check whether they are valid or not. If the URLs are valid, it will scan each URL and, after interacting and scanning, it will continue to the second URL until interactions with all URLs in the file have been made. After choosing the target server, the user will be able to choose the web browser that Honeyware will simulate and then interact with the target server as the specific web browser. The benefit of this is that some web-based Malware tools, such as MPack, give the attacker the ability to target visitors who use Internet Explorer or Firefox, or they may choose to exploit a version of Internet Explorer, such as version 6 as discussed in Microsoft Security Bulletin MS06-072. The attacker can exploit this to check the visitor's web browser version and then only infect them if they use this one. Honeyware can simulate the web browser by using a specific user agent string for each web browser supported. The user agent is a string sent first to the target server in order to visit it. The user agent includes important information such as the web browser name, version and name of the host operating system. Honeyware supports most of

the available web browsers that attackers might try to infect, with additional web browser products also available to give the user the ability to test various targets. Honeyware supports six popular web browsers, along with some of the versions of popular web browsers that are targeted by attackers for known vulnerabilities. Table 7-1 shows the web browsers supported by Honeyware along with their user agent strings.











	The browser	The user agent
	Internet explorer 5	Mozilla/4.0 (compatible; MSIE 5.0; Windows NT 5.1; .NET CLR 1.1.4322)
	Internet explorer 6	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)
	Internet explorer 7	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
	Internet explorer 8	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
	Firefox 2	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.8.1.6) Gecko/20070725 Firefox/2.0.0.6
	Firefox 3	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-GB; rv:1.9.0.6) Gecko/2009011913 Firefox/3.0.6
	Safari	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0 Safari/528.16
	Google Chrome	Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US) AppleWebKit/525.19 (KHTML, like Gecko) Chrome/1.0.154.48 Safari/525.19
	Opera	Opera/9.00 (Windows NT 5.1; U; en)
	Konqueror	Mozilla/5.0 (compatible; Konqueror/4.1; Linux) KHTML/4.1.2 (like Gecko)

Table 7-1: Web browsers supported by Honeyware with each user agent string.

For example, the user agent can be used to detect the browser and operating system of a visitor and then redirect that user to an exploit that works specifically against their system, which is a method used by MPack web-based exploit tool version 0.90 [123]. Fig 7-2 shows MPack's method of attack using the browser's product and version. It shows how MPack reads the visitor user agent and determines the browser product and version; based on this it will redirect the page to a corresponding exploit file that is able to compromise a known vulnerability in the visitor's web browser. For example, Mpack exploits vulnerability (MS06-044 [132]) if the visitor is using Internet Explorer version 5.

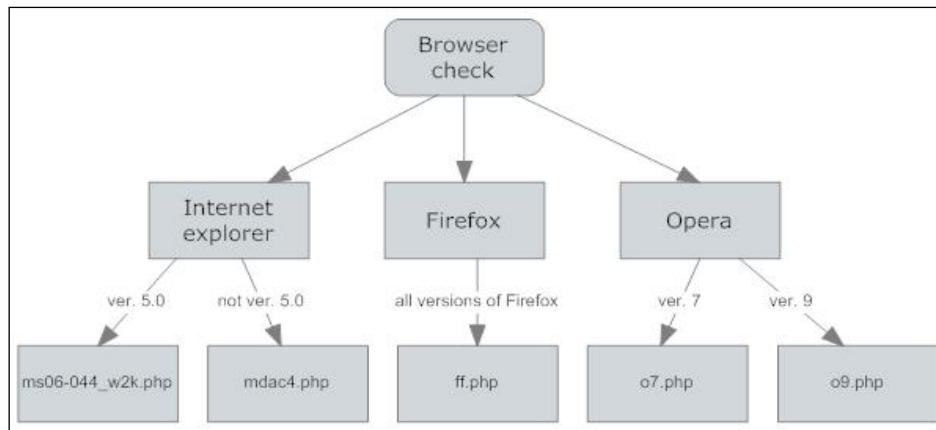


Fig 7-2: Mpack attack methods.

7.1.4 Honeyware design

Honeyware is designed and presented in a way that makes it easy to understand and uses some default options to make it easy for both professionals and end users to operate. Honeyware is also able to save all scan and crawling results into a database to create archives for easy data storage and provides the ability to delete the scan or crawling results. Fig 7-3 shows the Honeyware architecture. It shows that Honeyware starts by receiving the URL of the target web page, which can be inserted as an individual web page or a list, or by crawling search engines using specified search terms. The next stage is to emulate a specific web browser by using each browser's user agent while interacting with the external server. The user also has the ability to select the anti-virus products to use to scan the downloaded files. It also supports downloading all the website files or only to download custom file extensions, such as executable files (exe). The number of downloads is responsible for setting how many times the website is downloaded; previous versions are deleted in order to detect any changes in the version by the external server, which might affect the visitor in different ways according to the number of visits to ensure the visitor is human and not an automatic tool. When using the Honeyware client, a scan request will be sent which will perform the download and the files will be sent back to Honeyware; alternatively it will download the files using the same tool and then scan them using the selected anti-virus tools. Finally, it will present the result to the user, as shown in Fig 7-3.

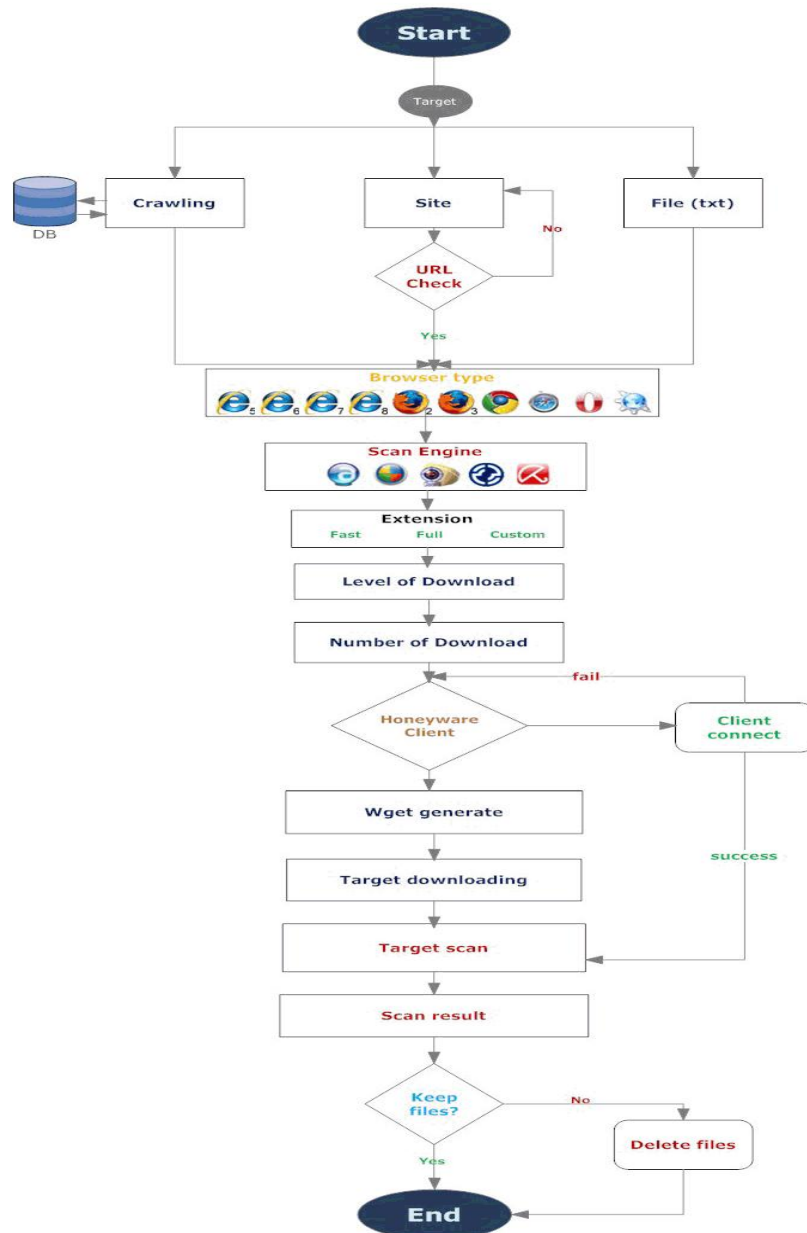


Fig 7-3: Honeyware design.

The main page, shown in Fig 7-5, has two main parts, the scan form and the archive. A user is able to scan by completing the form and changing the default settings, such as the web browser or scan engine. The scan archive displays the name of the scan and additional information, such as the web browsers and scan engines used. The user can delete any scan from the main page or from inside each scan. Fig 7-4 shows the scan results given by Honeyware. It shows the example of downloading a JavaScript file by emulating IE 5 from the local system. It is then scanned using the five supported anti-virus tools (AVG, F-Prot,

Avast, ClamAV and AntiVir). The result indicates that it has been detected as a malicious file by only one anti-virus tool (Avast).

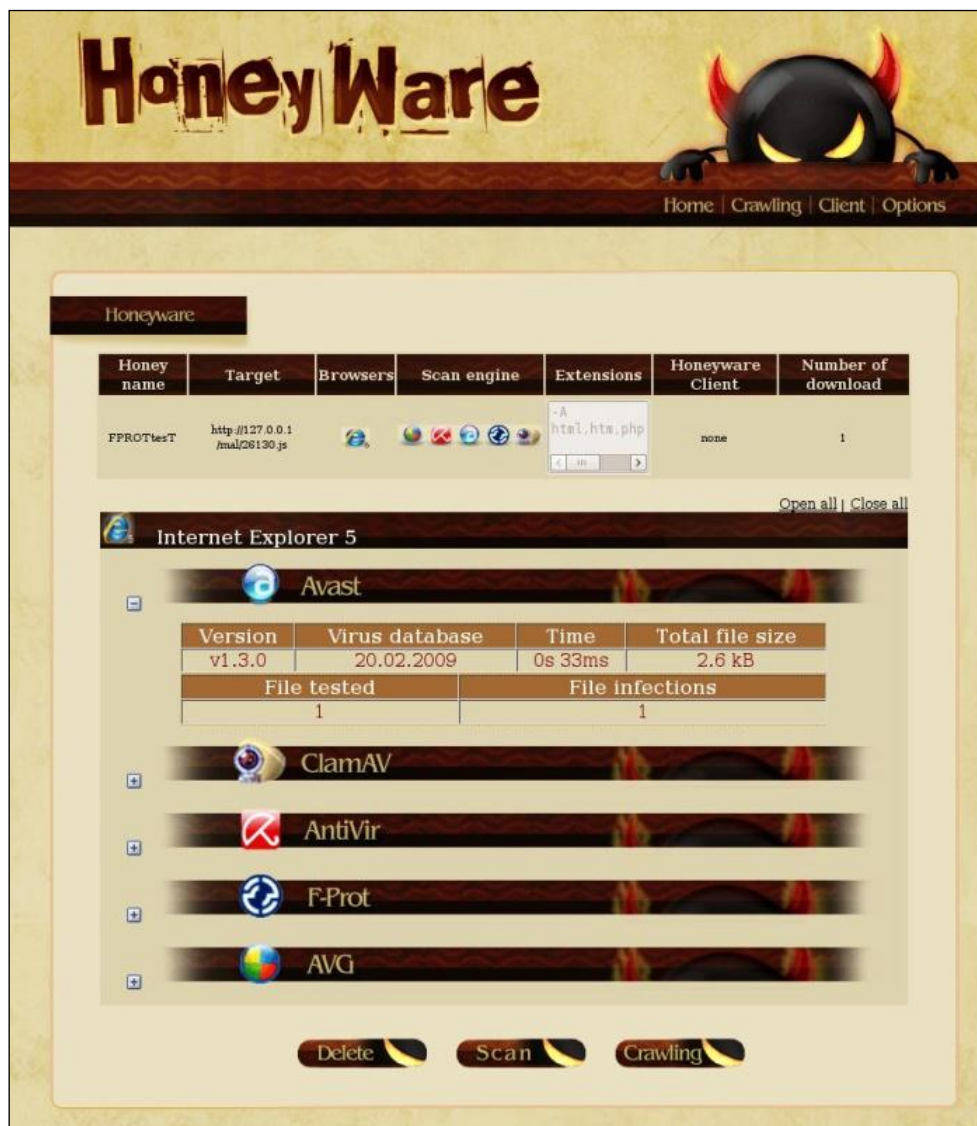


Fig 7-4: Honeyware results page.

