MALICIOUS URL DETECTION BY DYNAMICALLY MINING PATTERNS WITHOUT PRE-DEFINED ELEMENTS

by

Da Huang

B.Sc., South China University of Technology, 2010

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

© Da Huang 2012
SIMON FRASER UNIVERSITY
Spring 2012

All rights reserved.
However, in accordance with the *Copyright Act of Canada*, this work may be reproduced without authorization under the conditions for "Fair Dealing." Therefore, limited reproduction of this work for the purposes of private study, research, criticism, review and news reporting is likely to be in accordance with the law, particularly if cited appropriately.

APPROVAL

Name: Da Huang

Degree: Master of Science

Title of Thesis: Malicious URL Detection by Dynamically Mining Patterns without Pre-defined Elements

Examining Committee: Dr. Qianping Gu
Chair

____

Dr. Jian Pei, Senior Supervisor

____

Dr. Binay Bhattacharya, Supervisor

____

Dr. Ke Wang, SFU Examiner

Date Approved: April 18, 2012

Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection (currently available to the public at the "Institutional Repository" link of the SFU Library website (www.lib.sfu.ca) at http://summit/sfu.ca and, without changing the content, to translate the thesis/project or extended essays, if technically possible, to any medium or format for the purpose of preservation of the digital work.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

While licensing SFU to permit the above uses, the author retains copyright in the thesis, project or extended essays, including the right to change the work for subsequent purposes, including editing and publishing the work in whole or in part, and licensing other parties, as the author may desire.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2011

Abstract

Detecting malicious URLs is an essential task in network security intelligence. In this thesis, we make two new contributions beyond the state-of-the-art methods on malicious URL detection. First, instead of using any pre-defined features or fixed delimiters for feature selection, we propose to dynamically extract lexical patterns from URLs. Our novel model of URL patterns provides new flexibility and capability on capturing malicious URLs algorithmically generated by malicious programs. Second, we develop a new method to mine our novel URL patterns, which are not assembled using any pre-defined items and thus cannot be mined using any existing frequent pattern mining methods. Our extensive empirical study using the real data sets from Fortinet, a leader in the network security industry, clearly shows the effectiveness and efficiency of our approach. The data sets are at least two orders of magnitudes larger than those reported in literature.

To my family.

"Knowledge talks, wisdom listens."

— Jimi Hendrix *(1942 - 1970)*

Acknowledgments

I would like to express my sincerest gratitude to my senior supervisor, Dr. Jian Pei, who has supported me thoughout my thesis with his patience and knowledge. He taught me a lot of skills that I will benefit in my future research and life. I thank him for his valuable advice, warm encouragement and care. Without him, never can I accomplish this thesis.

I am grateful to my supervisor, Dr. Binay Bhattacharya, for his time reviewing my work and helpful suggestions that helped me to improve my thesis. I would like to thank Dr. Ke Wang and Dr.Qianping Gu for serving my examing committee. My gratitude also goes Dr. Jian Chen at South China University of Technology for inspiring me to work in the field of data mining.

I thank Ruiwen Chen, Yi Cui, Bin Jiang, Qiang Jiang, Luping Li, Hossein Maserrat, Xiao Meng, Guanting Tang, Bin Zhou and Guangtong Zhou for their kind help during my study at SFU. I am also grateful to my friends at Fortinet. I thank Kai Xu, for his guide and insight suggestions. A particular acknowledgement to Raymond Chan, for his great help and warm care.

My deepest gratitude goes to my parents and my brother. Their endless love supports me to overcome all the difficulties in my study and life.

Contents

Approval ii

Abstract iii

Dedication iv

Quotation v

Acknowledgments vi

Contents vii

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Malicious URL Detection 1

1.2 Bussiness Backgroud 3

1.3 Lexical Features of URLs 3

1.4 Contributions 3

1.5 Thesis Organization 4

2 Related Work 5

3 Extracting Lexical Features 8

3.1 URL Segment Patterns 8

3.2 Finding Maximal Segment Patterns 10

3.3 URL Segment Sequential Patterns 12
3.4 URL Patterns 14

4 Mining Patterns 15
4.1 The Complete Pattern Set Algorithm (CA) 15
4.2 Quality of URL Patterns 17
4.3 The Greedy Selection Algorithm (GA) 18
4.4 Indexing 18

5 Experimental Results 20
5.1 Data Sets 20
5.2 Effectiveness of URL Patterns 22
5.3 Effectiveness of Malicious Probability Ratio Threshold 22
5.4 The CA Algorithm and the GA Algorithm 24
5.5 Effect of Training Data Set Size 29
5.5.1 White training dataset 29
5.5.2 Malicious training dataset 29
5.6 Effect of Dynamic Feature Extraction 30
5.7 Comparison with Machine Learning Methods 32

6 Conclusions 36

Bibliography 38

List of Tables

5.1 Some statistics of the malicious data set 21

5.2 Some statistics of the white data set 21

5.3 Some top ranking and bottom ranking malicious URL patterns 23

5.4 Some malicious URL patterns 24

5.5 Some malicious URLs covered by malicious URL patterns 26

List of Figures

4.1 Bottom-up search of URL patterns 16

5.1 Malicious probability ratio distribution of URL patterns 23
5.2 Detection under different malicious probability ratio thresholds 25
5.3 Malicious URL hit count distribution 27
5.4 Comparison between the CA and GA algorithm in different malicious data sets 28
5.5 Comparison between the CA and GA algorithm 29
5.6 The GA algorithm in different white training data sets 30
5.7 Runtime and number of patterns in different malicious training data sets . . . 31
5.8 ROC curve of the GA algorithm in different malicious training data sets . . 32
5.9 Algorithms with and without dynamic features 33
5.10 Comparison between the GA algorithm and WEKA machine learning methods 34

Chapter 1

Introduction

A web threat refers to any threat that uses the internet to facilitate cybercrime [15]. In practice, web threats may use multiple types of malware and fraud. A common feature is that web threats all use HTTP or HTTPS protocols, though some threats may additionally use other protocols and components, such as links in emails or IMs, or malware attachments. Through web threats, cyber-criminals often steal private information or hijack computers as bots in botnets. It has been well realized that web threats lead to huge risks, including financial damages, identity thefts, losses of confidential information and data, thefts of network resources, damaged brand and personal reputation, and erosion of consumer confidence in e-commerce and online banking. For example, Gartner [2] estimated that phishing attacks alone caused 3.6 million adults losing 3.2 billion US dollars in the period from September 2006 to August 2007, and the global cost of spam in 2007 was about 100 billion US dollars. The cost of spam management in the U.S. alone was estimated 71 billion dollars in 2007 [4]. In Web 2.0 applications, users are even more vulnerable to web threats due to the increasing online interactivity.

1.1 Malicious URL Detection

Although the exact adversary mechanisms behind web criminal activities may vary, they all try to lure users to visit malicious websites by clicking a corresponding URL (Uniform Resource Locator). A URL is called *malicious* (also known as *black*) if it is created in a malicious purpose and leads a user to a specific threat that may become an attack, such as spyware, malware, and phishing. Malicious URLs are a major risk on the web. Therefore,

detecting malicious URLs is an essential task in network security intelligence.

In practice, malicious URL detection faces several challenges.

- *Realtime detection.* To protect users effectively, a user should be warned before she/he visits a malicious URL. The malicious URL detection time should be very short so that users would not have to wait for long and suffer from poor user experience.

- *Detection of new URLs.* To avoid being detected, attackers often create new malicious URLs frequently. Therefore, an effective malicious URL detection method has to be able to detect new, unseen malicious URLs. In practice, the capability of detecting new, unseen malicious URLs is of particular importance, since emerging malicious URLs often have high hit counts, and may cause serious harms to users.

- *Effective detection.* The detection should have a high accuracy. When the accuracy is of concern, the visit frequency of URLs should also be considered. From a user's point of view, the accuracy of a detection method is the number of times that the detection method classifies a URL correctly versus the number of times that the method is consulted. Please note that a URL may be sent to a detection method multiple times, and should be counted multiple times in the accuracy calculation. Therefore, detecting frequently visited URLs correctly is important.

  Similarly, it is highly desirable that a malicious URL detection method should have a high recall so that many malicious URLs can be detected. Again, when recall is calculated in this context, the visit frequency of URLs should be considered.

To meet the above challenges, the latest malicious URL detection methods try to build a classifier based on URLs. *A foundamental assumption is that a clean training sample of malicious URL and good URL samples is available.* Such methods segment a URL into tokens using some delimiters, such as "/" and "?", and use such tokens as features. Some methods also extract additional features, such as WHOIS data and geographic properties of URLs. Then, machine learning methods are applied to train a classification model from the URL sample.

1.2 Bussiness Backgroud

Most network security enterprises do malicious URL detection by rating engines which are powered by various techniques, such as manual labeling, user feedback, and web content analysis. Although these rating engines can capture a large portion of the malicious URLs on the web, some malicious URLs, especially the new ones, are still not detected and can be harmful to the users. Method to further increase the detection coverage is highly desirable for enterprises. In this thesis, we introduce an approach to promote the detection coverage of rating engines by mining patterns from the URL log data.

1.3 Lexical Features of URLs

Besides URLs, people also do classification using lexical features in some other areas, such as text mining and pattern mining in bioinformatics. The lexical features of URLs are different from those features in other areas. In bioinformatics, lexical features are composed of a small number of elements and they can be very long, while the features of URLs are much shorter. In addition, lexical features in URLs are more likely to be some meaningful tokens which are real words or similar to some real words. On the other hand, comparing to the features in text mining, there are no clear separators, such as space or punctuations, between features in URLs.