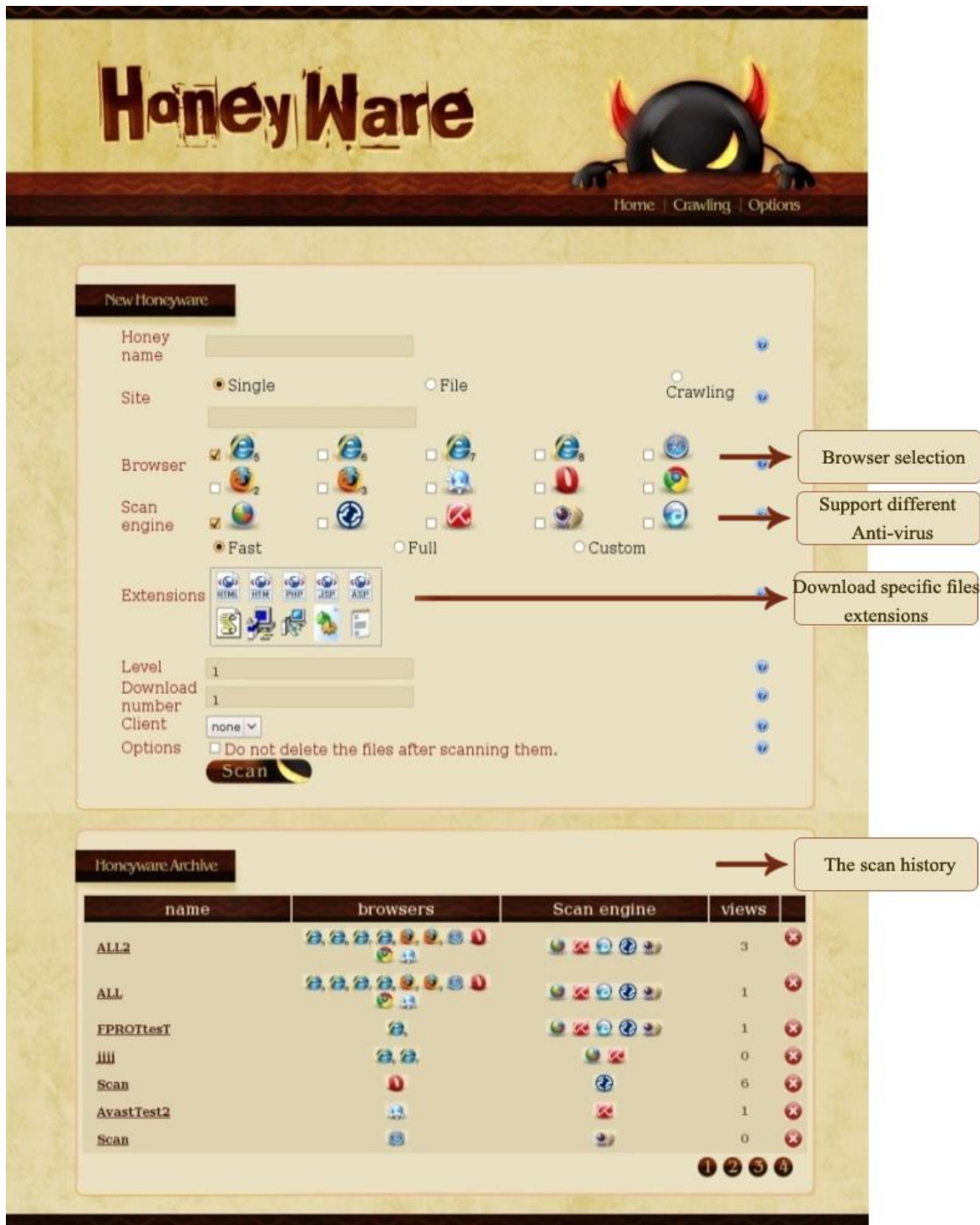


Fig 7-5: The Honeyware main page.

7.1.5 The Honeyware client

Honeyware provides a new and powerful feature for low-interaction client Honeypots. As previously discussed, the MPack tool allows visitors from specific countries to be targeted [85]. To overcome this, the Honeyware client, which can be set up in different locations, was created. The structure of the Honeyware client is simple; it is a single PHP file and an empty directory.

A client will interact with Honeyware via the web server on port 80, and this port has to be open so a connection can be established between Honeyware and its clients. However, the targeted malicious server can easily scan the visitor's port address. Because the normal end-user does not need to open port 80, if the port is open then the malicious server may assume it to be a Honeypot and so interact safely with the visitor. There is an easy solution for this, which is implemented while installing the Honeyware client. It uses iptables [133], a free and open source tool that is shipped with most major Linux distributions. Iptables can be configured to only allow connections from the IP address of the Honeyware server and to reject any other attempts to connect, thus preventing the attacker scanning port 80 to see if the Honeyware client exists. Fig 7-6 shows the communication between Honeyware and its clients.

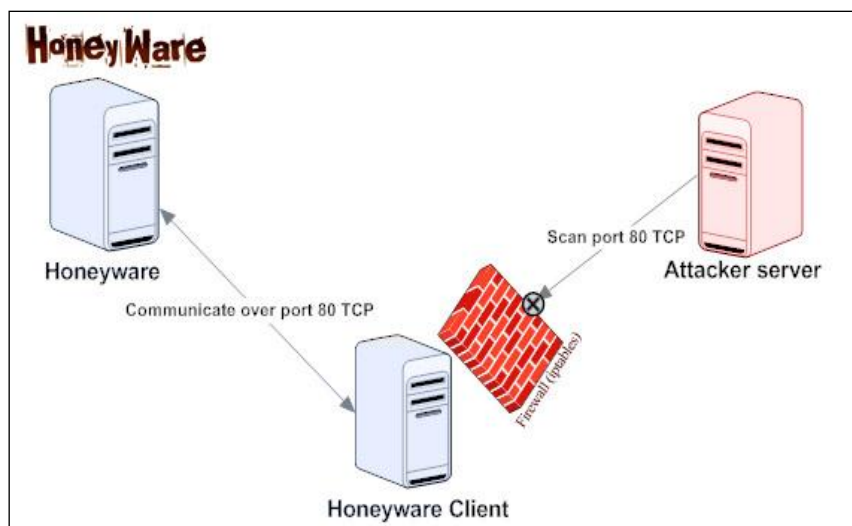


Fig 7-6: Communication between Honeyware and its clients.

7.2 Web-based behaviour model evolution overview

This section will evaluate our Web-based behaviour system, which has three main models: state machine and clustering and prediction models, which all work together to detect and predict malicious Web page behaviour. The evaluation covers the three models, first by collecting the malicious Web pages and then scanning them using Capture-HPC. Second, log files are used to encode them to state machine signatures before they are passed to the clustering model to build the malware database. Finally, based on HMM, we conducted a number of experiments to validate the prediction model to detect 10 malicious Web pages by predicting their potential malicious behaviour.

7.2.1 Data collection

The scenarios in our experiments are based on real web-based attacks available online; not on offline attacks tested in the lab. This provides a more reliable method to test our proposed system in the real world and to validate our assumptions. The data consists of a set of online malicious websites that are able to attack a user just by them visiting a web page; this is called a drive-by download attack [82]. The malicious URLs have been collected from public sources that provide lists of known malicious URLs and IP addresses. We used two main public sources to collect the malicious web page URLs and IP addresses [116] [117]. In addition, we have an updated list that includes all the malicious URLs that have been attacking Cardiff University systems. The list is provided by Mr Damian Southard, a security engineer at Cardiff University, who helped our research from the beginning by providing malicious URLs and the IP address of attacks on Cardiff University. We were able to select 120 live malicious URLs that we could use to test our system. The problem, as mentioned in the previous section, is that it is hard to find live malicious web pages as they regularly change their URLs and IP addresses to hide from blacklists and security companies; therefore, we found that hundreds of the known malicious websites were down and not available to test.

7.2.2 Experiment methodology

The aim of this experiment is to evaluate our web-based behaviour system to model, group and predict malicious web pages in order to protect the end user against this type of attack. The experimental data will be based on the URLs identified in the previous section and then

separated into two parts: 110 URLs with which to create the malware database, and 10 URLs of malicious web pages for testing in order to detect their malicious behaviour and match them to the behaviours stored in the database. Generally, the experimental methodology has two stages: the first stage is to build up the malware database by scanning malicious URLs collected using high-interaction client Honeypots and the state machine model. The second stage of the experiment is to monitor malicious web pages online and to attempt to predict their future malicious behaviour, based on their actions on the system's state. Firstly, each of the experimental URLs is scanned using Capture-HPC, together with the state machine model, in order to visualise and encode the behaviour of the malicious web pages and to build the malware database. The malicious web page is loaded into the client, which is a default installation of Windows XP Service Pack 2 with Capture-HPC client software. Capture-HPC will load the URL into Internet Explorer version 6, monitor the system to detect any changes made by the malicious web page and then send it to the Capture-HPC server. The state machine also receives the changes made on the client by the web-based attack and visualises them in the security analyser, as shown in Fig 7-8. It shows that the state machine has 3 main states (File system, Process, Registry) as well as the Start (S0) and End (S5) states. Secondly, in each change to the system state, the prediction model measures the similarity between the applied changes and the malware database, as illustrated in Listing 6-2. If the model detects any malicious behaviour in any stage of scanning, it then blocks the interaction with the malicious web page, passing it on to a sandbox environment to determine more about its behaviour – without harming the user's system. If known malicious similarities are found, then it predicts the possible future malicious behaviour of the current web page and reports this back to the system.

7.2.3 Experiments

Our experiments start by building a malware database which includes the state machine signatures grouped into main and sub behaviour groups. First, we pass each URL to our scan engine, Capture-HPC, and then generate a state machine signature based on the behaviour of the web-based attack. Fig. 1-1 (in Chapter 1) and 7-7 show how the state machine is generated in this experiment.

An experiment was conducted to show how the state machine is generated as well as to demonstrate how both signature and behaviour-based detections differ. A piece of malware called Poison Ivy, which is a Remote Administration Tool (RAT), is used by attackers to access victims' systems remotely. Poison Ivy is a well-known backdoor malware and therefore it is detected by most anti-malware products by using signature-based detection (based on known hash). The scan results of the Poison Ivy malware are shown in Figs 7-8.

Home » Services » Multi-Engine Antivirus Scanner

Trojan Removal Software -> Free Detection. Removes Spyware, Adware & Popups. Download Now. www.STOPzilla.com Ads by Google

Permalink :: Email a Friend :: Print this Page

File information

Report date: 2010-10-20 10:46:31 (GMT 1)
 File name: trojan-clean-exe
 File size: 6656 bytes
 MD5 hash: d2a64a222dc2e94f14479a5467c54b96
 SHA1 hash: 03277470a9fc338a9d487604b85d2e815caa6c4c
 Detection rate: 15 on 16 (94%)
 Status: **INFECTED**

Antivirus	Database	Engine	Result
a-squared	20/10/2010	5.0.0.20	Virus.Win32.Poison!IK
Avast	20/10/2010	5.0	Win32.Tiny-ADY [Trj]
AVG	20/10/2010	9.0.0.725	BackDoor.Generic9.MQL
Avira AntiVir	20/10/2010	7.6.0.59	BDS/Poison.v.A.8704
BitDefender	20/10/2010	7.0.0.2555	Trojan.Keylog.ZKT
ClamAV	20/10/2010	0.96.2.1	Trojan.Downloader-24568
Comodo	20/10/2010	4.0	Backdoor.Win32.Poison.NAN
Dr.Web	20/10/2010	5.00.0	BackDoor.Poison.685
F-PROT6	20/10/2010	4.6.1.107	W32/Agent.G.gen!Eldorado
Ikarus T3	20/10/2010	1001084	Virus.Win32.Poison
Kaspersky	20/10/2010	9.0.0.736	Backdoor.Win32.Poison.aec
NOD32	20/10/2010	4.2.42.0	Win32/Poison.NAE
Panda	20/10/2010	10.0.3.0	
TrendMicro	20/10/2010	9.120-1004	BKDR_POISON.DS
VBA32	20/10/2010	3.12.14.1	Backdoor.Win32.Hupigon.dguz
VirusBuster	20/10/2010	1.5.6	Trojan.DL.CKSPost.Gen

Extra information

Menu

- Scan File
- Statistics
- Sample Report
- Credits

Quick Links

- Products
- Services
- Virus Scanner
- Research Blog
- Support
- Company

Services

- Site Worth
- Site Status

Help

- FAQs
- Forum
- Email

Fig 7-8: the scan results of Poison Ivy on <http://vscan.novirusthanks.org>.

Furthermore, we used a type of packer software (publicly available) which was able to encrypt malware in order to avoid the detection by anti-malware products. The basic idea was to change the code of the malware involved as the signature-based detection only attempts to match the known signatures against the malware code. This is a problem as, if no matches are found, then the anti-malware software will not detect malware even if it is the same malware (before encryption) with a slight change in its coding. The scan results of the encrypted Poison Ivy malware are shown in Figs 7-9.

Permalink :: Email a Friend :: Print this Page

File information

Report date: 2010-10-20 10:46:31 (GMT 1)
 File name: trojan-encode-exe
 File size: 80389 bytes
 MD5 hash: 6015303fc4be3913f46c5396b3a06f2e
 SHA1 hash: 7972c9bf5427c8a863625a5d91f440c299ea94e8
 Detection rate: 6 on 16 (38%)
 Status: **INFECTED**

Antivirus	Database	Engine	Result
a-squared	20/10/2010	5.0.0.20	Trojan.Win32.Ircbrute!IK
Avast	20/10/2010	5.0	Win32.Malware-gen
AVG	20/10/2010	9.0.0.725	
Avira AntiVir	20/10/2010	7.6.0.59	
BitDefender	20/10/2010	7.0.0.2555	Trojan.Generic.KDV.44945
ClamAV	20/10/2010	0.96.2.1	
Comodo	20/10/2010	4.0	
Dr.Web	20/10/2010	5.00.0	
F-PROT6	20/10/2010	4.6.1.107	
Ikarus T3	20/10/2010	1001084	Trojan.Win32.Ircbrute
Kaspersky	20/10/2010	9.0.0.736	HEUR:Trojan.Win32.Generic
NOD32	20/10/2010	4.2.42.0	
Panda	20/10/2010	10.0.3.0	
TrendMicro	20/10/2010	9.120-1004	
VBA32	20/10/2010	3.12.14.1	
VirusBuster	20/10/2010	1.5.6	Trojan.Injector.YOU

Extra information

Fig 7-9: the scan results of the encrypted Poison Ivy on <http://vscan.novirusthanks.org/>.