Due to those differences, it may not be effective to directly use the lexical feature extraction methods from other areas on URLs. For example, Sittichai *et al.* [5] conducted biological entity classification using n-grams as lexical features. These fixed n-grams, however, may not be able to extract the flexible meaningful tokens in URLs effectively.

1.4 Contributions

In this thesis, we make two new contributions beyond the state-of-the-art methods on malicious URL detection. First, instead of using any pre-defined features or fixed delimiters for feature selection, we propose to dynamically extract lexical patterns from URLs. Our novel model of URL patterns provides new flexibility and capability on capturing malicious URLs algorithmically generated by malicious programs. For example, our method can extract patterns like "`*07db0*/get2.php`" where $*$ is a wildcard symbol. Second, we develop a new method to mine our novel URL patterns, which are not assembled using any pre-defined

items and thus cannot be mined using any existing frequent pattern mining methods. Our extensive empirical study using the real data sets from Fortinet, a leader in the network security industry, clearly shows the effectiveness and efficiency of our approach. The data sets are at least two orders of magnitudes larger than those reported in literature.

1.5 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we review the state-of-the-art methods and point out how our method is different from them. We discuss our lexical feature extraction method in Chapter 3, and devise our pattern mining method in Chapter 4. We report our empirical study results in Chapter 5, and conclude the thesis in Chapter 6.

Chapter 2

Related Work

The existing work on malicious URL detection can be divided into three categories, namely the blacklist based methods, the content based methods, and the URL based methods.

The *blacklist based methods* maintain a blacklist of malicious URLs. During detection, a URL is reported as malicious if it is in the blacklist. Most of the current commercial malicious URL detection systems, such as Google Safebrowsing (`http://www.google.com/tools/firefox/safebrowsing/index.html`), McAfee SiteAdvisor (`http://www.siteadvisor.com`), Websense ThreatSeeker Network (`http://www.websense.com/content/threatseeker.asp`), and Fortinet URL lookup tool (`http://www.fortiguard.com/ip_rep.php`), use some blacklist based methods. In such detection methods, the blacklists may be created and maintained through various techniques, such as manual labeling, honeyports, user feedbacks and crawlers. The blacklist based methods are simple and have a high accuracy. At the same time, they are incapable of detecting new, unseen malicious URLs, which often cause big harms to users.

The *content based methods* analyze the content of the corresponding web page of a URL to detect whether the URL is malicious. Web page content provides rich features for detection. For example, Provos *et al.* [13] detected malicious URLs using features from the content of the corresponding URLs, such as the presence of certain javascript and whether iFrames are out of place. Moshchuk *et al.* [11] used anti-spyware tools to analyze downloaded trojan executables in order to detect malicious URLs. The content based methods are useful for offline detection and analysis, but are not capable of online detection. For online detection, the content based methods often incur significant latency, because scanning and analyzing page content often costs much computation time and resource.

Most recently, the *URL based methods* use only the URL structures in detection, even without using any external information, such as WHOIS, blacklists or content analysis. McGrath and Gupta [10] analyzed the differences between normal URLs and phishing URLs in some features, such as the URL length, domain name length, number of dots in URLs. Such features can be used to construct a classifier for phishing URL detection. Yadav *et al.* [16] examined more features, such as the differences in bi-gram distribution of domain names between normal URLs and malicious ones. Their study confirmed that normal URLs and malicious ones indeed have distinguishable differences in the features extracted from URLs themselves alone. Their study, however, did not propose any classifier for malicious outlier detection. Ma *et al.* [8] applied machine learning methods to construct classifiers for malicious URL detection. They used two kinds of features. The *host-based features* are related to the host information such as IP addresses, WHOIS data, and the geographic properties of the URLs. Those features are highly valuable for classification, but they may cause non-trivial latency in detection. To this extent, their method is not completely URL based. The second group of features they used is the *lexical features*, which include numerical information, such as lengths of features, number of delimiters, and existence of tokens in the hostname and in the path of the URL. Le *et al.* [7] showed that using only the lexical features still can retain most of the performance in phishing URL detection.

Our method proposed in this thesis is URL based, and uses only the lexical features. Our method differs from [8, 7] on two aspects: how to select the lexical features and how to use the features in malicious URL detection.

In the previous studies [8, 7], the lexical features are the tokens in the URL strings delimited by characters "/", "?", ".", "=", "-", and "_". Tokens extracted using those delimiters may not be effective. For example, the previous method may not be able to identify pattern "`*biz*main.php`" from malicious URL segments "`mybiz12832main1.php`" and "`godbiz32421main2.php`". To tackle the problem, we develop a method to extract lexical features automatically and dynamically. Our method is more general than the existing methods in lexical feature extraction, and can use more flexible and informative lexical features.

The previous studies [8, 7] treated lexical features as binary ones, and simply applied machine learning methods on those features to construct classifiers. They cannot be scalable on large training data sets due to the large number of features. The training samples used in those studies are at the scale of tens of thousands URLs. To tackle the scalability problem,

we propose a pattern mining approach. Please note that the traditional frequent pattern mining methods [3] do not work well here, since the items, which are lexical features, cannot be predefined. We devise a method to combine pattern mining and lexical feature mining that can scale up to large training data sets. We report our experimental results on real data sets with more than one million URLs, which are two orders of magnitudes larger than those reported in literature.

Chapter 3

Extracting Lexical Features

A URL can be regarded as a sequence of **URL segments**, where a URL segment is a domain name, a directory name, or a file name. As discussed in Chapter 2, the previous studies [8, 7] often treat a segment as a token in mining patterns and extracting features for malicious URL detection. Simply treating each URL segment as a token, however, may limit the detection of malicious URLs. For example, suppose we have malicious URLs containing segments "`mybiz12832main1.php`" and "`lucybiz32421main2.php`". If we treat a segment in whole as a token, we cannot extract the meaningful common substrings "`biz`" and "`main`" in those two segments.

In this chapter, we discuss how to extract the common substrings as lexical features in URLs that may be generated by the same malicious program. Such patterns will be the building blocks later for malicious URL detection. We start from URL segments, and then extend to segment sequences and URLs.

3.1 URL Segment Patterns

A URL segment is a string of predefined characters, as specified in the URL specification (`http://www.w3.org/Addressing/URL/url-spec.txt`). We define a URL segment pattern as follows.

Definition 1 (Segment pattern)**.** *A **URL segment pattern** (or a* **segment pattern** *for short) is a string $s = c_1 \cdots c_l$, where c_i $(1 \leq i \leq l)$ is a normal character defined by the URL specification, or $c_i = *$ where $*$ is a wildcard meta-symbol. Denote by $|s| = l$ the length of*

the segment pattern, and by $s[i] = c_i$ *the* i*-th character in the segment pattern. We constrain that for any* i $(1 \leq i < l)$*, if* $c_i = *$*, then* $c_{i+1} \neq *$*.*

For two URL segment patterns $s = c_1 \cdots c_l$ *and* $s' = c'_1 \cdots c'_m$*,* s *is said to* **cover** s'*, denoted by* $s \sqsupseteq s'$*, if there is a function* $f : [1, m] \rightarrow [1, l]$ *such that*

1. $f(j) \leq f(j+1)$ *for* $(1 \leq j < m)$*; and*

2. *for any* i $(1 \leq i \leq l)$*, if* $c_i \neq *$*, then there exists a unique* j $(1 \leq j \leq m)$ *such that* $f(j) = i$*, and* $c'_j = c_i$*.* ■

Trivially, a URL segment itself is a segment pattern that contains no "*" symbols.

Example 1 (URL segment pattern)**.** *"`*biz*main*.php`" is a segment pattern. It covers URL segment "`godbiz32421main2.php`". Segment pattern "`abc*xyz`" does not cover "`xyzabc`", and "`abc*abc`" does not cover "`abc`".* ■

The cover relation on all possible segment patterns form a partial order.

Property 1. *Let* $\mathcal{S}$ *be the set of all possible segment patterns.* $\sqsupseteq$ *is a partial order on* $\mathcal{S}$*.*

Proof. (Reflexivity) $s \sqsupseteq s$ holds for any segment pattern s by setting $f(i) = i$ for $1 \leq i \leq |s|$.

(Antisymmetry) Consider two segment patterns $s = c_1 \cdots c_l$ and $s' = c'_1 \cdots c'_m$. Suppose $s \sqsupseteq s'$ under function $f : [1, m] \rightarrow [1, l]$ and $s' \sqsupseteq s$ under function $f' : [1, l] \rightarrow [1, m]$. We show $c_i = c'_i$ where $1 \leq i \leq m$ by induction.

(The basis step) If $c_1 \neq *$, then there exists a unique j_1 $(1 \leq j_1 \leq m)$ such that $f(j_1) = 1$ and $c'_{j_1} = c_1$. If $j_1 > 1$, then $f(j_1 - 1) < f(j_1) = 1$, a contradiction to the assumption that $f : [1, m] \rightarrow [1, l]$. Thus, $c_1 = c'_1$.

If $c_1 = *$, we assume $c'_1 \neq *$. Then, there exists a unique i_1 $(1 \leq i_1 \leq l)$ such that $f'(i_1) = 1$ and $c_{i_1} = c'_1$. Since $c_1 = * \neq c'_1$, $i_1 > 1$. This leads to $f'(i_1 - 1) < f(i_1) = 1$, and a contradiction to the assumption that $f' : [1, l] \rightarrow [1 : m]$. Thus, $c'_1 = * = c_1$.

(The inductive step) Assume that $c_i = c'_i$ for $1 \leq i \leq k$ $(1 \leq k < m)$. We consider two cases. First, if $c_k \neq *$ and thus $c'_k \neq *$, then, using an argument similar to that in the basis step, we can show $c'_{k+1} = c_{k+1}$. Second, if $c_k = c'_k = *$, then $c_{k+1} \neq *$ and $c'_{k+1} \neq *$. There exists a unique j_{k+1} $(k + 1 \leq j_{k+1} \leq m$ such that $f(j_{k+1}) = k + 1$ and $c'_{j_{k+1}} = c_{k+1}$. If $j_{k+1} > k + 1$, then $f(j_{k+1} - 1) < f(j_{k+1}) = k + 1$. This leads to a contradiction to the assumption that $c_i = c'_i$ for $1 \leq i \leq k$ and $c_k \neq *$. Thus, $j_{k+1} = k + 1$, and $c_{k+1} = c'_{k+1}$.

(Transitivity) Consider three URL segment patterns $s = c_1 \cdots c_l$, $s' = c'_1 \cdots c'_m$, and $s'' = c''_1 \cdots c''_n$. Suppose $s \sqsupseteq s'$ under function $f : [1, m] \to [1, l]$ and $s' \sqsupseteq s''$ under function $f' : [1, n] \to [1, m]$. We can construct a function $g : [1, n] \to [1, l]$ by $g = f \circ f'$. For any $1 \leq i < n$, since $f'(i) \leq f'(i+1)$, $f(f'(i)) \leq f(f'(i+1))$, that is, $g(i) \leq g(i+1)$. Moreover, for any j $(1 \leq j \leq l)$, if $c_j \neq *$, then there exists a unique k $(1 \leq k \leq m)$ such that $c'_k = c_j$ and $f(k) = j$. Since $c'_k \neq *$, there exists a unique i $(1 \leq i \leq n)$ such that $c''_i = c'_k = c_j$ and $f'(i) = k$. Thus, $f(f'(i)) = f(k) = j$. Under function g, $s \sqsupseteq s''$. ∎

Definition 2 (Maximal segment patterns)**.** *Given a set of URL segments or URL segment patterns $S = \{s_1, \ldots, s_n\}$, a URL segment pattern s covers S, denoted by $s \sqsupseteq S$, if for each $s_i \in S$, $s \sqsupseteq s_i$.*

Segment pattern s is called a **maximal segment pattern** *with respect to S if $s \sqsupseteq S$ and there exist no other segment pattern $s' \sqsupseteq S$ such that $s \sqsupseteq s'$ and $s \neq s'$. Denote by $MAX(S) = \{s | s$ is a maximal segment pattern with respect to $S\}$ the set of maximal segment patterns.* ∎

Example 2 (Maximal segment patterns)**.** *Consider $S = \{$`abcabc`, `abcaabc`$\}$. Segment pattern "`abc*abc`" is maximal with respect to S. Although `ab*abc` $\sqsupseteq S$, it is not a maximal segment pattern, since `ab*abc` $\sqsupseteq$ `abc*abc` and `ab*abc` $\neq$ `abc*abc`.*

Interestingly, given a set of segments, there may exist more than one maximal segment patterns. For example, "`abca*bc`" is another maximal segment pattern with respect to S. $MAX(S) = \{$`abc*abc`, `abca*bc`$\}$. ∎

3.2 Finding Maximal Segment Patterns

Given a set S of segments extracted from a set of malicious URLs, how can we compute $MAX(S)$? If S contains only two segments, the problem is similar to the longest common subsequence (LCS) problem and the longest common substring problem [9, 1]. Following from the known results on those problems, we can use dynamic programming to compute the maximal segment patterns.

Let D be a set of segment patterns, we define a function $\Omega(D) = \{s | s \in D, \nexists s' \in D, s \sqsupseteq s', s \neq s'\}$, which selects the maximal segment patterns from a set. Moreover, for a segment $s = c_1 \cdots c_l$, we write the prefix $s[1, i] = c_1 \cdots c_i$ for $1 \leq i \leq l$. Trivially, when $i > l$, $s[1, i] = s$.

We also write $s \diamond c$ a sequence where a character c is appended to the end of s. Specifically, if a `*` is appended to a sequence ended by a `*`, only one `*` is kept at the end of the sequence. For example, `abc` $\diamond$ `*` = `abc*` and `abc*` $\diamond$ `*` = `abc*`. Moreover, for a set of segment patterns D, we write $D \diamond c = \{s \diamond c | s \in D\}$.

Given two URL segments s and s', denote by $MAX_{s,s'}(i, j)$ the set of maximal segment patterns in the set $\{s[1, i], s'[1, j]\}$. Apparently, $MAX_{s,s'}(|s|, |s'|) = MAX(\{s, s'\})$. We can compute the function using dynamic programming by

$$MAX_{s,s'}(i,j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ \Omega(\{MAX_{s,s'}(i-1, j-1) \diamond s[i], & \\ MAX_{s,s'}(i-1, j) \diamond *, MAX_{s,s'}(i, j-1) \diamond *\}) & \text{if } s[i] = s'[j] \\ \Omega(\{MAX_{s,s'}(i-1, j), MAX_{s,s'}(i, j-1)\}) \diamond * & \text{if } s_i \neq s'_j \end{cases}$$

Consider two segment patterns s and s'. Without loss of generality, assume $|s| \geq |s'|$. At each step of computing $MAX_{s,s'}(i, j)$, function Ω is called once, which takes time $O(|D|^2(i + j))$, where D is the set of segment patterns where the maximal segment patterns are derived. Since D is often small, at most 2 in our experiments, we treat $|D|$ as a constant. Thus, the complexity of the dynamic programming algorithm is $\sum_{1 \leq i \leq |s|, 1 \leq j \leq |s'|}(i + j) = O(|s|^2|s'|)$.

To compute the maximal segment patterns on a set of URL segments, we have the following result.

Theorem 1. *Let $S = \{s_1, \ldots, s_n\}$ $(n > 2)$ be a set of URL segments.*

$$MAX(S) = \Omega(\bigcup_{s \in MAX(S - \{s_n\})} MAX(\{s_n, s\}))$$

Proof. Consider a segment pattern $s' \in MAX(S)$. Apparently, $s' \sqsupseteq s_n$ and $s' \sqsupseteq S - \{s_n\}$. Thus, there exists at least one segment pattern $s'' \in \Omega(\bigcup_{s \in MAX(S-\{s_n\})} MAX(\{s_n, s\})$ such that $s' \sqsupseteq s''$. For each segment pattern $s'' \in \Omega(\bigcup_{s \in MAX(S-\{s_n\})} MAX(\{s_n, s\})$, $s'' \sqsupseteq S$. If $s' \neq s''$, then s' is not a maximal segment pattern. This contradicts to the assumption $s' \in MAX(S)$. Therefore, $MAX(S) \subseteq \Omega(\bigcup_{s \in MAX(S-\{s_n\})} MAX(\{s_n, s\})$. Moreover, due to the Ω function, $MAX(S) = \Omega(\bigcup_{s \in MAX(S-\{s_n\})} MAX(\{s_n, s\})$. ∎

To avoid some segment patterns that are too general to be useful in malicious URL detection, such as "`*a*`", we constrain that the non-* substrings in a segment pattern must have at least 3 characters of numbers or letters.

3.3 URL Segment Sequential Patterns

We can treat a URL as a sequence of URL segments.

Definition 3 (URL segment sequential pattern)**.** *A* ***URL segment sequential pattern*** *(or* **sequential pattern** *for short)* $u = \langle s_1, \ldots, s_l \rangle$ *is a sequence of URL segment patterns.* $|u| = l$ *is the length of the sequential pattern.*

For two sequential patterns $u = \langle s_1, \ldots, s_l \rangle$ *and* $u' = \langle s'_1, \ldots, s'_m \rangle$*, u is said to* **cover** u'*, denoted by* $u \sqsupseteq u'$*, if* $m \geq l$ *and there is a function* $f : [1, l] \rightarrow [1, m]$ *such that* $s_i \sqsupseteq s'_{f(i)}$ $(1 \leq i \leq l)$*, and* $f(j) < f(j+1)$ $(1 \leq j < l)$*.* ■

Please note that, to keep our discussion simple, we do not have a wildcard meta-symbol at the sequential pattern level. When u covers u', the segment patterns in u cover the segment patterns in a subsequence of u' one by one.

Example 3 (URL segment sequential pattern)**.** *⟨`20207db*.deanard.*`, `file4`, `get*.php`⟩ is a segment sequential pattern. It covers ⟨`20207db09.deanard.com`, `dir`, `file4`, `get2.php`⟩. Sequential pattern ⟨`20207db*`, `*deanard*`⟩ does not cover ⟨`20207db09.deanard.com`⟩.* ■

Similar to segment patterns, the cover relation on all possible sequential patterns form a partial order.

Property 2. *Let* $\mathcal{U}$ *be the set of all possible URL segment sequential patterns.* $\sqsupseteq$ *is a partial order on* $\mathcal{U}$*.*

Proof. (Reflexivity) $u \sqsupseteq u$ holds for any sequential pattern u by setting $f(i) = i$ for $1 \leq i \leq |u|$.