As shown in the scan results in the aforementioned figures, there are 9 anti-malware products which failed to detect the same malware (Posion Ivy) which had been encrypted. However, it should be noted that we scanned the behaviour of both the clean and the encrypted versions of the Poison Ivy. The same behaviour for these two versions is evident in Figs 4 and 5; therefore, we can conclude that by using behaviour-based detection, we will be able to detect malware even if it is encrypted, as distinct from the signature-based detection which cannot detect most forms of encrypted malware or those forms using packers.

The state machine poison ivy clean version:

S0, S2N1=2, S3S2=3, S4

The state machine poison ivy encrypted version:

S0, S2N1=2, S3S2=3, S4

The state machine URL table:

ID	URL/ Path
1	C:\WINDOWS\explorer.exe
2	C:\Documents and Settings\Yaser\Desktop\o\trj\trojan_encode.exe
3	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\rr

Table 7-3: the path table of poison ivy.

we

After we successfully encode the 110 URLs as state machines, as described in section 4.2.2, we pass their state machine signatures into the FP-Growth clustering model. FP-Growth then searches for common behaviours between the state machine signatures and builds behaviour groups based on these common behaviours, as illustrated in the pseudo-code in Listing 5.1 in Chapter 5.

The outputs of the previous clustering model are the main and sub behaviour groups for the 110 URLs in which the sub groups show greater similarities, as shown in the results in section 7.2.3. After building the malware database of the behaviour groups, we use the prediction model shown in Listing 6.1 to identify any possible malicious future action while interacting with a malicious web page. The prediction uses the malware database we have built; therefore, the more data we have in the database, the more accurately we are able to predict future malicious behaviour. The prediction model, based on the HMM algorithm, finds similarities between the current web page and the malicious behaviours stored in the malware database.

We use the system to scan ten live malicious web pages, through which we are able to scan the system while interacting with the malicious web page and then predict its future behaviours, as discussed in Section 7.2.3. There are two possible outcomes from the prediction model:

- **Match:** it then predicts the next two states and re-matches them with malicious web page behaviour from the sandbox, and if they match then it will stop and block interaction with the malicious web page, and the model will then predict the possible future behaviour based on the length of similar state machine signatures to generate its state machine, and pass it to the classification and clustering models to group it in the database.

- Does not match: it will re-select other similar state machines and predict the next state and then re-match it; in the case it matches it will follow the process described previously. However, if this too does not match then the model will stop the interaction and generate a state machine signature from the actions collected from the malicious web page.

Finally, the model sends the state machine signature to the classification and clustering models which feed it into the malware database to speed up the process and accuracy.

7.2.4 The results

The total number of state machines generated by scanned the malicious webpages as explained in Section 7.2.3 was 110, an example of the state machine signatures is given Fig 7-10 below:

... S1W1=31,S2N1=31,S1W1=29,S1W1=19 ...

Fig 7-10: one state machine signature from the malicious web page experiment.

Fig 7-10 shows one state machine signature that was generated from a malicious web page taken from the experiment URL. An explanation of the first state in Fig 7-10 is given below:

1. S1W: indicates that there is change in the file system involving the insertion of a new file into the end-user's file system.
2. 1: is the source of the change, and it indicates the file number in the path table; the file is Internet Explorer (C:\Program Files\Internet Explorer\IEXPLORE.EXE).
3. 31: the result of the change, which was a new file being inserted in the system (C:\WINDOWS\TEMPwn10039.exe). The file is a Trojan horse, trojan-clicker.win32.spywad.b, for the Windows platform and it displays notification messages for the user and hides other system notifications, such as anti-virus messages.

After modelling the behaviour of the experiment URLs, we pass them into the FP-Growth clustering algorithm to group similar behaviours into groups. We show how the clustering algorithm works in section 7.2.2. We firstly group the state machines based on the most frequently-occurring items. We also apply the same algorithm to each of the state machines in

order to identify the behaviour sub-groups. These show the similarities in the relationships between each of the main groups. This similarity measure, as presented in Listing 6.2, is used to build a rule for each main group and to identify a label (as described previously). Tables 7-4, 7-5 and 7-6 show the main group and the sub-groups formed.

Group	Group label	Number of State machines
1	S1W1=5, S1W1=19, S2N1=19	34
2	S1W1=70, S2N1=71	15
3	S1W1=31, S2N1=31, S3S31=32	8
4	S1W1=8, S2N1=8	9
5	S1W1=115, S2N1=115, S3S115=9, S2N115=61, S1W115=116,	5
6	S1W1=51, S2N1=51	4
7	S1W1=3, S2N1=3	4
8	S1W1=64, S2N1=64, S3S64=40, S3S64=41, S3S64=42, S3S64=43	3
9	S1W1=54, S1W56=57, S1W56=58	3
10	S1W1=104, S2N1=104	3
11	S1W1=45, S2N1=45, S3S47=49, S3S47=50	2
12	S2N1=71	5
13-27	1 in each group	1

Table 7-4: the FP-growth experiment results.

Group 1: Group label: S1W1=5, S1W1=19, S2N1=19 (34 state machines)			
	Sub group	Group label	numbers
	1.1	S1W1=18, S1W1=19, S1W19=21, S3S19=20	11
	1.2	S1W1=19, S1W1=23, S1W19=21, S3S19=20	9
	1.3	S1W1=5, S1W1=19, S1W1=22, S2N1=19	8
	1.4	S1W1=19, S1W1=29, S1W19=21, S3S19=20	6

Table 7-5: the third sub-groups from the FP-growth clustering experiment

Group 3: Group label: S1W1=31, S2N1=31, S3S31=32 (8 state machines)			
	Sub group	Group label	numbers
	3.1	S1W1=31, S2N1=31, S3S31=32, S3S31=33	4
	3.2	S1W1=22, S1W1=31, S2N1=31, S3S31=32	2
	3.3	S1W1=19, S1W1=29, S1W1=31, S2N1=31,	2

		S1W11=31,S3S31=32		
--	--	-------------------	--	--

Table 7-6: the third sub-groups from the FP-growth clustering experiment

For instance, group 3.2 has two state machine signatures and has the group label:

S1W1=22, S1W1=31, S2N1=31, S3S31=32

The group label means that each state machine in the group does the following:

1. **S1W1=22**: Write a file called xp.exe from Internet Explorer to the Desktop.
2. **S1W1=31**: Write a file called xp3.exe from Internet Explorer to C: drive.
3. **S2N1=31**: Execute an Internet Explorer xp3.exe executable file.
4. **S3S31=32**: Insert a registry value for “HKCU\Software\Microsoft\Windows\CurrentVersion\ Run\con” from the process xp3.exe.

An example of the behaviour groups can be seen in Group 1 in table 7-4, which has a label and a state machine. These items can be described as follows:

- Label: this field shows the states that are present in the state machine and demonstrates similarity between each state machine. The label is “S1W1=5, S1W1=19,S2N1=19”, which means that all 34 state machine signatures in this group contain this sequence of states, which starts by writing a file (a.exe), then running it and writing another file (b.exe).
- The state machine: this shows the state machine identifier stored in the database. Examples of the state machine signatures that are grouped together in the first main group, “group ID 1”, are presented in Fig 7-11 below:

S1W1=23, <u>S1W1=5, S1W1=19,S2N1=19</u> ,S3S19=20,S1W19=21
S1W1=23, <u>S1W1=5, S1W1=19,S2N1=19</u> ,S3S19=20,S1W19=21
S1W1=29, <u>S1W1=5, S1W1=19,S2N1=19</u> ,S1W19=21,S1W1=5
S1W1=22, <u>S1W1=5, S1W1=19,S2N1=19</u> ,S1W1=5,S3S19=20

Fig 7-11: four state machine signatures from Group ID 1 in Table 7-2.

The malware database includes groups of clustering outputs which form the basis of the prediction model when interacting with a web page and are used to detect and predict malicious behaviour as well as to detect the behaviour group of current web-based attacks.

The table below shows the variations for each state, indicating the diversity that exists across the scanned web pages. Table 7-7 indicates that there were 47 possible states in stage 1, 40 in stage 2, etc. The possible states are the different states the database has for each state machine; for example, if we have a state machine signature such as: $SIWI=2$. Then the next state can be one of 40 possible states for the second stage. This number reduces as similar state machine signatures are selected.

stage	Variation of states
1	47
2	40
3	45
4	63

Table 7-7: the variation of states for each malicious stage.

The experiment methodology involves taking random state machine signatures representing ten real malicious web pages and then predicting the second state for each state machine. Given the first state, the prediction algorithm selects the most similar state machine signatures and calculates the probability of occurrence each second state. Table 7-8 shows all the experiment results. It shows ten state machine signatures and predicts their second state by looking at the first state only. The calculation of the next state is based on a HMM model which uses Baum-Welch algorithm (Forward-backward probabilities) [119], as illustrated in Listing 6.3. For example, for the second state machine (ID 2), the algorithm found 9 similar state machine signatures and then found four possible predicted states for the second state, ordered according to the highest possible state. Another example is the state machine signature ID 4, which has 14 similar state machine signatures, all of which start with the state “ $SIWI=70$ ” which is the same as that in the first state machine ID 4. HMM calculated the possible next states and found 4 different states for the second state: $S2NI=71$, $SIWI=156$, $SIWI=71$ or $S2NI=157$. When the prediction algorithm does not find any similar state machine signatures for a given malicious web page, it estimates the most likely state based on the contents of the database. Overall, we were able to predict the second states (behaviours) with 80% accuracy and failed to predict the correct second state for state machine IDs 6 and 8.

Web page ID	No. similar state machine signatures	No. of possible next states	First state	Predicting the second state
1	3	5	S1W1=15	Success
2	9	4	S1W1=23	Success
3	2	4	S1W1=64	Success
4	14	4	S1W1=70	Success
5	4	6	S1W1=29	Success
6	3	5	S1W1=2	failed
7	3	4	S1W1=153	Success
8	4	3	S1W1=24	failed
9	1	1	S1W1=14	Success
10	10	5	S1W1=131	Success

Table 7-8: Demonstrating the variations in the next state given the current state and the contents of the behaviour database

Table 7-7 shows the state variations for the first 4 stages in all the state machine signatures in our experiment's malware database, which the prediction model uses to select similar state machine signatures. This is useful as it demonstrates how our malware database has different states which are used to conduct the experiments. The first experiment is to validate how well the system predicts the next state by using ten malicious web page behaviours from the malware database, which has 110 malicious web pages grouped using the FP-Growth clustering algorithm. It shows that when passing the first state of each of the 10 experiment state machine signatures, it is able to predict 80% successfully while it has 20% error rate. The next experiment uses the same 10 state machine signatures as those shown in Table 7-8. We assume that each of the 10 experiment URLs has been loaded into the end-user's browser and moved into the sandbox as they are malicious. In each malicious web page, the prediction model will get the first state from the web-based attack and attempt to predict the next three states. It first estimates the next three states only, without the changes in parameters (File system, Process, Registry or Networks states), and in the second test it attempts to predict the

next three states with changes in parameters (source and destination of the change, based on similar state machine signatures in the malware database).

This is a real scenario in which the prediction model will be used on the end-user's system along with other models to build a malware database, as discussed previously in section 7.2.4. The experiment starts by feeding the first state to predict the second, and if it matches we predict the next two states in order to match them with the behaviour of the web-based attack, and then the attack group and potential malicious behaviour. The experimental scenario is demonstrated in Fig 7-12 below:



Fig 7-12: the prediction scenario for the second experiment.

We begin by passing the first state of the attack to the prediction model. Then, in each test the system attempts to predict the next three activities for the current web-based attack. In the experiment we conduct two tests: the first tests the possibilities of the next three states, such as file system or process, without being concerned with the source or destination.

The second tests the next three states, including the source and destination of the change. Experimental results are shown in Table 7-9, which shows that the model is able to predict 86% of the states for the next three states. This means that the model is able to understand the relationships between similar web-based attacks based only on the current one. Based on these similarities, it is then able to predict the possible next states. The tests also show that the model is correctly able to predict 60% of the next three activities, including the sources and destinations of the changes. The error rate of the two tests was 23%, and we assume that this decrease occurs because most attackers change their malicious actions slightly in order to avoid detection; for example, by changing the rootkit or key logger type for each attack after a number of attacks. We are therefore able to benefit from both tests in the prediction model, as predicting the state without the source and destination of the change can help in writing behaviour rules that work in the same way as IDS. For example, to write a detection rule for group ID 1-1 in Table 7-5, it has a label of S1W1=18, S1W1=19, S1W19=21, S3S19=20, and the rule is as presented in Listing 7-1. It shows the suggested behavior rule created from the group label for detecting when an ordered sequence of states is identified in the list of state machine signature. After the identification, the associated malicious web page is then blocked

and matched to its group. In the listing example 7-1, the elements {X,Y,Z,P,Q} are the changes operations (source and destination of the change) where {S1W,S3S} are the associated states.