(Antisymmetry) Consider two sequential patterns $u = \langle s_1, \ldots, s_l \rangle$ and $u' = \langle s'_1, \ldots, s'_m \rangle$. Suppose $u \sqsupseteq u'$ under function $f : [1, l] \rightarrow [1, m]$ and $u' \sqsupseteq u$ under function $f' : [1, m] \rightarrow [1, l]$. Apparently, $|u| = |u'|$, otherwise the longer sequential pattern cannot cover the shorter sequential pattern. Since $f(i) < f(i+1)$ and $|u| = |u'|$, for any $1 \leq i \leq l$, $f(i) = i$, $s_i \sqsupseteq s'_i$. In the same way, for any $1 \leq i \leq m$, $s'_i \sqsupseteq s_i$. By the antisymmetry in property 1, $s_i = s'_i$.

(Transitivity) Consider three sequential patterns $u = \langle s_1 \ldots s_l \rangle$, $u' = \langle s'_1 \ldots s'_m \rangle$, and $u'' = \langle s''_1 \ldots s''_n \rangle$. Suppose $u \sqsupseteq u'$ under function $f : [1, l] \rightarrow [1, m]$ and $u' \sqsupseteq u''$ under function $f' : [1, m] \rightarrow [1, n]$. We can construct a function $g : [1, l] \rightarrow [1, n]$ by $g = f \circ f'$. For any $1 \leq i < l$, since $f(i) < f(i+1)$, $f'(f(i)) < f'(f(i+1))$, that is, $g(i) < g(i+1)$.

Moreover, for any $1 \leq i \leq l$, $s_i \sqsupseteq s'_{f(i)}$, and for any $1 \leq f(i) \leq m$, $s'_{f(i)} \sqsupseteq s''_{f'(f(i))}$, by the transitivity in property 1, $s_i \sqsupseteq s''_{f'(f(i))}$. Under function g, $u \sqsupseteq u''$. ■

We can also define maximal sequential patterns.

Definition 4 (Maximal URL segment sequential pattern)**.** *Given a set of URL segment sequential patterns $U = \{u_1, \ldots, u_n\}$, a URL segment sequential pattern u is said to **cover** U, denoted by $u \sqsupseteq U$, if for each $u_i \in U$, $u \sqsupseteq u_i$.*

A URL segment sequential pattern u is called a **maximal URL segment sequential pattern** *(or **maximal sequential pattern** for short) with respect to U if $u \sqsupseteq U$ and there exists no other sequential pattern $u' \sqsupseteq U$ such that $u \sqsupseteq u'$ and $u \neq u'$.* ■

Example 4 (Maximal sequential patterns)**.** *Consider $U = \{\langle$`abcabc, index`$\rangle, \langle$`abcaabc, index`$\rangle\}$. Sequential pattern $\langle$`abc*abc, index`$\rangle$ is maximal with respect to U. Although $\langle$`ab*abc, index`$\rangle \sqsupseteq U$, it is not maximal, since $\langle$`ab*abc, index`$\rangle \sqsupseteq \langle$`abc*abc, index`$\rangle$.*

Given a set of sequential patterns, there may exist more than one maximal sequential patterns. For example, $\langle$`abca*bc, index`$\rangle$ is another maximal sequential pattern with respect to U. ■

Similar to mining maximal segment patterns, we can use dynamic programming to compute maximal sequential patterns. We denote by $seqMAX(U)$ the set of maximal sequential patterns with respect to a set of sequential patterns U.

Let E be a set of sequential patterns, we define a function $\Omega(E) = \{u | u \in E, \nexists u' \in E, u \sqsupseteq u', u \neq u'\}$, which selects the maximal sequential patterns from a set. Moreover, for a sequential pattern $u = \langle s_1, \ldots, s_l \rangle$, we write the prefix $u[1, i] = \langle s_1 \cdots s_i \rangle$ for $1 \leq i \leq l$. Trivially, when $i > l$, $u[1, i] = u$. We also write $u[i] = s_i$ and $u'[i] = s'_i$.

Given two URL sequential patterns u and u', denote by $seqMAX_{u,u'}(i, j)$ the set of maximal sequential patterns in the set $\{u[1, i], u'[1, j]\}$. Apparently, $seqMAX_{u,u'}(|u|, |u'|) = seqMAX(\{u, u'\})$. We can compute the function by

$$seqMAX_{u,u'}(i,j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ \bigcup\limits_{s \in MAX(u[i], u'[j])} (seqMAX_{u,u'}(i-1, j-1), s) & \text{if } MAX(u[i], u'[j]) \neq \emptyset \\ \Omega(\{seqMAX_{u,u'}(i-1, j), seqMAX_{u,u'}(i, j-1)\}) & \text{if } MAX(u[i], u'[j]) = \emptyset \end{cases}$$

To compute the maximal sequential pattern on a set of segment sequences, we have the following result.

Theorem 2. *Let $U = \{u_1, \ldots, u_n\}$ ($n > 2$) be a set of sequential patterns. Then, $seqMAX(U) = seqMAX(seqMAX(U - \{u_n\}) \cup \{u_n\})$.*

Proof. Consider a sequential pattern $u' \in seqMAX(U)$. Apparently, $u' \sqsupseteq u_n$ and $u' \sqsupseteq U - \{u_n\}$. Thus, there exists at least one sequential pattern $u'' \in seqMAX(seqMAX(U - \{u_n\}) \cup \{u_n\})$ such that $u' \sqsupseteq u''$. For each segment pattern $u'' \in seqMAX(seqMAX(U - \{u_n\}) \cup \{u_n\})$, $u'' \sqsupseteq U$. If $u' \neq u''$, then u' is not a maximal sequential pattern. This contradicts to the assumption $u' \in seqMAX(U)$. Therefore, $seqMAX(U) \subseteq seqMAX(seqMAX(U - \{u_n\}) \cup \{u_n\})$. Moreover, due to the $seqMAX$ function, $seqMAX(U) = seqMAX(seqMAX(U - \{u_n\}) \cup \{u_n\})$. ■

3.4 URL Patterns

Although we can treat a URL as a sequence of URL segment patterns, due to the big differences among the roles of domain name, directories, and file name in a URL, we clearly distinguish those three parts in our definition of URL patterns.

Definition 5 (URL pattern)**.** *A* **URL pattern** *is a tuple $p = (h, d, f)$, where h is a URL segment pattern corresponding to the domain name, $d = \langle s_1, \ldots, s_n \rangle$ is a URL sequential pattern corresponding to the directory path, and f is a URL segment pattern represent the file name.*

For two URL pattern $p = (h, d, f)$ and $p' = (h', d', f')$, p is said to ***cover*** *p', denoted by $p \sqsupseteq p'$, if h covers h', f covers f', and d covers d'.* ■

Definition 6 (Maximal URL pattern)**.** *Given a set of URL patterns $P = \{p_1, \ldots, p_n\}$, a URL pattern p covers P, denoted by $p \sqsupseteq P$, if for each $p_i \in P$, $p \sqsupseteq p_i$.*

A URL pattern p is called a **maximal URL pattern** *with respect to P if $p \sqsupseteq P$ and there exists a URL pattern $p' \sqsupseteq P$ such that $p \sqsupseteq p'$ and $p \neq p'$. We denote by $urlMAX(P)$ the set of maximal URL patterns with respect to P.* ■

Based on all the previous discussion and Theorems 1 and 2, we have the following result immediately.

Theorem 3. *Let $P = \{p_1, \ldots, p_n\}$ ($n > 1$) be a set of URL patterns. $urlMAX(\{p_1, p_2\}) = \{(h, d, f) | h \in MAX(h_1, h_2), d \in seqMAX(d_1, d_2), f \in MAX(f_1, f_2)\}$. When $n > 2$, $urlMAX(P) = urlMAX(urlMAX(P - \{p_n\}) \cup p_n)$.* ■

Chapter 4

Mining Patterns

In Chapter 3, we discuss how to extract patterns as features from a set of URLs generated by a common malicious mechanism. Given a set of malicious URLs that are generated by different mechanisms, how can we extract meaningful patterns for detection? One fundamental challenge is that URL patterns are generated from URLs, and cannot be assembled using any pre-defined elements, such as a given set of items in the conventional frequent pattern mining model. Thus, all existing frequent pattern mining methods cannot be used.

In this chapter, we develop new methods to mine URL patterns. We start with a baseline method that finds all URL patterns. Then, we present a heuristic method that finds some patterns that are practically effective and efficient.

4.1 The Complete Pattern Set Algorithm (CA)

As discussed, a URL pattern can be generated by a set of URLs. Given a set of URLs, the complete set of URL patterns are the maximal URL patterns from different subsets of the URLs. To search the complete set of URL patterns systematically, we can conduct a bottom-up search, as illustrated in Figure 4.1. We first compute the maximal URL patterns on every pair of malicious URLs, and add the generated valid URL patterns into a result pattern set. Then, we further compute the maximal patterns on the resulting URL patterns, and repeat this process until we can not generate new URL patterns. The pseudocode is shown in Algorithm 1, and the algorithm is called the **complete pattern set algorithm (CA)**.

Depending on different situations, the definition of “valid patterns” may be different. In

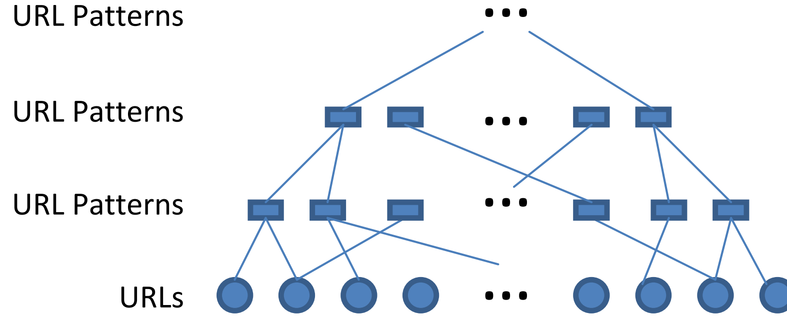


Figure 4.1: Bottom-up search of URL patterns

our system, the patterns contain only "*" (empty) or only a full domain name are invalid, because the former are too general, and the latter appear only in one malicious domain, and are ineffective in detecting unseen URLs. Some other heuristics can also be applied. For example, pattern contains only a file type like ".jpg" should be removed.

We formally justify the correctness of the algorithm.

Theorem 4. *Algorithm 1 outputs the complete set of maximal URL patterns.*

Proof. Apparently, every pattern output by Algorithm 1 is a maximal pattern on some subsets of P. Let p be a valid maximal URL pattern generated from a subset $P' \subseteq P$. We only need to show that Algorithm 1 outputs p. Since p is valid, every patterns p' such that $p \sqsupseteq p'$ must also be valid.

We show that a valid pattern generated from a subset of l URLs will be included into A in the algorithm no later than the $(l-1)$-th iteration by mathematical induction.

(The basis step) The first iteration generates all valid patterns on every pair of URLs.

(The inductive step) Assume that all valid patterns generated by subsets of k URLs are included into P in the algorithm in no later than the $(k-1)$-th iteration. Consider a valid pattern p that is generated by a subset of $(k+1)$ URLs $P' = \{r_1, \ldots, r_k, r_{k+1}\}$. Clearly, the patterns p_1 generated from $\{r_1, \ldots, r_k\}$ and p_2 generated from $\{r_2, \ldots, r_{k+1}\}$ are generated in the same iteration. If pattern p has not been generated using p_1 and p_2 in a previous iteration, it is generated in the k-th iteration. ∎

The complexity of the CA algorithm is exponential to the number of input malicious

Algorithm 1 Complete Pattern Set Algorithm (CA)

```
P ← malicous URLs set;                              ▷ P is the training URL pattern set
A ← ∅;                                              ▷ A is the result URL pattern set
repeat
    R ← ∅;                                          ▷ R is a intermediate URL pattern set
    for all pattern p ∈ P do
        for all pattern p′ ∈ P − {p} do
            pattern set Q ← urlMAX({p, p′});
            for all pattern q ∈ Q do
                if q is valid and q ∉ A then
                    R ← R ∪ {q}, A ← A ∪ {q};
                end if
            end for
        end for
    end for
    P ← R;
until R = ∅;
return A;
```

URLs.

4.2 Quality of URL Patterns

Many URL patterns are generated from a large training data set of malicious URLs. Thus, we need a method to assess the quality of the URL patterns and select the effective ones. Based on the assumption that an effective pattern in malicious URL detection should appear frequently in different malicious domains and be infrequent in the white websites, we use two criteria: the malicious frequency and white frequency to estimate the quality of it. Please note that our white data set still may contain malicious URLs.

For a URL pattern p, the **malicious frequency** of p is the number of unique domains it covers in the malicious training data, and the **white frequency** of p is the number of unique domains it covers in the white data set. Later we will use $Sc(p)$ to present the quality score of a URL pattern p. In this thesis, we use malicious probability ratio as quality score.

Definition 7 (Malicious probability ratio)**.** *For a pattern p, the* **malicious probability** *of p is $\frac{p\text{'s malicious frequency}}{n_b}$, where n_b is the number of unique malicious domains in the malicious data set. The* **white probability** *of p is $\frac{p\text{'s white frequency}}{n_w}$, where n_w is the number of unique*

white domains in the white data set. If p's white frequency is 0, we assign the probability of it a small number. In this thesis, we use $0.1/n_w$.

The **malicious probability ratio** *of* p *is* $\frac{p\text{'s malicious probability}}{p\text{'s white probability}}$. ■

4.3 The Greedy Selection Algorithm (GA)

In real applications, the size of training data (malicious URLs) can be huge. The CA algorithm can be very costly. In the CA algorithm, we compare every malicious URL with all other URLs to extract all patterns. Consequently, one URL may be covered by many URL patterns. Although those patterns are different, they may heavily overlap with each other. In other words, there may be many redundant features in the result URL pattern set.

In our real data sets, the major features of a URL can be captured well after a small number of patterns are extracted. Moreover, one malicious URL generation mechanism is often represented by a non-trivial number of URLs in the training data set. Thus, we develop a greedy algorithm. At each stage, if a URL generates a valid URL pattern, then we remove the URL. We intersect two URLs (or URL patterns) only if the quality of resulting URL patterns increase. This algorithm is called the **greedy selection algorithm(GA)**, as shown in Algorithm 2.

4.4 Indexing

The space of malicious URLs is very sparse. Most of the URLs do not share any common features. Thus, we can build an index on the URLs to facilitate the retrieval of similar URLs so that meaningful URL patterns can be mined.

In the system, we build an inverted index IL_{file} on the trigrams of file names in malicious URLs. Similarly, we also build an inverted index IL_{dir} on the trigrams of directory names, and an inverted index IL_{domain} on the trigrams of domain names. We also build the corresponding inverted lists on white URL data, including IL_{wfile}, IL_{wdir}, and $IL_{wdomains}$. To avoid many meaningless trigrams such as "jpg", "com", "htm", and "www", we remove the top 2% most frequent trigrams of white data set from the malicious trigram inverted indexes. This step is similar to the well recognized stop-word removal in information retrieval. After building the indexes, when running our algorithms, we only need to compute

Algorithm 2 Greedy Pattern Selection Algorithm

```
P ← malicous URLs set;                                   ▷ P is the training URL pattern set
A ← ∅;                                                   ▷ A is the result URL pattern set
repeat
    R ← ∅;                                               ▷ R is a intermediate URL pattern set
    for all pattern p ∈ P do
        for all pattern p′ ∈ P − {p} do
            pattern set Q ← urlMAX({p, p′});
            for all pattern q ∈ Q do
                if q is valid and q ∉ A and Sc(q) > Max(Sc(p), Sc(p′)) then
                                                         ▷ Sc(q) is the quality score of pattern q
                    R ← R ∪ {q}, A ← A ∪ {q};
                    P ← P − {p, p′};
                end if
            end for
        end for
    end for
    P ← R;
until R = ∅;
return A;
```

the common pattern between those URLs sharing at least one trigram.

Chapter 5

Experimental Results

In this chapter, we report an extensive empirical study to evaluate our approach.

5.1 Data Sets

We use the real data from Fortinet. In Forinet, the rating of a URL is a category, such as malware, phishing, or sports, there are 79 such categories so far (`http://www.fortiguard.com/webfiltering/webfiltering.html`).

We use two data sets in training: a malicious data set where we extract the malicious URL patterns, and a white data set that is used to measure the quality of result patterns. The malicious data set contains 1.14 million URLs that are rated as malware, spyware or phishing from the Fortinet's log files in September and October in 2011. Table 5.1 summarizes the statistics of unique domain names, directory names and file names in the malicious data set. The most frequent domain name is "`www.tns-counter.ru`", whose hit count is 191, 983. The most frequent directory name is "`ru`" with a hit count of 185, 286. The most frequent file name is "`favicon.ico`", which appears 6, 282 times. On average, each URL consists of 4.1 URL segments.

For the white URL data set, there are 1.45 million URLs, not labeled as malicious by the existing detection method. Those URLs are randomly sampled from the Fortinet's log files in September and October in 2011. In the sampling process, in order to gather more diverse URL samples, we remove the URLs from the top 100 popular websites (`http://toolbar.netcraft.com/stats/topsites`), because those URLs contribute most of the counts in the log files. Table 5.2 shows the statistics of unique domain names, directory names and file

	Unique counts
File names	0.8 million
Directory names	0.16 million
Domain names	0.06 million
URLs	1.14 million

Table 5.1: Some statistics of the malicious data set

names in the white data set. The most frequent domain name is "`profile.ak.fbcdn.net`", whose hit count is 52,284. The most frequent directory name is "`images`" with a hit count of 174,299. The most frequent file name is "`default.jpg`", which appears 11,892 times. One URL in the white data set on average has 4.4 segments.

	Unique counts
File names	1.13 million
Directory names	0.52 million
Domain names	0.24 million
URLs	1.45 million

Table 5.2: Some statistics of the white data set

To avoid some meaningless patterns from being extracted, we remove the top 2% most frequent directory names and file names in the white URL data set from the malicious data set. Some examples of these "stop words" are "`default.jpg`", "`index.html`", and "`images`".

Every day, there are a few billions of URLs be verified by Fortinet, most of them (over 95%) can be rated properly into some categories, but there is still a small percentage of URLs are unrated. We sampled 3 million unrated URLs in Fortinet's log files from September and October. Although these 3 million URLs are unrated in September and October 2011, some of them can be rated now since the blacklist is updated.