Block M{S1WX = Y, S1WX = Z, S1WZ = P, S3SZ = Q, } → G{1 – 1};

Listing 7-1: an example of behaviour rule to detect malicious web page

Therefore, by applying these rules we can improve detection efficiency, even if the attacker changes the MD5 signature of the executable files or the process, registry names and values.

Web page ID	Next 3 states without change parameters	Next 3 states with change parameters
1	100%	100%
2	66%	66%
3	100%	100%
4	66%	33%
5	66%	33%
6	100%	66%
7	66%	66%
8	100%	33%
9	100%	66%
10	100%	66%
Total =	86%	63%

Table 7-9: Predicting the next three states experiment results

7.3 Hybrid system

Honeyware can be used to detect known malware and simulates operating systems or services (as explained in Section 7.1). On the other hand, a high-interaction client Honeypot like Capture-HPC is able to detect known and unknown attacks. Capture-HPC is the scan engine we have used in our state machine model to encode the behaviour collected from web-based attacks; it takes about 17 seconds to scan each URL whereas Honeyware takes an average of one minute because it downloads the web page first and then scans it using its scan engine.

To validate Honeyware, an experiment was conducted to compare its effectiveness as a low-interaction client Honeypot used to scan a collection of malicious and benign web pages against Capture-HPC [36]. The experimental scenario involved 94 web pages collected from public malware blacklists [116] [117], of which 84 were malicious and 10 benign. As previously mentioned,

The experiment's methodology involves scanning experimental URLs by using two scan methods: Honeyware and Capture-HPC. First, we passed each URL to Honeyware, which downloaded the URL's files and then scanned them, using signature-based detection. Honeyware uses anti-virus software to scan a web page after it has been downloaded and considers a web page to be malicious if one or more of its antivirus programs detect malicious codes or files in the page. Secondly, we passed each URL to Capture-HPC, scanning pages by using behaviour-based detection and checking if any malicious activity had occurred by analysing its log files manually. Capture-HPC loads a URL to the default browser, Internet Explorer version 6, and monitors the system's state and logs any activities for 30 seconds. Figures 7-13 and 7-14 show the experiment's results.

Honeyware detected 83 of the 84 malicious web pages and failed to detect 1 web page. After the antivirus software was updated, the test was conducted again on 8/11 and Avast Anti-Virus detected the web page it failed to detect the first time. On the other hand, we used Capture-HPC to scan the same 94 based on their behaviour. It was able to detect 62 of the malicious web pages, while incorrectly classifying 22 malicious web pages as benign.

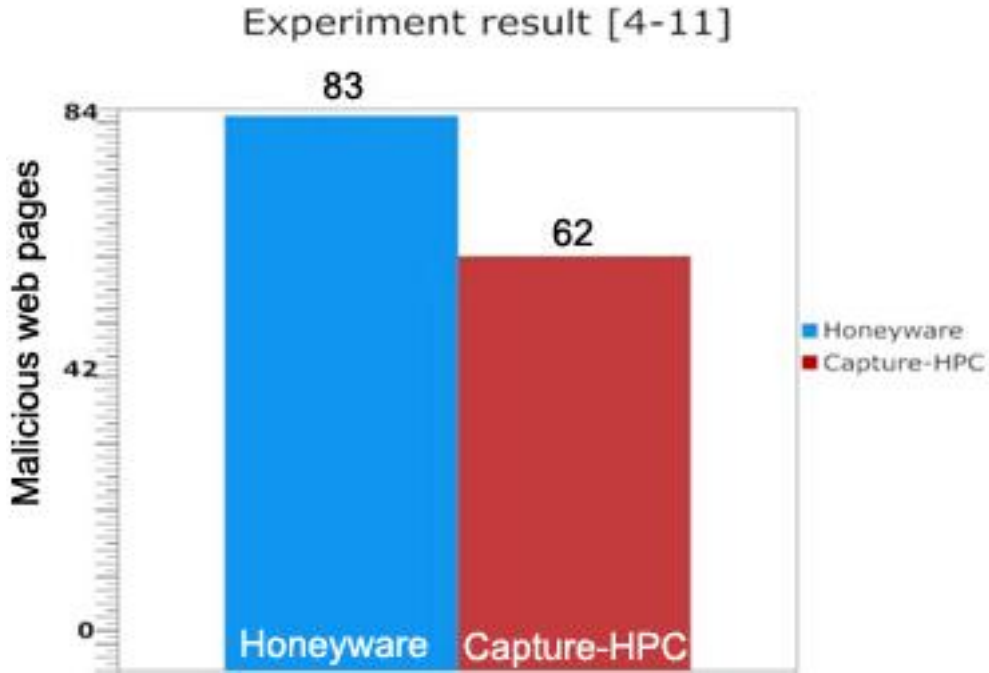


Fig 7-13: Experiment results on 4 November.

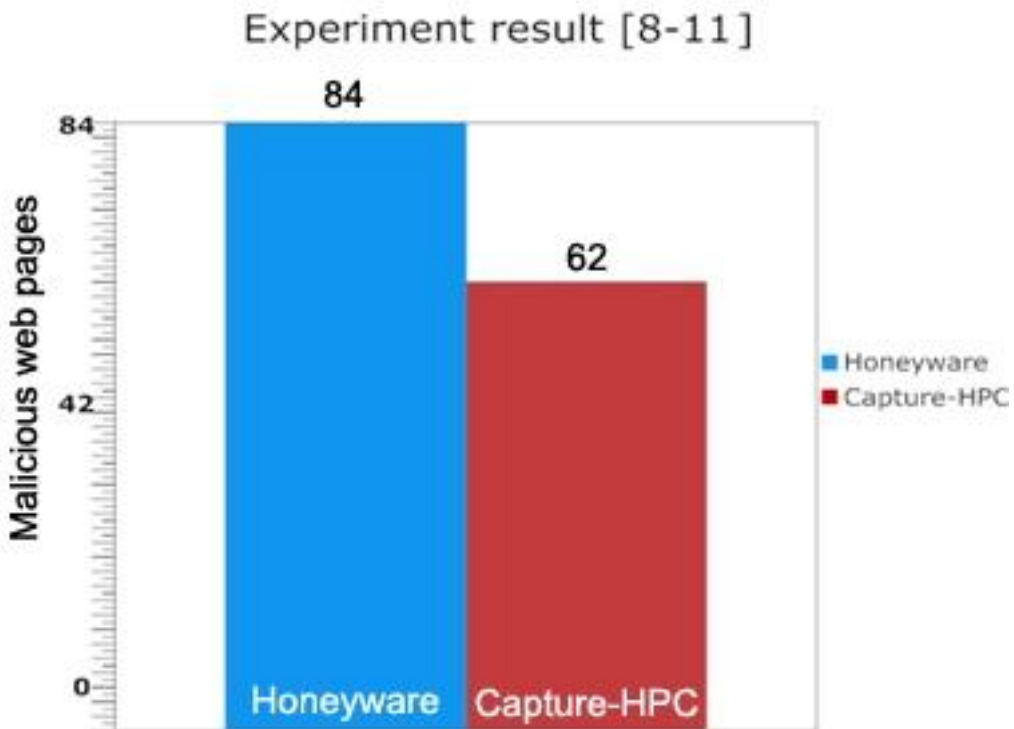


Fig 7-14: Experiment results on 8th November.

Honeyware is only able to detect known attacks as it uses antivirus software as the scan engine, and this needs to be updated regularly to achieve valid results. This was shown in this experiment when the Avast antivirus software later detected the new web page, which had not previously been detected. On the other hand, Capture-HPC failed to detect 22 malicious

web pages and classified them as benign. We believe that in Capture-HPC detection is triggered by visiting a web page, but not based on subsequent interaction with a user, to ensure the user was human and not a client Honey-pot. Other web pages try to fool the user into downloading a malicious application by pretending to be a useful benign application. Capture-HPC can only detect the behaviour of a web page scanned without any interaction, but Honeyware is able to download all the files and links that point to other sources, so it can download html pages, executable applications and other extensions that can be customised by the analyst. Fig 7-15 shows how well each antivirus program performed in the experiment.

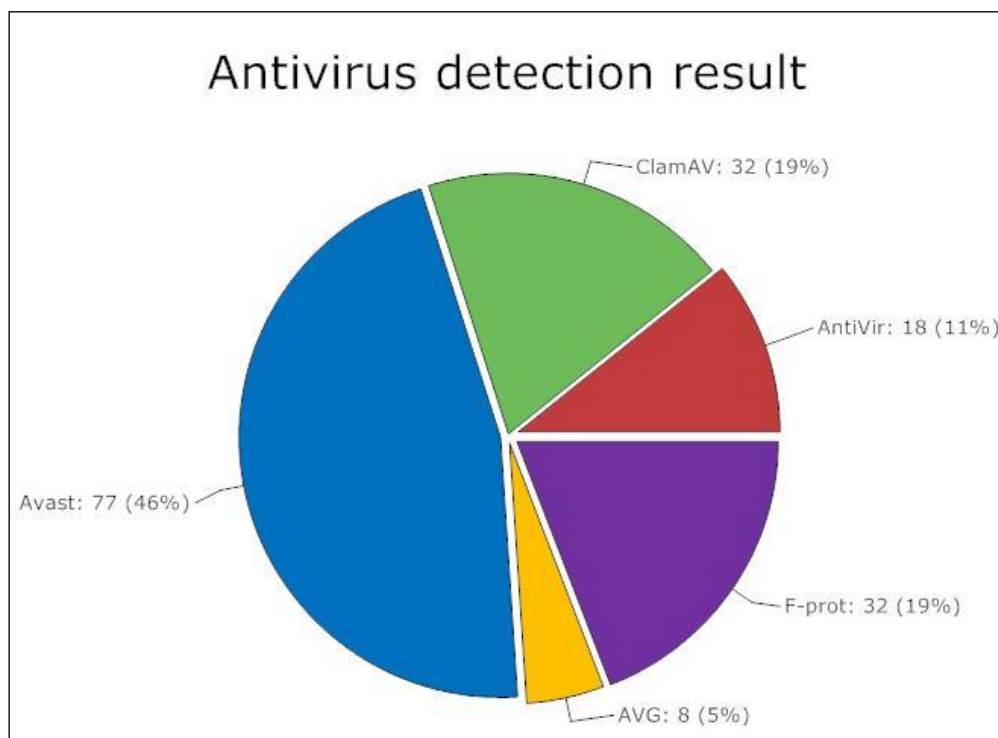


Fig 7-15: the antivirus detection results.

The experiment showed that Honeyware was able to detect all the malicious web pages, but took a long time to finish scanning all the pages and returning the results to the investigator, while Capture-HPC scanned all 94 web pages in about 30 minutes. The idea of a hybrid system is that it should combine high and low-interaction client Honey-pots. The benefit of this hybrid system is that Capture-HPC is fast enough to scan web pages and is able to detect unknown malicious websites, while Honeyware can detect web pages that need some interaction from an end-user and so cannot be detected by Capture-HPC. The hybrid system starts by scanning all web pages with Capture-HPC and then forwarding all web pages classified as benign by Capture-HPC to Honeyware to be scanned. This hybrid system would be fast and would detect all our experimental web pages, as Capture-HPC detects the 62 web

pages quickly and then forwards the benign and false-negative web pages, of which there were 32, to Honeyware. Fig 7-16 shows the idea behind the hybrid system.

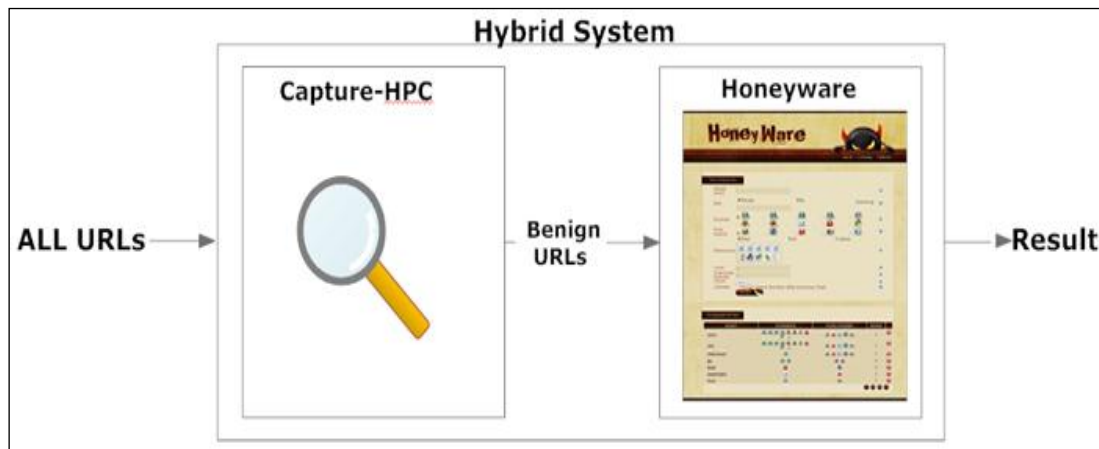


Fig 7-16: the hybrid system.

7.4 Summary and Discussion

In this chapter we have shown a validation of the web-based behaviour system by conducting experiments to show how the three main models of the system function; state machine, clustering and prediction. The evaluation experiments show how the malware database, which is the basis of the prediction model, is built by scanning known web-based attacks and then generating the state machine signature for each web-based attack behaviour. The clustering model is a novel grouping mechanism used to cluster malware based on common behaviour. This has a number of advantages, as mentioned in chapters 2 and 3, such as the ability to group web-based attack behaviours even if they use obfuscation such as changing the order of the attack sequence and steps. Our experiments show how the prediction model works against real web-based attacks.

The experiment results show that we were able to detect malicious Web pages by predicting their behaviour as illustrated in Section 7.2.4. The first experiment was conducted to test the capability of predicting the second state when the first state is given. It shows that we were able to achieve 80% accuracy when predicting the second state of each state machine signature. The second experiment was conducted to simulate the proposed scenario on the end-user system, in which the user visited a Web page, and our system detected malicious behaviour from the visited server. The malicious Web page was moved to a sandbox, obtaining the first malicious behaviour, and then predicting the next three states to validate

this. We were able to achieve 63% accuracy when predicting the next three states without a change in operations (source and destination of the change). Further, we were able to achieve 55% accuracy when detecting the next three states in predicting the states with a change in operations.

We learned from our experiment that we were able to detect and predict web-based attacks based on its behaviour, detecting both known and unknown attacks, unlike signature-based detection, which can only detect known and non-encrypted attacks. Examples are blocking the interaction with malicious Web pages when detecting their malicious behaviour to predict the user system and learning what methods the target server uses to affect the user system even if it did not complete its attack by predicting the possible actions from the attacks. We also learned that the use of the common state clustering approach is helpful in detecting similar Web-based attacks as it attempts to determine the similarity in different parts of its state machine signature. However, one of the challenges in the clustering approach is the order of states and how a change in the order of states can be dealt with, which in turn preforms the same action but in a different order. One of these solutions is to improve the clustering model to understand these changes and to obtain the common states that preform a particular action even if the order is changed. This can be a limitation of the monitoring software or an attacker intending to fool the clustering model.

We note that because we used a high-interaction client Honeypot as the basis for generating the state machine model for the behaviour of Web-based attacks, it might not detect all kinds of Web-based attacks. Therefore, we propose the use of Honeyware, a low-interaction client Honeypot, alongside a high-interaction Honeypot in order improve accuracy. We found that by using signature-based detection (Honeyware) alongside behaviour detection (Capture-HPC), we can provide rich data on the Web-based attacks. We can conduct a behaviour scan of the Web page as well as obtain MD5 signatures and malware types by using Honeyware and the associated anti-virus system it make use of.

Chapter 8

Conclusion and future work

The internet is an ideal place for many different kinds of people to pursue their various interests. However, there is one group of people who use the internet to attack end-users and the originations. The goals and behaviour patterns of these people ('attackers') can be categorised into two groups. The first group aims to assault as many computers as possible by using attack tools and scripts. These people tend not to have high IT skills and therefore attack any computer which could belong to end-users or an organisation, rather than targeting specific computers. The second group comprises of highly-skilled attackers whose main goal is to attack particular computers of high value to them. The second group's motivations tend to be political, religious or financial. Both these groups are equally dangerous because they each pose a serious threat to our computers. Web-based attacks can be detected by a number of methods, including blacklists, signature or behaviour detection. Most web security protection approaches are based on signature detection. This type of detection is only able to detect known web-based attacks which do not change their malicious code or use obfuscation. However, behaviour detection is able to detect known and unknown malicious web pages and is more accurate as it detects web-based attacks even if they change their methods or codes. Behaviour detection involves understanding how web based attacks work; therefore, we propose a state machine to model the behaviour of web-based attacks using a predefined structure that can then be used for analysis.

This thesis proposes a web-based behaviour system that is able to analyse, learn and predict web-based attacks by combining three main models: state machine, clustering and prediction. Our proposed system is able to understand web-based attacks and detect known and unknown attacks based on a malware database which is built using behaviour groups. Our system is

intended to protect end user systems from the risk of drive-by download attacks that use the web browser. The rest of this chapter summarises the key contributions presented in this thesis and possible future work we believe would be valuable to improve the analysis and detection of web based attack behaviors.

8.1 Evaluating research hypothesis

In Section 1.3 the research hypothesis for this thesis was presented. The core part of the hypothesis is reiterated below as a reminder for the reader:

The hypothesis of this research is that monitoring behaviour is a more efficient, safer and accurate approach to detecting web-based malware than a signature-based approach.

The research conducted in this thesis and the approach documented in Chapters 4, 5 and 6 tested this hypothesis to the point where it is possible to say that it does indeed hold true. The three models used in this research have been developed together in order to build a web-based behaviour system which can encode, analyze and detect malicious web pages based on their behaviour. Firstly, we based our analysis on the behaviour of malicious web pages and we used Capture-HPC as the basis of our scanning engine as previously discussed in detail in Chapter 4. The high-interaction client Honeypot provided us with the relevant data regarding behaviour of the running malicious web pages in real time in order to encode it using a state machine model. Our state machine model encoded malicious web page behaviour and it provided the analytical base for other complex algorithms and processes in order to derive more relationships, such as clustering and prediction algorithms, from the database of the state machine signatures. Secondly, in order to detect malicious web pages based on their behaviour, we introduced a malware database which contained the previously scanned web-based attacks categorised into both main and sub-behaviour groups in order to aid our system in detecting malicious web pages. Therefore, we developed a clustering model based on the FP-Growth algorithm that was able to find similarities between state machine signatures based on their common states. Finally, we developed a prediction model to detect malicious web pages based on behaviour as described in detail in Section 6.1.1. As shown from the experimented results in Section 7.2.4, we were able to detect malicious web pages through two means: through a match to a specific web-based attack in a group; or by creating a different behaviour group by using the clustering models discussed in chapters 5 and 6.

8.2 Answers to the Research Questions and Problems

In this Section the four research questions identified in Section 1.4 will be discussed in relation to the research undertaken in this thesis in Sections 8.2.1 – 8.2.3. In each Section, the research question will be repeated and the relevant models which have been implemented will be discussed. In addition, the analysis performed to demonstrate the added capabilities will also be discussed, along with the new knowledge that has consequently been delivered.

8.2.1 Behaviour states

Question 1: How does the security analyst, network administrator and end-user interpret the behaviour of a malicious web page?

Malicious websites may cause a number of activities to be performed on the victim's system; each activity is performed in different stages. These changes are performed in order to damage and change the victim's system in certain way. We use our monitoring engine to get the behaviour of the malicious web pages. The Honeypot will monitor the machine state and log all the malicious behaviour and send it to an analysis server. There are a number of modeling methodologies that assist representing the structure and methods of web-based attacks. These models help to understand the possibility of the threats, as mentioned in chapter 2, there are different modeling methods such as attack trees, Petri nets and UML. Most of them can help to present and model complex attacks that involve modeling different paths and scenarios. In our work, we need to model web-based attacks, in which a malicious web page attacks the end-user's system through a web browser in stages, and each stage causes a transition from one stage to another. The malicious behaviour steps should be ordered over time, in order to understand how it works and which state is responsible for causing the change to the next activity. Therefore, we needed to choose an attack graph, as discussed in section 2.2 in chapter 2 to enable us to visualise the steps in each attack scenario.

The steps in the malicious behaviour should be ordered over time. We chose an attack graph to enable us to visualise the steps in each web-based attack scenario. We based our model on a finite state machine, as this provides a simple way to represent and save the behaviour data in the states, starting with the first state, where the attack begins and from that point tracing the event-driven process; any change should trigger an event within the current state and cause it to transfer to the next state, which represents another malicious activity. The model

was based on the (FSM) because the parts of the system that needed to be monitored were limited. The state machine signature allows security researchers and automatic tools to understand and follow the steps of the web-based attacks.

Finally, the experiment in section 4.2.4 shows how we can get the web-based malicious behavior from the Honeypot. It shows that we were able, by using Weka data mining toolkit, to extract relationships from the state machine signatures that can assist the security analyst, network administrator and end-user to understand what the web-based attacks performs.

8.2.2 Grouping web-based attacks

Question 2: How can the relationships between different malicious web pages be analysed and displayed, based on their behaviour within the end-user's system?

We use a state machine, as discussed in the previous chapter, to represent the activities performed by the malicious web page into pre-defined states. These states can be used to summarise interactions with malicious web pages using the same state machine structure. The clustering model was introduced to group similar web-based attacks based on their behaviour. These behaviour groups showed that similar behaviour has been used in web-based attacks. This will help security companies and researchers to study the different tricks used by the same exploit and develop a security patch or detection mechanism against this type of attack.

In Chapters 2 and 5 we mentioned the other approaches used to group attacks in general and we have selected the most familiar approach, Hierarchical Agglomerative Clustering (HAC), which has been used widely to cluster attacks and exploit security vulnerabilities. However, some limitations exist within the HAC clustering approach, such as: (a) each state machine has to be of the same length, requiring the insertion of null states; (b) the web-based behaviour groups produced by the clustering algorithm do not provide a label for each group; (c) to add additional behaviour groups, it is necessary to re-cluster the entire data set, which is a time-consuming process. Therefore, we use the FP-Growth clustering approach to overcome the previous limitations. The use of FP-Growth has been discussed in Section 5.2.2 and showed that we were able to acquire the behaviour groups, as well as sub-groups, from the state machine signatures.

There are two main benefits of our clustering model, the model is able to mark each behavior group with a label and can deal with arbitrary length state sequence. The label represents

frequent states within each group. By using this label, we can identify similarities between groups and then use the label to match with new data. Each sub-group will have a label to identify the common states within each group which indicates why each sub-group state machine is similar. Using this clustering approach we can add new behaviour groups to the already discovered groups. The second benefit is that we are able to output two behaviour group types: main and sub-groups. The main group contains similar state machine signatures that share the same sequence of states. This is helpful for grouping similar web-based attacks together, such as grouping attacks that download executable files and change specific registry values or attacking the web browser using a specific exploit. However, within each main group some of the web-based behaviour attacks might share other similarities. The sub-group is responsible for further refining each group. This stage can provide more information within each group, such as finding additional similarities in each behaviour group, or showing that these sub-groups can help better characterise each piece of recorded behaviour.

Finally, we have evaluated our clustering model by comparing it to the most widely used clustering approach HAC in Section 5.2.5, we have used the same data set with the two clustering approaches. The experiment shows that the HAC algorithm was only able to identify 77 groups. On the other hand, the FP-growth algorithm had 27 behavior groups and the largest group had 34 state machines. Hence, FP-growth was able to find additional similarities between the state machines. In terms of security, FP-Growth, as the basis of our clustering model, identifies the behaviour sequence responsible for exploiting vulnerability in a victim's machine, analyze this behaviour to learn its technique and methods, and then develops a path for a 0-day exploit. It is also able to trace common attack behaviours in different web-based attacks and identify them and learning how a group of web-based attacks use the same exploit in different formats.

8.2.3 Predicting malicious behaviour

Question 3: How can the future behaviour of a running web page be predicted based on its past behaviour?

The prediction algorithm will scan the current web page and then attempt to predict future behaviour to determine if it is malicious or benign. The prediction algorithm uses knowledge from previously-seen web-based malware behaviour to understand how they operate and which types of malicious web pages are related to which behaviour group. The current web

page is monitored using high interaction client Honeypot to get its behaviour in real-time and send them to the analysis server. On the server side, the state machine firstly will encode the behaviour to state machine signature and then pass this to a prediction model to calculate the potential future malicious behaviour based on the malware database.

This model uses a state machine to encode malicious behaviour as discussed in chapter 4. Hidden Markov Model (HMM) is used Baum-Welch learning algorithm that computes the forward and backward probability matrices for a given observation referenced by its index in the input training data, as described in details in chapter 6.

8.3 Contributions and novelty

In this section we will discuss our contributions to the security field and how we have built and developed our models in order to achieve the research aims and objectives.

8.3.1 Common behaviour clustering approach

As discussed in chapter 2, there are different approaches to clustering and grouping similar malware automatically. A number of studies have used the HAC clustering approach [49] [53] to group similar malware while others use algorithms such as K-means [51]. We propose a novel clustering approach, based on FP-Growth, which is able to cluster the behaviour of web-based attacks based on the common behaviour between different attacks. The proposed clustering model is able to group web-based attacks into behaviour groups based on state machine signatures, and can also form sub-groups which include attacks that similar. The model produces a label for each behaviour group which shows the common behaviours held in that group. The label can be used to identify similarities between attacks types as well as to merge new web attacks based on their label. The experiment and evaluation results show that FP-Growth is more effective when compared to other approaches such as HAC, as shown in Chapters 5 and 7.