We obtain our test data as follows. We first use the Fortinet URL look up tool to rate those 3 million URLs. As a result, 10,234 of them are rated as malicious, 606,273 are rated as benign, 1,828,060 are still unrated, and the rest of them are rated as porn, illegal, unethical or other categories that "between malicious and white". We remove those "unclear" URLs because they sometimes may be rated as malicious and sometimes may not

be, depending on the categorization method. Finally, we got a test data set consisting of $10,234$ malicious URLs and $606,273$ white URLs.

5.2 Effectiveness of URL Patterns

After running the GA algorithm on the malicious data set, we get a set of $78,873$ URL patterns. We first use all these URL patterns to do detection in the test data set. As a result, there are 45,538 URL patterns that cover at least one URL in the test data set, 4,574 of them cover only malicious URLs, 27,820 patterns only cover white URLs, and the rest of them cover both malicious URLs and white URLs. Those 45,538 URL patterns cover 527,335 URLs in the test data set (8,771 malicious). The 4,574 URL patterns that only cover malicious URLs cover 2,482 malicious ones. We can not use all those URL patterns in detection efficiently, since there are many noise patterns among them, and the precision is low.

How can we select the right URL patterns? We use the malicious probability ratio as defined in chapter 4. We first rank those URL patterns by malicious probability ratio. Figure 5.1 shows the distribution of URL patterns with respect to different ratio, as we can see, most patterns' ratio is between 0.1 to 100. Interestingly, there is a gap around ratio 1000. Later we will explain that the patterns on the right of this gap have a high precision in detecting malicious URLs.

The left of Table 3 shows some of the top ranking patterns found by our algorithm. These top patterns capture rich features, including segment patterns of domains, full directory names, full file names, and file name segment patterns. The right of Table 3 lists some of the bottom ranking patterns. These patterns rank low because they are too general, thus can not distinguish malicious against normal URLs well.

5.3 Effectiveness of Malicious Probability Ratio Threshold

In this section, a malicious probability ratio threshold is used to choose patterns. We show how this ratio threshold affects the performance of detection. To measure the performance of detection, we use three criteria: precision, recall and $F1$.

The result is shown in Figure 5.2(a). As the ratio threshold increases, the precision increases quickly because more and more low-quality URL patterns are filtered out. At the

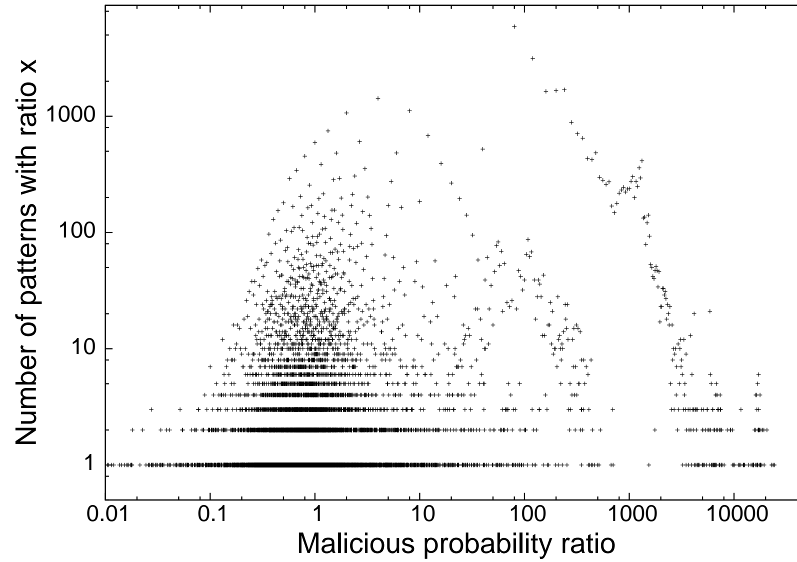


Figure 5.1: Malicious probability ratio distribution of URL patterns

Top Ranking Patterns	Bottom Ranking Patterns
`*07db*.com/*`	...
`*survery*/offers/*`	`*download*.com/*`
`*/*dmVyPT*hawQ9*`	`*/music`
`*/lmg/am1.rar`	`*/Common/*`
`*/d/*2h1o`	`*/2.0/*`
`*/finance-journal/*`	`*/feeds/*`
`*.my2/ut.php?m=`	`*motor*.com/*`
`*/esubmit/*bizopp_main*`	`*/blogs/*`
`*.loxblog*.com/*`	`*ding/*`
`*/d/mli2n1277`	`*/150/*`
`*807db*.com/files4/*`	`*/52.jpg`
`*onlinewebshop*.net/*`	`*/br/*`
...	`*/gp/*`

Table 5.3: Some top ranking and bottom ranking malicious URL patterns

Top patterns	Number of matched URLs
`*/*dmVyPT*hawQ9*`	95
`*/*ZiaWQ9*mc2lK*`	66
`*/finance-journal/*`	63
`*/worms.jar`	26
`*/*OfferImageo*`	23
`*survery*/offers/*`	17
...	...

Table 5.4: Some malicious URL patterns

same time, the recall also decreases quickly. The reason is that the number of patterns used in detection decreases dramatically, as shown in Figure 5.2(b).

To look deeply into the pattern detection performance, we conduct a detailed analysis at the point where ratio threshold is set to 800. There are 5,716 out of 78,873 patterns used in the detection. Among those 5,716 patterns, 1,951 detect at least one truely malicious URLs from the test data set.

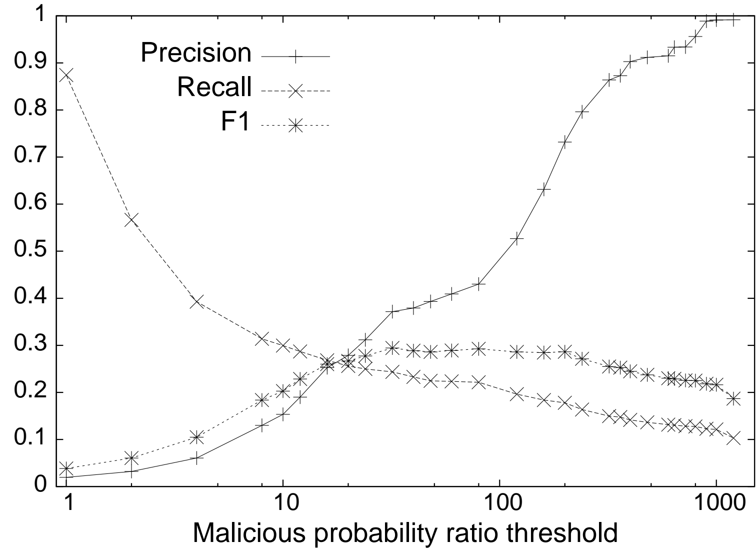
The total number of URLs covered by these patterns is 1,365, among them, 1,305 is indeed malicious URLs. The precision is 0.95. In Figure 5.3, we show the number of patterns that cover different numbers of malicious URLs. Most URL patterns only detect very few URLs, and a small percentage of patterns detect most of the malicious URLs. On average, each URL pattern covers 3.96 URLs, each URL in the test data set is covered by 17.39 patterns, which indicates that there are still some overlaps in the pattern set produced by the GA algorithm.

In Table 5.4, we list some URL patterns that detect malicious URLs, and some of these malicious URLs are shown in Table 5.5. We can clearly see some mechanisms behind the malicious URLs, and our URL patterns indeed capture some effective features of these mechanisms.

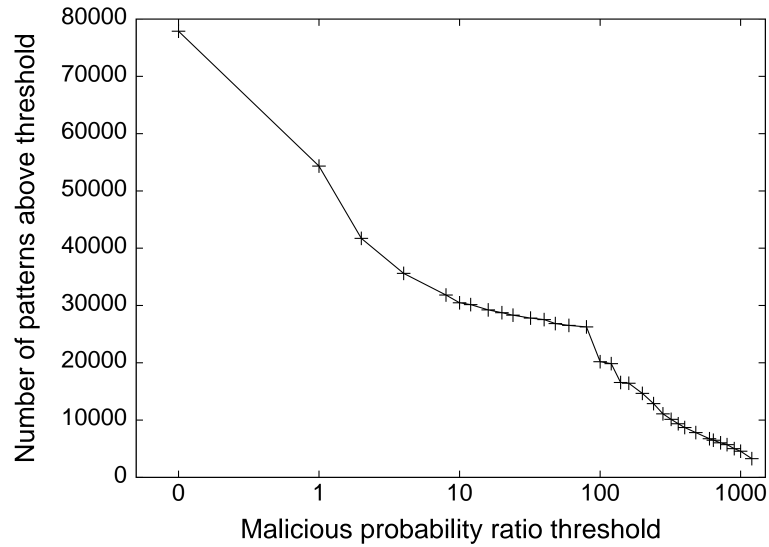
5.4 The CA Algorithm and the GA Algorithm

In this section, we compare the runtime and detection performance of the CA and GA algorihtm.

Figure 5.4(a) compares the runtime of the CA and GA algorithm with respect to different



(a) Detection performance



(b) Number of patterns above threshold

Figure 5.2: Detection under different malicious probability ratio thresholds

Malicious URLs detected in test data set
...
1rbyyj3swwmvlkry3.com/6AG0m63dmVyPTQuMiZiaWQ9ZkZjQ···GVmOWQ9NjY2N=18Z
all1incks--4this.com/yab3rb5p7p7m2Lo2dmVyPTQuMCZia···MDAmc2lECR0J
crozybanner.com/ck12tS7l8p4Mtt07dmVyPTMuOTYmYmlkPW5v···cpdmVycw==16A
g00o00gle.com/1aq0DX1E8b6xLRS1dmVyPTQuMCZiaWQ9MC.9MzAw···CZlbmc9d3N.LTQsN
...
aeybgxpzbnhyjt.cu.cc/content/worms.jar
wqtszgiqfnnlob.cu.cc/worms.jar
129.121.66.16/home/content/worms.jar
...
save.lmyn312oincidence.com/OfferImageo5829ojpg
sjiu112oawhile.com/OfferImageo5243ogif
cmad232ohuman.com/OfferImageo4256opng
bwgp11o.com/OfferImageo4910gif
...
365-job.net/finance-journal/hpproof.html
business9journal.net/finance-journal/images/gplex.jpg
work15home.net/finance-journal/
home22solutions.com/finance-journal/images/banner.jpg
...
easysurveyguide.com/media/offers/16146_2.jpg
onlinesurveyguide.com/media/offers/15342_2.jpg
funsurveyonline.com/offers/15893_2.jpg
supersurverysite.com/offers/e2r044823
...
internetquizsite.com/d/w2h1o80792
quizfindtraffic.com/d/w2h1o80792
quizfungam.com/d/gfx/902.gif
quiztakeonline.com/d/s1b8il7482
...

Table 5.5: Some malicious URLs covered by malicious URL patterns

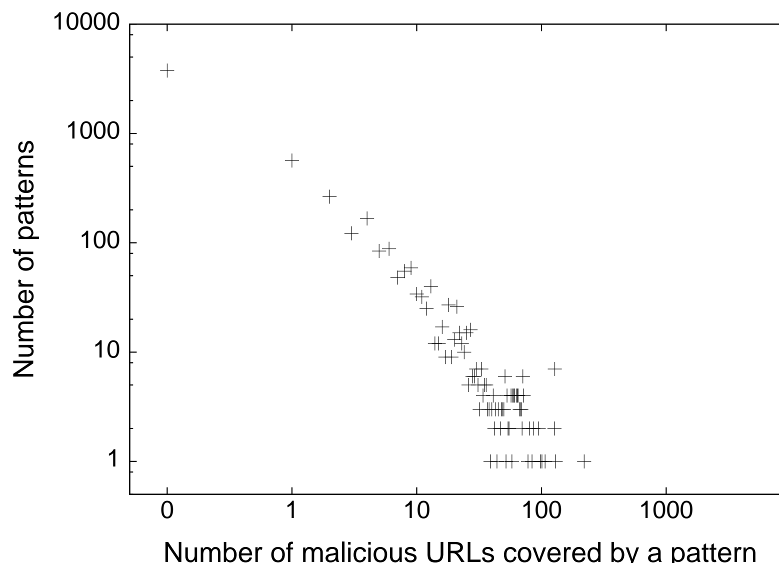
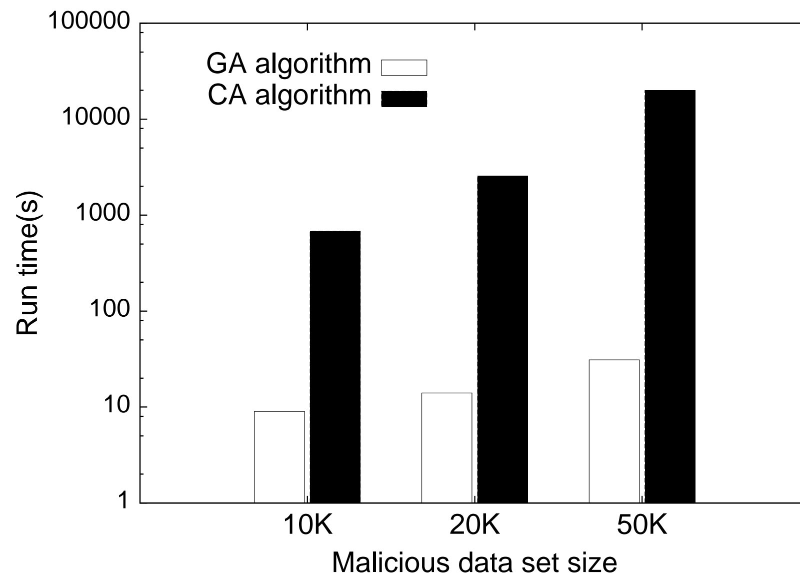


Figure 5.3: Malicious URL hit count distribution

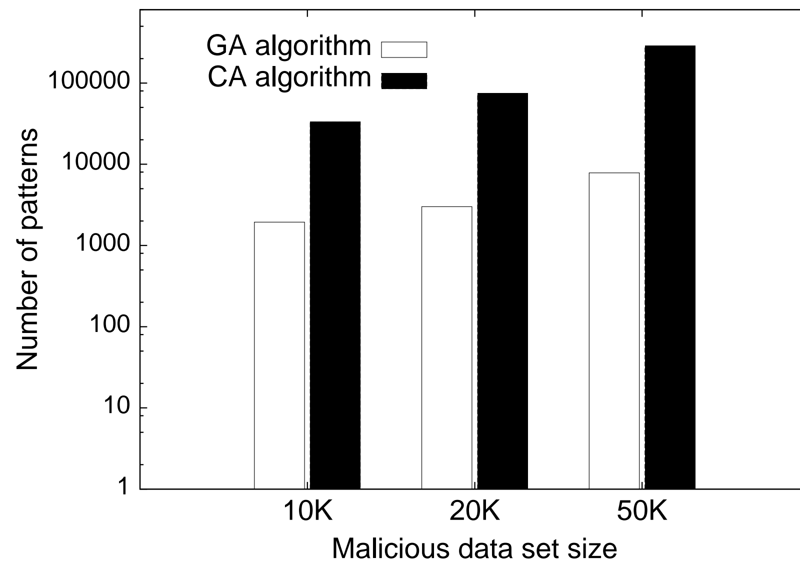
training data set size. Figure 5.4(b) shows the number of URL patterns found. Apparently, the CA algorithm takes much more time and generates much more URL patterns than the GA algorithm. How do the patterns generated by the CA and GA algorithm, respectively, perform in detection?

In Figure 5.5, we compare the detection performance of the CA and GA algorithm by setting the training data set size to 10K. Although the CA algorithm generates much more URL patterns, the performance is not much better than the GA algorithm. The reason is that the patterns generated by the CA algorithm have heavy overlaps with each other. Many URL patterns actually point to the same feature, and most patterns cannot detect any malicious URLs. For example, when the ratio threshold is 800, the GA algorithm generates 915 patterns over the threshold, and each pattern on average covers 28 malicious URLs in the test data set, while the CA algorithm generates 17,438 patterns over the threshold, but each pattern covers only 3 malicious URLs in the test data set.

In summary, the GA algorithm uses much less time and still captures important features from the training data.



(a) Run time



(b) Number of generated patterns

Figure 5.4: Comparison between the CA and GA algorithm in different malicious data sets

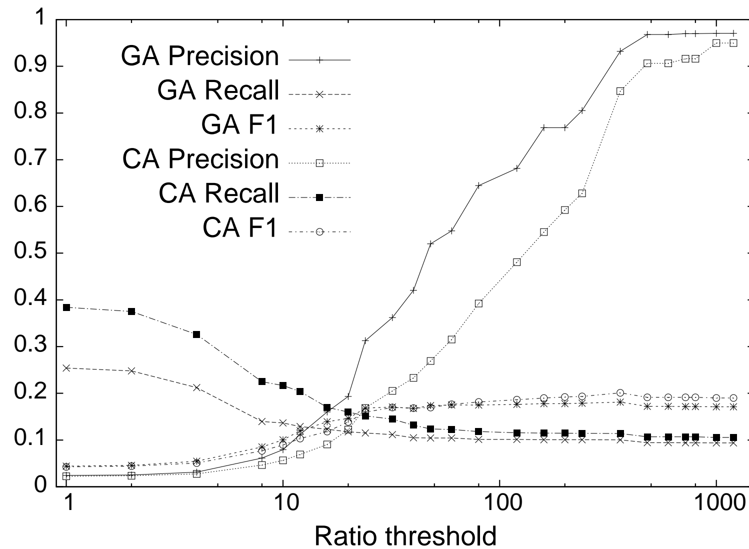


Figure 5.5: Comparison between the CA and GA algorithm

5.5 Effect of Training Data Set Size

As mentioned, we have two training data sets: a malicious URL data set to extract URL patterns, and a white URL data set to rate and select patterns. In this section, we test how the training data set size can affect the detection performance.

5.5.1 White training dataset

Figure 5.6 shows the precision using different sizes of white training data sets, where the size of malicious data set is fixed to 1000K. The detection precision using a small white training data set is much lower than that of using a larger data set. When the white data set is small, many noise URL patterns cannot be filtered out. Therefore, a large white URLs training data set is critical in selecting effective patterns for detection.