8.3.2 Predicting malicious behaviour

There are a number of different approaches and methods to detect malware either dynamically or statically, such as anomaly or signature-based detection. Our proposed detection model is based on two main principles: an anomaly behaviour detection system and

a prediction algorithm. The web-based behaviour system uses a high-interaction client Honeypot, Capture-HPC, which defines the benign behaviours of systems and then categorises these as normal behaviour. Once it detects any abnormal behaviour then it encodes these into a state machine model and uses our prediction model to detect the malware's future behaviour. Furthermore, the model predicts the possible future behaviours of the current web-based attack.

The prediction model bases its process on the malware database built using the clustering model. The malware database includes the behaviour of web-based attacks grouped into main and sub behaviour groups. The prediction model works by finding attacks with similar behaviour to the current stage of the web based interaction. Moreover, it calculates the possibilities of future activities for current web-based attacks based on similar state machine signatures. In the case of a successful match with predicted future behaviour, the model blocks interaction with the web-based attack and returns a result to the system, which includes the type of web-based attack and its possible future malicious activities. It also saves the new state machine signature to the malware database to speed up the prediction process in the future. Experimental results in the previous chapter show that we were able to detect 70% of the future activity for 10 real web-based attacks. This shows that the prediction model helps to protect users from malicious web pages using drive-by download attacks. The web browser is the gateway for this type of attack and it is increasing every year as many applications move to the cloud, meaning users are using web browsers to access these services and other websites.

8.4 Future Work

In this section we describe work not yet conducted, but one that would be a valuable contribution to this research in the future.

8.4.1 Improvement of the scan engine

The current scan engine we are using, Capture-HPC, is only able to monitor and detect changes to Processes, the File system, the Registry and the Network. However, attacks can also target the memory such as buffer overflow. We therefore suggest adding a memory

detection mechanism as this would improve the ability to detect even more web-based attacks.

The state machine signature could be easily adapted to include memory in the model, alongside the other four types. The model is able to create a state for memory changes and define the change types, such as read and overwrite. We believe that by adding a memory detection component to the state machine model, we would be able to discover more web-based attacks and understand how changes in memory could affect changes to other parts of the system.

8.4.2 Clustering obfuscation

Malware are using obfuscation and encryption to avoid detection and hide their malicious behaviour from security and analysis tools. We faced the problem of obfuscation at the beginning of our clustering research, as attackers inserted unwanted behaviours before or after the malicious behaviour. This can easily lead to similar web-based attacks being grouped into different behaviour groups as most clustering approaches attempt to find similarities between the behaviours of web-based attacks from the beginning of the behaviour stream. Therefore, we propose a clustering model that is able to find the main malicious behaviour sequences in any part of a behaviour sequence of a particular web-based attack. This can group similar web-based attacks even if they use obfuscation methods by inserting unwanted behaviours or changing the position of the main malicious behaviour in its behaviour sequence.

While our proposed clustering model is able to overcome the obfuscation technique mentioned previously, there are other techniques used to avoid detection and analysis, we suggest that they also need to be explored and solved to build a more accurate grouping model. For example, the attacker could change the order of some behaviours in order to avoid clustering and analysis to find its group and family. We suggest that we use the impact of the behaviour sequence where it calculates the effect and results of a given behaviour sequence, even if the order of states are different from one web based attack to another.

8.4.3 Client-server model

Currently in our web-based behaviour system, the prediction model runs within the end user and testing systems in order to predict and detect web-based attacks. This is a good solution for users; however, it suffers from a number of limitations, such as the small number of state

machine signatures in the malware database. As the users would need to update the malware database in order to speed up the process of detection, and the system might be slow depending on the user's system capabilities. These and other disadvantages might discourage users from using our system; therefore, we suggest the use of a client-server model for our system. The basic idea behind this is that the server would sit in the cloud and have one central malware database that is dynamically updated by all the users in many different places.

The prediction model could also be deployed on the server, but we suggest that we would install a lightweight monitoring system on the user's system which would be able to detect changes in the system state from the web browser and send these behaviours to the cloud in real time to detect and predict any malicious behaviour. By using this methodology, we could remotely predict and detect malicious web pages, thereby providing features such as improving speed of detection, a central database of known malicious state machine signatures and the ability to use the prediction model on different platforms, such as PCs or mobiles.

8.5 Web-based Malware Framework

The web-based malware framework aims to detect, analyze and predict malicious activities from web-based malware. Its objective is to protect the end-user's machine from attacks on the client application based on dynamic analysis, which has a number of benefits, such as detecting the encrypted and obfuscation malware. Our framework has a number of advantages as follows:

1. Ability to monitor any malicious web pages and model malicious actions in real-time.
2. Analyze different malicious websites and group them into behaviour groups by finding the similarities between them based on the common states approach as discussed in Chapter 5.
3. Protect the end-user's system and prevent malicious web pages from attacking the system by predicting and blocking malicious behaviour before it can harm the end-user's system.

On the other hand, our system suffers from the following disadvantages:

1. It is able to detect the malicious main actions in different parts of the system while grouping them, but it cannot detect them if the states have exchanged their order. Instead, it performs the same malicious actions.
2. It is only able to protect the system from client-side attacks, and this includes drive-by download attacks. However, it cannot detect other web attacks, such as SQL injections or XSS.

8.6 Lessons Learned

This section describes the lessons learned during research on the Honeypot, web security and data mining. To be precise, we describe the problems and challenges we faced as well as how we overcame and solved them.

Malicious websites: an attacker usually changes their malicious web page's URL in order to avoid detection by security applications, which would add the malicious site to a blacklist. Therefore, we found that it is difficult to find working malicious web pages as most of the public ones are down or inactive. We learned that in order to test our framework on the malicious web pages, we should test it directly and collect the data without waiting to test large numbers as the sites frequently change their URLs and their behaviour from malicious to benign in order to avoid detection.

Monitor the malicious website: this is different from scanning a single malware, such as a worm. As a malicious website may affect different parts of a system, we need to monitor the whole system in order to collect the actions and changes to the system. We have used the client Honeypot in order to perform system monitoring. Still, we also need to use other detection tools, such as monitoring the memory and detecting attacks, such as buffer overflow.

Grouping similar attacks: clustering algorithms and approaches helps to find any similarities between sequences and groups them into similar families. In order to group similar malicious websites, we compared the malicious activities for each web-based malware to find any similarities between them. We discovered that there are a number of obfuscation techniques used by attackers to avoid detection and link attacks to their family. Therefore, we had to extensively learn these methods. In doing so, we found that most of the attacks move the order of important malware actions to different parts of the attack and insert

unwanted actions to avoid detection. We were able to detect these methods by using the common states clustering approach as described in Chapter 5.

8.7 Concluding Remarks

The thesis has researched and analysed web-based attacks based on behaviour. We have proposed a number of approaches to achieve our hypothesis, which focuses on detecting malicious web pages based on behaviour, and not signature, to protect the user from web-based attacks. We learned from the experiments in Section 7.2 that by using behaviour detection, we were able to achieve between 86% to 63% accuracy. These values show that we can detect known and unknown malicious Web pages by using a behaviour-based strategy, unlike the use of a signature-based strategy in which the attacker can avoid detection by using encryption but causing the same malicious behaviour without detection as illustrated in Section 7.2.3. We have based our proposed system on the behaviour scanning approach using a high-interaction client Honeypot. We believe that our system can improve this technology by adding the capability to detect additional attacks, saving scanning time and improving the performance in capturing attacks.

Bibliography

1. Ostrowick, J., "*How the Web Works - An Introduction*", Publisher; Lulu, 7th edition, 2009, ISBN = 978-1409256519.
2. Cheswick, W.R., Bellovin, S.M., and Rubin, A.D., 2nd edition, "*Firewalls and Internet Security: Repelling the Wily Hacker*", Addison Wesley professional, 2003, ISBN = 978-0201634662.
3. Gordon, L.A., Loeb, M.P., Lucyshyn, W., and Richardso, R., "*CSI/FPI COMPUTER CRIME AND SECURITY SURVEY*", Computer Security Institute (CSI), 2006, [Online]. Available at: <http://pdf.textfiles.com/security/fbi2006.pdf> last accessed 1 Dec. 2011.
4. Schneier, B., "*Honeypots and the HoneyNet Project*", 2001, [Online]. Available at: [\[http://www.cs.rochester.edu/~brown/Crypto/news/3.txt\]](http://www.cs.rochester.edu/~brown/Crypto/news/3.txt), last accessed 1 Dec. 2011.
5. Spitzner, L., *Honeypots: Tracking Hackers*. 2002, 1st edition. Addison-Wesley Professional. ISBN-10: 0321108957.
6. Jain, Y.K., and Singh, S., "Honeypot based Secure Network System", International Journal on Computer Science and Engineering (IJCSSE), 2011, Vol. 3, no. 2, pp. 612-620. Engg Journals Publications.
7. McGraw, G., and Morrisett, G., "*Attacking malicious code: a report to the Infosec Research Council*," Software, IEEE , vol.17, no.5, pp.33-41, Sep/Oct 2000, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=877857&isnumber=19003>
8. Cohen, F., "*A cryptographic checksum for integrity protection*", publisher: Lehigh University, USA, Computers and Security Volume 6, Issue 6, 1987. pp., 505-510.
9. Cracker, S., and Pozzo, M.M., "*A proposal for a verification-based virus filter*", 1989, publisher: Oakland, CA , USA , Proceedings 1989 IEEE Symposium on Security and Privacy , May 1989. pp. 319-324.
10. Lo, R., Levitt, K., Olsson, R., *MCF: a malicious code filter*, *Computers & Security*, publisher: Department of Computer Science, University of California, USA, Volume 14, Issue 6, 1995, pp. 541-566, ISSN 0167-4048.

11. Christodorescu, M. and Jha, S., *Static Analysis of Executables to Detect Malicious Patterns*. 2003. In Proceedings of the 12th conference on USENIX Security Symposium. Vo.12. USENIX Association, pp. 169-186. Available at: <http://portal.acm.org.proxy.queensu.ca/citation.cfm?id=1251353.1251365#>.
12. Christodorescu, M., Jha, S., Seshia, S.A., Song, D., Bryant, R.E., 2005, "*Semantics-aware malware detection*," Security and Privacy, 2005 IEEE Symposium on, pp. 32-46.
13. DataRescue. IDA Pro Disassembler. 1995, 2nd edition, Available at [<http://www.datarescue.com/idabase>]. Last accessed 3 Dec. 2011.
14. Preda, M. D., Christodorescu, M., Jha, S., and Debray, S., 2007, *A semantics-based approach to malware detection*. In Proceedings of 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2007.
15. Preda, M., Christodorescu, M., Jha, S., and Debray, S., 2008, *A semantics-based approach to malware detection*. *ACM Trans. Program. Lang. Syst.*, 30(5), 2008.
16. Schultz, M., Eskin, E., Zadok, F., Stolfo, S., "*Data mining methods for detection of new malicious executable*," 2001, Security and Privacy, S&P. Proceedings. 2001 IEEE Symposium on, pp.38-49, 2001
17. Muazzam Ahmed Siddiqui, "*Data Mining Methods for Malware Detection*," .2008, University of Central Florida, Tech. Rep., PhD Dissertation.
18. Siddiqui, M., Wang, M., and Lee, J., 2008. *A survey of data mining techniques for malware detection using file features*. In Proceedings of the 46th Annual Southeast Regional Conference on (ACM-SE 46). ACM, New York, NY, USA, 509-510.
19. Tsai, C., Tseng, C., Han, C., "*Intrusive behavior analysis based on honey pot tracking and ant algorithm analysis*," 2009. Security Technology, 43rd Annual 2009 International Carnahan Conference on , pp.248-252.
20. Visumathi, J., and Shunmuganathan, K., 2011, "*A computational intelligence for evaluation of intrusion detection system* ", Indian Journal of Science and Technology. Vol. 4, no. 1, 40 p.
21. Radosavac, S., Seamon, K., Baras, J., 2005. "*Short Paper: bufSTAT - a tool for early detection and classification of buffer overflow attacks*," Department of Electrical and Computer Engineering and the Institute for Systems research University of Maryland, College Security and Privacy for Emerging Areas in Communications Networks. SecureComm 2005. First International Conference on, pp. 231- 233, 05-09 Sept. 2005
22. Wang, J., Deng, P., Fan, Y., Jaw, L., Liu, Y., "*Virus detection using data mining techniques*" 2003. Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on, pp. 71- 76.

23. Wu, R., Li, W., Huang, H., "An Attack Modeling Based on Hierarchical Colored Petri Nets," 2008. Computer and Electrical Engineering, ICCEE 2008. International Conference on, pp.918-921.
24. Chen, S., Kalbarczyk, Z., Xu, J., Iyer, R., "A data-driven finite state machine model for analyzing security vulnerabilities,". 2003, Dependable Systems and Networks, Proceedings. 2003 International Conference on, pp. 605- 614.
25. Shafiq, M., Khayam, S., Farooq, M., Zamboni, D., "Embedded Malware Detection Using Markov n -Grams". Detection of Intrusions and Malware, and Vulnerability Assessment, V. 5137, pp.88-107, Springer Berlin / Heidelberg, 2008, ISBN: 978-3-540-70541-3. Available at: http://dx.doi.org/10.1007/978-3-540-70542-0_5
26. Chen, T. M., Sanchez-Aarnoutse, J. C., Buford, J., "Petri Net Modeling of Cyber-Physical Attacks on Smart Grid," 2011. Smart Grid, IEEE Transactions on , vol.2, no.4, pp.741-749. ISSN: 1949-3053.
27. Linn, C., and Debray, S.K., "Obfuscation of executable code to improve resistance to static disassembly", 2003. in Proc. ACM Conference on Computer and Communications Security, pp.290-299.
28. Popov, I., Debray, S., and Andrews, G., 2007. " Binary obfuscation using signals". In Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium (SS'07), Niels Provos (Ed.). USENIX Association, Berkeley, CA, USA, Article 19, 16 pages.
29. Ferri, P., "ANTI-UNPACKER TRICKS – PART ONE", *VIRUS BULLETIN*, 2008, [Online]. Available at: [<http://pferrie.tripod.com/papers/unpackers21.pdf>] Last accessed; 11 Dec 2011.
30. Szor, P., *The Art of Computer Virus Research and Defense*. Addison Wesley, Reading (2005)
31. Yetiser, T., *Polymorphic Viruses – Implementation, detection, and protection*, 1993. [Online]. Available at: [<http://vx.netlux.org/lib/ayt01.html>] Last accessed; 10 Dec 2011.
32. Rashid, F., "Cloud Security Services Can Reduce Malware", *eWeek* , [Online], Available at: [<http://www.eweek.com/c/a/Security/Cloud-Security-Services-Can-Reduce-Malware-329353/>] Last accessed; 12 Dec 2011.
33. Willems, C., Holz, T., Freiling, F., "Toward Automated Dynamic Malware Analysis Using CWSandbox," 2007, Security & Privacy, IEEE, vol.5, no.2, pp.32-39, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4140988&isnumber=4140976>.

34. Norman ASA. "SandBox Information Center", [Online]. Available at: [http://www.norman.com/security_center/security_tools/] Last accessed; 14 Dec 2011.
35. ThreatExpert, [Online]. Available at: [<http://www.threatexpert.com>] Last accessed; 16 Dec 2011.
36. Capture-HPC, 2008. [Online] Available at: [<https://projects.honeynet.org/capture-hpc>] Last accessed; 1 February 2009]
37. Anubis, "*Anubis: Analyzing Unknown Binaries*", 2011, [Online]. Available at: [<http://anubis.iseclab.org>] Last accessed; 3 February 2009.
38. Moser, A., Kruegel, C., Kirda, E. , "*Exploring Multiple Execution Paths for Malware Analysis*," Security and Privacy, 2007. SP '07. IEEE Symposium on , vol., no., pp.231-245, 20-23 May 2007
39. Talha, "*Detecting Virtualization*", 2006, [Online]. Available at: [<http://talhatariq.wordpress.com/2006/05/14/detecting-virtualization-2/>] Last accessed; 6 February 2009.
40. Klein, T., "*ScoopyNG - The VMware detection tool*", [Online]. Available at: [<http://www.trapkit.de/research/vmm/scoopyng/index.html>] Last accessed; 8 February 2009.
41. Skoudis, E., and Liston, T., *On the cutting edge: Thwarting virtual machine detection. Presentation at SANS@Night, 2006.*
42. Jaelani, "*Just Another VM Detection (was "VM Detection Combo")*", 2009. [Online]. Available at: [<http://my.opera.com/jaelanicu/blog/just-another-vm-detection-was-vm-detection-combo>] Last accessed; 9 February 2009.
43. Li, S., Ebringer, T., and Boztag, S., "*An automatic anti-anti-VMware technique applicable for multi-stage packed malware*," Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on, pp.17-23, 7-8 Oct. 2008, URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4690853&isnumber=4690850>
44. Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., and Nazario, J. *Automated classification and analysis of internet malware.* 2007. In Proceedings of the 10th international conference on Recent advances in intrusion detection

- (RAID'07), Christopher Kruegel, Richard Lippmann, and Andrew Clark (Eds.). Springer-Verlag, Berlin, Heidelberg, 178-197.
45. VMware, Inc. [Online]. Available at: [<http://www.vmware.com>] Last accessed;11 February 2009.
 46. King, S., and Chen, P., 2003. *Backtracking intrusions*. SIGOPS Oper. Syst. Rev. 37, 5 (October 2003), pp. 223-236.
 47. Rieck, K., Trinius, P., Willems, C., and Holz, T., Automatic Analysis of Malware Behavior using Machine Learning. Technical Report 18–2009, Berlin Institute of Technology, December 2009
 48. Bailey, M., Oberheide, J., Andersen, J., Mao, Z., Jahanian, F., and Nazario, J., *Automated classification and analysis of internet malware*. 2007. In Proceedings of the 10th international conference on Recent advances in intrusion detection (RAID'07), Christopher Kruegel, Richard Lippmann, and Andrew Clark (Eds.). Springer-Verlag, Berlin, Heidelberg, 178-197.
 49. Lee, T., Mody, J.J.: *Behavioral classification*. In: Proceedings of EICAR 2006 (April 2006).
 50. Perdisci, R., Lee, W., and Feamster, N., *Behavioral clustering of HTTP-based malware and signature generation using malicious network traces*. 2010. In Proceedings of the 7th USENIX conference on Networked systems design and implementation (NSDI'10). USENIX Association, Berkeley, CA, USA, pp. 26-26.
 51. Christian, R., Lim, C., Nugroho, A.S., Kisworo, M., *"Integrating Dynamic Analysis Using Clustering Techniques for local Malware in Indonesia,"* Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on , vol., no., pp.167-169, 2-3 Dec. 2010
 52. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., *"The WEKA Data Mining Software: An Update"*, SIGKDD Explorations, volume = 11, pp.10-18, 2009, [online]. Available at:[<http://www.kdd.org/explorations/issues/11-1-2009-07/p2V11n1.pdf>] Last accessed; 22 February 2009.
 53. Gurrutxaga, I., Arbelaitz, O., Ma Perez, J., Muguerza, J., Martin, J.I. and Perona, I. *Evaluation of Malware clustering based on its dynamic behaviour*. (2008). In Proc. Seventh Australasian Data Mining Conference (AusDM 2008), Glenelg, South Australia. CRPIT, 87. Roddick, J. F., Li, J., Christen, P. and Kennedy, P. J., Eds. ACS. 163-170.
 54. Wang, X., Yu, W., Champion, A., Fu, X., Xuan, D., *"Detecting worms via mining dynamic program execution,"* Security and Privacy in Communications Networks and the Workshops, 2007. Third International Conference on SecureComm 2007., pp.412-421, 17-21 Sept. 2007

55. Stopel, D., Boger, Z., Moskovitch, R., Shahar, Y., Elovici, Y. , "*Application of Artificial Neural Networks Techniques to Computer Worm Detection*," Neural Networks, 2006. IJCNN '06. International Joint Conference on , pp.2362-2369.
56. Linda, O., Vollmer, T., Manic, M., "*Neural Network based Intrusion Detection System for critical infrastructures*," Neural Networks, 2009. International Joint Conference on IJCNN 2009., pp.1827-1834, 14-19 June 2009
57. Sequeira, K., and Zaki, M., *ADMIT: anomaly-based data mining for intrusions*. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '02). 2002. ACM, New York, NY, USA, 386-395. DOI=10.1145/775047.775103 <http://doi.acm.org/10.1145/775047.775103>
58. Schneier, B., *Attack Trees*. Doctor Dobb's Journal, pp. 21–29, December 1999. Available at: [<http://www.schneier.com/paper-attacktrees-ddj-ft.html>] pdf [Accessed 1 Feb 2012]
59. Wu, R., Li, W., Huang, H., "*An Attack Modeling Based on Hierarchical Colored Petri Nets*," Computer and Electrical Engineering, 2008. International Conference on ICCEE 2008., pp.918-921, 20-22 Dec. 2008
60. Chen, S., Kalbarczyk, Z., Xu, J., Iyer, R.K., "*A data-driven finite state machine model for analyzing security vulnerabilities*," Proceedings. 2003 International Conference on Dependable Systems and Networks, 2003., pp. 605- 614, 22-25 June 2003
61. Chen, T. M., Sanchez-Aarnoutse, J. C., Buford, J., "*Petri Net Modeling of Cyber-Physical Attacks on Smart Grid*," Smart Grid, IEEE Transactions on , vol.2, no.4, pp. 741-749.
62. Ten, C., Liu, C., Govindarasu, M., "*Vulnerability Assessment of Cybersecurity for SCADA Systems Using Attack Trees*" Power Engineering Society General Meeting, 2007. IEEE , pp.1-8, 24-28 June 2007
63. Khand, P.A., "*System level security modeling using attack trees*" Computer, Control and Communication, 2009. 2nd International Conference on IC4 2009., pp.1-6, 17-18 Feb. 2009
64. Ye, N., 2000. *A Markov Chain Model of Temporal Behavior for Anomaly Detection*. *Industrial Engineering*, 2(4), pp.6-7. Available at: [http://homepages.laas.fr/owe/METROSEC/DOC/WA1_1.pdf].
65. Radosavac, R., Baras, J., "*Detection and Classification of Network Intrusions using Hidden Markov Models*", Conference on Information Sciences and Systems, The Johns Hopkins University, March 12–14, 2003.

66. Idika, N., Mathur, A., *A Survey of Malware Detection Techniques*. 2007. Purdue University, p.48. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.4594>.
67. Robiah, Y., Rahaya, S., Mohd Zaki M, Sharin S., Faizal M. A., and Marliza R., 2009. *A New Generic Taxonomy on Hybrid Malware Detection Technique*. Journal of Computer Science, 5(1), p.6. Available at: [<http://arxiv.org/abs/0909.4860>].
68. Hofmeyr, S., Forrest, S., and Somayaji, A., *Intrusion detection using sequences of system calls*. 1998. J. Comput. Secur. 6, 3 (August 1998), 151-180.
69. Wang, K. and Stolfo, S., *Anomalous Payload-Based Network Intrusion Detection Recent Advances in Intrusion Detection*. 2004. E. Jonsson, A. Valdes and M. Almgren, Springer Berlin / Heidelberg. 3224: 203-222.
70. Matthew V. Mahoney. 2003. *Network traffic anomaly detection based on packet bytes*. In Proceedings of the 2003 ACM symposium on Applied computing (SAC '03). ACM, New York, NY, USA, 346-350. DOI=10.1145/952532.952601 [<http://doi.acm.org/10.1145/952532.952601>]
71. Masri, W. and Podgurski, A., 2005. *Using dynamic information flow analysis to detect attacks against applications*. ACM SIGSOFT Software Engineering Notes, 30(4), Available at: [<http://portal.acm.org/citation.cfm?doid=1082983.1083216>].
72. Sekar, R., Bowen, T.F., and Segal, M.E., *On Preventing Intrusions by Process Behavior Monitoring*. 1999. In Proceedings of the Workshop on Intrusion Detection and Network Monitoring. USENIX Association, Berkeley, CA, USA, 29-40.
73. Ko, C., Fink, G., Levitt, K. , *"Automated detection of vulnerabilities in privileged programs by execution monitoring,"* Computer Security Applications Conference, 1994. Proceedings., 10th Annual, pp.134-144, 5-9 Dec 1994
74. Kirda, E., Kruegel, C., Vigna, G., Jovanovic, N., *Noxes : A Client-Side Solution for Mitigating Cross-Site Scripting Attacks*. Proceedings of the 2006, p.330-337. Available at: [<http://portal.acm.org/citation.cfm?id=1141357>].
75. Ilgun, K., Kemmerer, R.A., and Porras, P.A., *"State Transition Analysis: A Rule-Based Intrusion Detection Approach"*, presented at IEEE Transactions on Software Engineering., 1995, pp.181-199.
76. Visumathi, J., and Shunmuganathan, K.L., *"A computational intelligence for evaluation of intrusion detection system "*, Indian Journal of Science and Technology. Vol. 4, no. 1, Jan 2011.
77. Lee, D., Kim, D., Jung, J., *"Multi-Stage Intrusion Detection System Using Hidden Markov Model Algorithm,"* Information Science and Security, International Conference on 2008. ICISS., pp.72-77, 10-12 Jan. 2008

78. Xiuqing, C., Yongping, Z., Jiutao, T., "HMM-based integration of multiple models for intrusion detection," 2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Vo12. pp. 137-140 .Aug. 2010
79. Xiuqing, C., Yongping, Z., Yu, G., "Adaptive intrusion prevention algorithm based on HMM Model," 2011 International Conference on E -Business and E -Government (ICEE),, pp.1-4, 6-8 May 2011
80. Tsai, C., Chang, A.Y., Chen, C., Yu, W., Chen, L. , "Dynamic intrusion detection system based on feature extraction and multidimensional hidden Markov model analysis,". 43rd Annual 2009 International Carnahan Conference on Security Technology, 2009, pp.85-88, 5-8 Oct. 2009
81. Provos, N., McNamee, D., Mavrommatis, D. W., K and Modadugu, N., *The Ghost In The Browser Analysis of Web-based Malware*. 2007. [Online]. Available at: http://www.usenix.org/events/hotbots07/tech/full_papers/provos/provos.pdf [Accessed 11 Feb 2009]
82. Polychronakis, M., Mavrommatis, P., and Provos, N., *Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware*. 2008. [Online]. Available at: <http://www.ics.forth.gr/dcs/Activities/papers/responders.leet08.pdf> [Accessed 11 Feb 2009]
83. Skoudis, E., and Zeltser, L., "Malware: Fighting Malicious Code", Prentice Hall, 2003, Page 3, ISBN = 978-0131014053.
84. Harris, S., Harper, A., Eagle, C., and Ness, J., "Gray Hat Hacking, Second Edition: The Ethical Hacker's Handbook". 2008, McGraw-Hill Osborne, Page 523, ISBN = 978-0071495684.
85. Seifert, C., *Know Your Enemy: Behind the Scenes of Malicious Web Servers*. 2007. [Online]. Available at: <http://honeynet.org/book/export/html/181> [Accessed 11 Feb 2009]
86. honeynet Project. *Know Your Enemy: Honeynets: What a honeynet is, its value, overview of how it works, and risk/issues involved*. 2006. [Online]. Available at: <http://old.honeynet.org/papers/honeynet/> [Accessed 11 Feb 2009]
87. Veysset, F., and Butti, L., *Honeypot technologies*. 2005. [Online]. Available at: <http://www.first.org/conference/2006/papers/veysset-franck-slides.pdf> [Accessed 11 Feb 2009]
88. Mokube, I., and Adams, M., *Honeypots: concepts, approaches, and challenges*. 2007. ACM Southeast Regional Conference, p.321-326. Available at: <http://portal.acm.org/citation.cfm?id=1233341.1233399>.

89. Seifert, C., Welch, I., Komisarczuk, P., HoneyC - *The Low-Interaction Client HoneyPot*, 2006. URL <http://www.mcs.vuw.ac.nz/~cseifert/blog/images/seifert-honeyc.pdf>
90. Provos, N., SpyBye.[<http://code.google.com/p/spybye>]. Last accessed; 11 Feb 2011.
91. İkinci, A., Holz, T., Freiling, F., *Monkey-Spider: Detecting Malicious Websites with Low-Interaction Honeyclients*. In: Proceedings of Sicherheit 2008 (2008). Available at: [<http://pi1.informatik.uni-mannheim.de/filepool/publications/monkey-spider.pdf>].
92. Wang, Y.M., Beck, D., Jiang, X., Roussev, R., Verbowski, C., Chen, S., and King, S., *Automated Web Patrol with Strider HoneyMonkeys*. 2006. In Symposium on Network and Distributed System Security (NDSS). Available at: [<ftp://ftp.research.microsoft.com/pub/tr/TR-2005-72.pdf>].
93. Moshchuk, A., Bragin, T., Gribble, S.D., and Levy, H.M., *A Crawler-based Study of Spyware on the Web*. In Proceedings of the 13th Annual Network and Distributed Systems Security Symposium (NDSS 2006), San Diego, CA, February 2006.
94. Russinovich, M., Cogswell, B., *FileMon for Windows v7.04*. 2006. Available at: [<http://technet.microsoft.com/en-us/sysinternals/bb896642.aspx>] Last accessed; 11 Feb 2009.
95. Russinovich, M., Cogswell, B., *RegMon for Windows v7.04*. 2006. Available at: [<http://technet.microsoft.com/en-gb/sysinternals/bb896652.aspx>]. Last accessed; 9 Feb 2009.
96. Russinovich, M., Cogswell, B., *Process Monitor v2.04*. 2009. Available at: <http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx> [Accessed 11 Feb 2009]
97. .NET Framework Class Library. *FileSystemWatcher Class*, [Online]. Available at: [[http://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher_\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/system.io.filesystemwatcher_(VS.80).aspx)] Last accessed; 11 Feb 2009.
98. Microsoft Help and Support. 2004. *How to Monitor TCP/IP Ports in Use*. [Online]. Available at: [<http://support.microsoft.com/kb/194938>] Last accessed; 11 Feb 2009.
99. Song, I., Eder, J. and Nguyen, T. *Data Warehousing and Knowledge Discovery: 9th International Conference, DaWaK 2007*. Springer.
100. Marczyk, “*Genetic algorithms and evolutionary computation*,” The Talk.Origins Archive, 2004. [Online]. Available at: [<http://www.talkorigins.org/faqs/genalg/genalg.html>] Last accessed; 11 June 2011.
101. Gebru, B., “*Genetic Algorithms for Solving Optimization Problems*”. Master of Science in Computational Science. [Online]. Available at:

[http://etd.aau.edu.et/dspace/bitstream/123456789/2660/1/coverpage%20_final.pdf]
Last accessed; 10 June 2011.

102. Pérez, Ó., Piccardi, M., García, J., Patricio, M.Á., Molina, J.M., *Comparison between genetic algorithms and the baum-welch algorithm in learning hMMs for human activity classification*. In: Giacobini, M. (ed.) *EvoWorkshops 2007*. LNCS, vol. 4448, pp. 399–406. Springer, Heidelberg (2007).
103. Wang, W.H.A., Tung, C.-L., Dynamic hand gesture recognition using hierarchical dynamic bayesian networks through low-level image processing, in: *International Conference on Machine Learning and Cybernetics*, Kunming, China, July 2008.
104. Panda Security, “Creation of New Malware Increases by 26 Percent, Reaching More than 73,000 Samples Every Day, According to PandaLabs”, MAR 16, 2011, [online] Available at: [<http://press.pandasecurity.com/usa/news/creation-of-new-malware-increases-by-26-percent-reaching-more-than-73000-samples-every-day-according-to-pandalabs/>] Last accessed; 7th Mar 2009.
105. Common Vulnerabilities and Exposures (CVE), “CVE-2005-0553”, [Online]. Available at: [<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0553>]. Last accessed; 4th Mar 2009.
106. Kruegel, C., Vigna, G., and Robertson, W., 2005. *A multi-model approach to the detection of web-based attacks*. Computer Networks. Elsevier North-Holland, Inc. New York, USA, 48, 5, pp.717-738.
107. Girault, A., Lee, B., Lee, E.A., "*Hierarchical finite state machines with multiple concurrency models* , 1999, " *Computer-Aided Design of Integrated Circuits and Systems*, IEEE Transactions on , vol.18, no.6, pp.742-760,
108. Raihan, M., Zulkernine, M., "*AsmLSec: An Extension of Abstract State Machine Language for Attack Scenario Specification*," *Availability, Reliability and Security*, 2007. ARES 2007. The Second International Conference on, pp.775-782.
109. Chen, E.Y., "*Detecting DoS attacks on SIP systems*," *VoIP Management and Security*, 2006. 1st IEEE Workshop on, pp. 53- 58.
110. Microsoft Security Bulletin, “MS05-020”, [Online]. Available at: [<http://technet.microsoft.com/en-us/security/bulletin/ms05-020>] Last accessed; 7th Mar 2009.
111. DeviceLock, *Active Ports 1.4*, [Online]. Available at: [<http://devicelock.com/freeware.html>] Last accessed; 11 Feb 2009.

112. Russinovich, M., and Cogswell, B., Process Explorer v11.33. [Online]. Available at: [<http://technet.microsoft.com/en-gb/sysinternals/bb896653.aspx>] Last accessed; 11 Feb 2009.
113. SkyLined, *Internet Explorer DHTML Arbitrary Code Execution (MS05-020)*, 2005, [Online]. Available at: [<http://www.securiteam.com/exploits/5CP0C0UFGG.html>] Last accessed; 3 Aug 2009.
114. Windows Workflow Foundation. [Online] Available at: [<http://msdn.microsoft.com/en-us/netframework/aa663328.aspx>] last accessed; 5 Aug 2009.
115. Gurrutxaga, I., Arbelaitz, O., Ma Perez, J., Muguerza, J., Martin, J.I. and Perona, I. *Evaluation of Malware clustering based on its dynamic behaviour*. (2008). In Proc. Seventh Australasian Data Mining Conference (AusDM 2008), Glenelg, South Australia. CRPIT, 87. Roddick, J. F., Li, J., Christen, P. and Kennedy, P. J., Eds. ACS. 163-170.
116. Malware URL. [online] Available at: [<http://www.malwareurl.com/listing-urls.php>]. Last accessed; 2 January 2011.
117. Malware Patrol. [online] Available at: [<http://www.malware.com.br/lists.shtml>] Last accessed; 2 January 2011.
118. Symantec Corporation. "Trojan.Apher" [Online]. Available at: [http://www.symantec.com/security_response/writeup.jsp?docid=2002-112211-1238-99]. Last accessed; 29 March 2011.
119. Souza, C.R., "*The Accord.NET Framework*.", 2010. [Online]. Available at: [<http://www.crsouza.com>]. Last accessed; 01 April 2011.
120. Snort, the Open Source Network Intrusion Detection System, [Online]. Available at: [<http://www.snort.org/>]. Last accessed; 21 Dec 2011.
121. RealPlayer, Cross-platform media player by RealNetworks, [Online]. Available at: [<http://uk.real.com/realplayer/>] Last accessed; 22 Dec 2011
122. The Swiss Security Blog [Online]. Available at: [<http://www.abuse.ch/?p=737>] Last accessed; 5th March 2009.
123. Mpack web-based exploit tool [Online]. Available at: [<http://blogs.pandasoftware.com/blogs/images/PandaLabs/2007/05/11/MPack.pdf>]. Last accessed 22 Dec 2011.
124. CVE-2006-5579 [Online]. Available at: <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5579>. Last accessed; 4th Mar 2011.

125. Microsoft Security Bulletin. 2009. MS06-072 [Online]. Available at: <http://www.microsoft.com/technet/security/Bulletin/ms06-072.mspx>. Last accessed 4th Mar 2009.
126. LeMay, R., "*The 12-minute Windows heist*". ZDNet Australia, July 1st, 2005. [Online]. Available at: [<http://www.zdnet.com.au/the-12-minute-windows-heist-139200021.htm>]. Last accessed: 11 Dec 2011.
127. Verizon RISK Team in cooperation with the United States Secret Service (USSS) "*2010 data breach investigations report*" 2010 .Available at: [http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf]. Last accessed: 4 Dec 2011.
128. Johnson, D, and Sinanovic, S., *Symmetrizing the Kullback-Leibler distance*. 2000. [Online]. Available at: [<http://www.ece.rice.edu/~dhj/resistor.pdf>]. Last accessed: 10 Dec 2011]
129. Unified Modelling Language (UML) Tutorial, [Online]. Available at: [http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/what_is_uml.htm], Last accessed: 10 Aug 2011.
130. Baker, W., Goudie, M., Hutton, A., Hylender, C., Niemantsverdriet, J., Novak, C., Ostertag, D., Porter, C., Rosen, M., Sartin, B., Tippet, P. (2010). *Verizon 2010 Data Breach Investigations Report. Verizon Business*. [Online]. Available at: [http://www.verizonbusiness.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf] Last accessed 20 Dec 2011.
131. VirusTotal. "*Free Online Virus and Malware Scan.*" [Online] Available at: [<http://www.virustotal.com/>]. Last accessed: 20 Dec 2011.
132. Microsoft Security Bulletin. "*Microsoft Security Bulletin MS06-044 - Critical*". 2006. [Online] Available at: <http://technet.microsoft.com/en-us/security/bulletin/ms06-044> last accessed: 21 Dec 2011.
133. Russell,P., *Netfilter: Firewalling, NAT and packet mangling for Linux 2.4*. 2001. Netfilter, [Online]. Available at: [<http://www.netfilter.org/>] last accessed: 21 Dec 2011]
134. Contos, B., "*Enemy at the Water Cooler: True Stories of Insider Threats and Enterprise Security Management Countermeasures*", Publisher: Syngress, 1st, 2006, ISBN-10: 1597491292.
135. Halkidi, M, Batistakis, Y and Vazirgiannis, M., "*On Clustering Validation Techniques*". *Journal of Intelligent Information Systems*, Vol. 17 (2001), pp. 107-145

136. Seifert, C. Stirling, D. Welch, I and Komisarczuk, P. 2008. "Internet NZ Study - Stage 1 Report". [online] Available at: http://internetnz.net.nz/system/files/workstreams/Stage_1_Honeypot_Report_0.pdf [Accessed 1 April 2012].
137. McFarland, B. 2005. "Ethical Deception and Preemptive Deterrence in Network Security". [online] Available at: http://www.sans.org/reading_room/whitepapers/firewalls/ethical-deception-preemptive-deterrence-network-security_1616 [Accessed 2 April 2012].
138. Wood, P, Egan, G, Haley, K, Tran, T, Cox, O, Lau, H, Wueest, C, McKinney, D, Millington, T, Nahorney, B, Mulcahy, J, Harrison, J, Parsons, T, Watson, A, Nisbet, M, Johnston, N, Krishnappa, B, Asrar, I, Hittel, S, Chien, E, Park, E, Maniyara, M, Thonnard, O, Vervier, P, Lee, M, Lewis, D and Wallace, S. "Internet Security Threat Report" 2011, Volume 17, Published April 2012. [online] Available at: <http://www.symantec.com/threatreport/> [Accessed 13 May 2012].
139. O. W. A. S. Project, "The ten most critical web application security vulnerabilities." 2010, [online] Available https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project , [Accessed 25 May 2012].