5.5.2 Malicious training dataset

To verify the efficiency of the GA algorithm, we run it on malicious training data sets of different size. Figure 5.7 shows the runtime and the number of URL patterns generated with respect to the size of malicious training data set. Due to the cost in dynamic programming, the runtime of the GA algorithm increases substantially as the malicious training data set

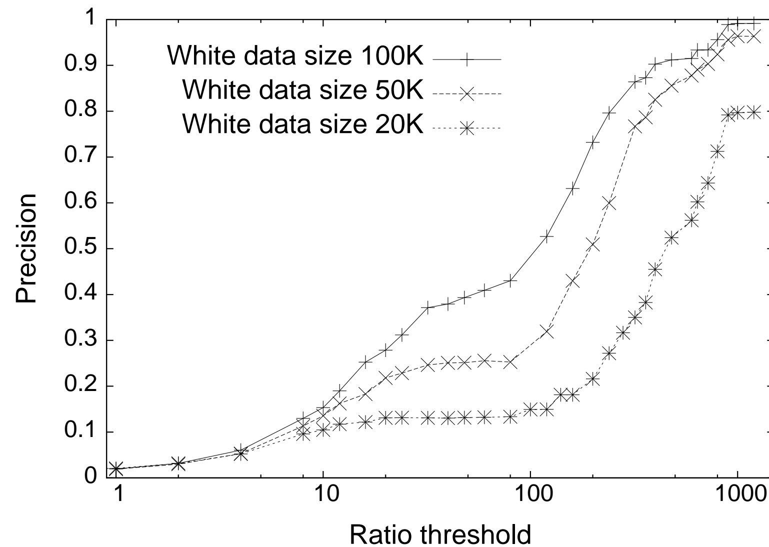


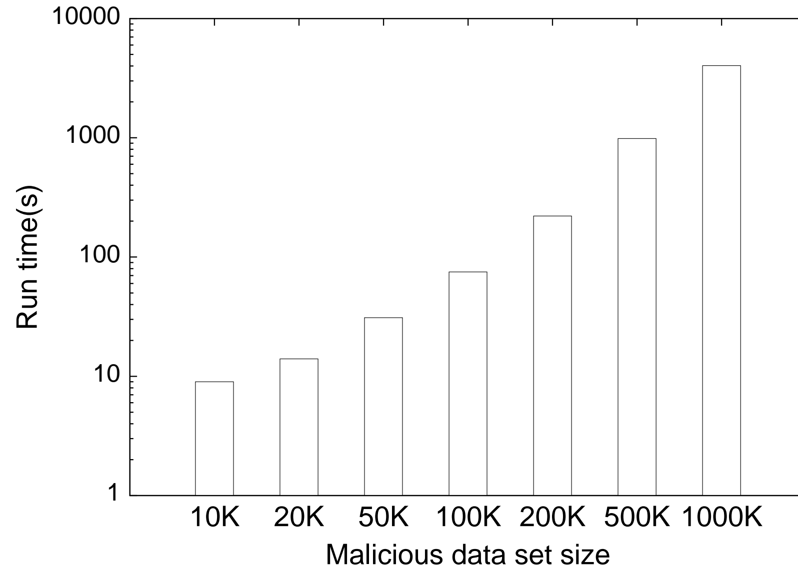
Figure 5.6: The GA algorithm in different white training data sets

becomes large.

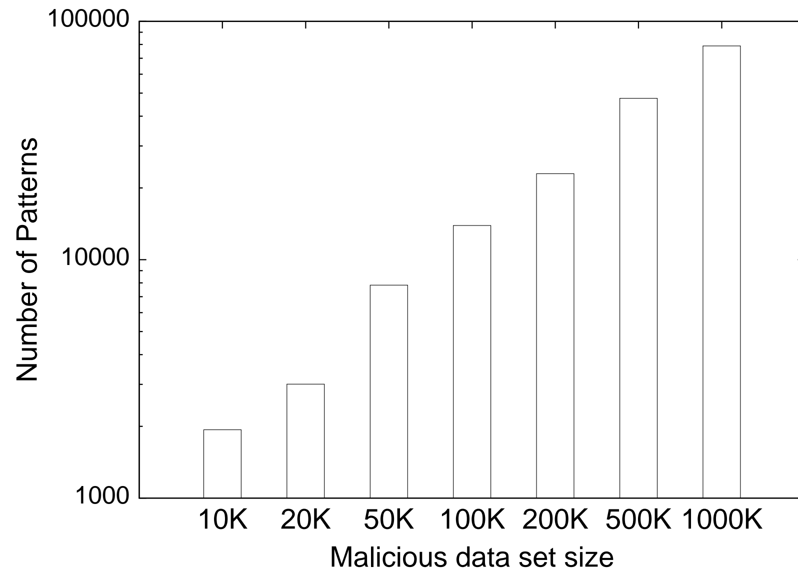
To understand the performance of detection, we show the ROC curves of the GA algorithm using different malicious training data sets in Figure 5.8. When the malicious training set size increases, we can get a higher true positive rate with a stable false positive rate. That means when we use more malicious URLs, we can detect more malicious URLs from the test data set without sacrificing the precision.

5.6 Effect of Dynamic Feature Extraction

As mentioned in chapter 3, our algorithm uses an automatic lexical feature extraction method instead of using pre-defined delimiters. In this section we test how these lexical features work in detection. In Figure 5.9, we compare the GA algorithm and a variation of the GA algorithm where we only use the features extracted by using "/" as the delimiter. Without dynamic feature extraction, the precision is a little higher. That is because more general patterns using dynamic features may mismatch some URLs. However, we get a much higher recall when using dynamically extracted features. Those features indeed help to detect much more malicious URLs.



(a) Runtime



(b) Number of generated patterns

Figure 5.7: Runtime and number of patterns in different malicious training data sets

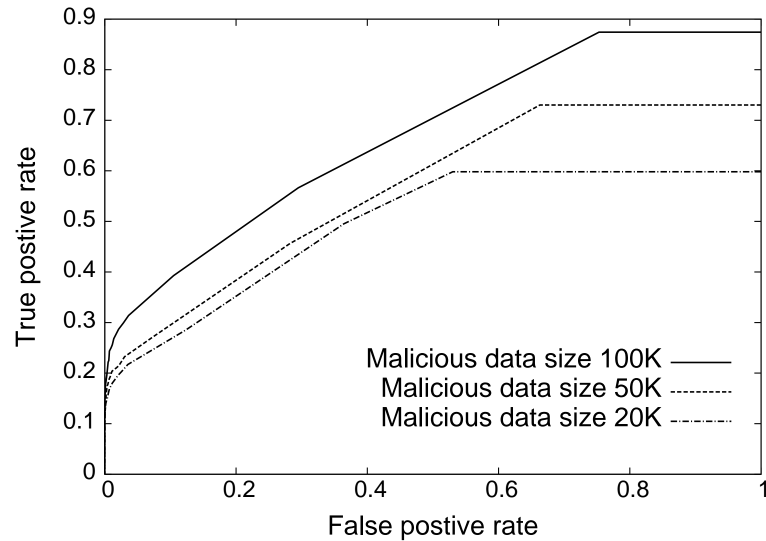


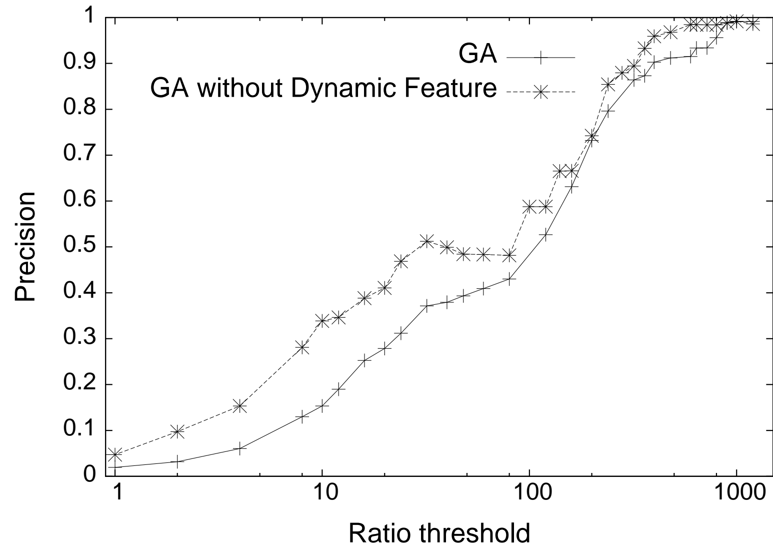
Figure 5.8: ROC curve of the GA algorithm in different malicious training data sets

5.7 Comparison with Machine Learning Methods

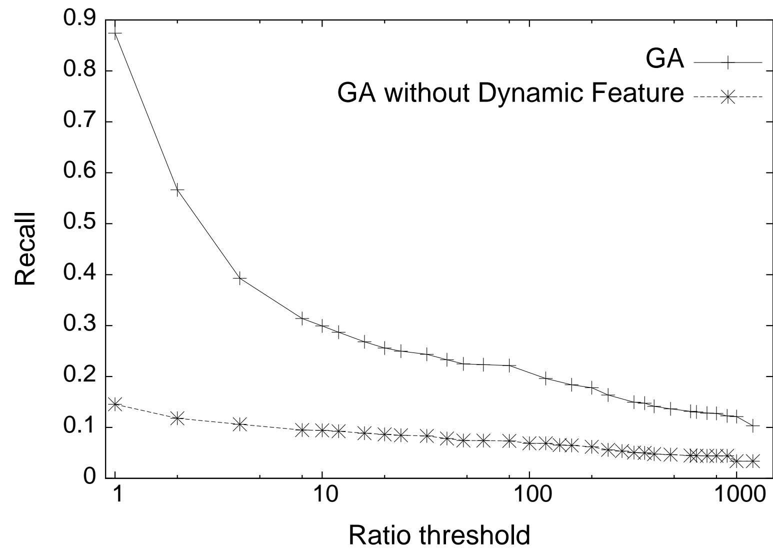
Chapter 2 reviews some machine learning based methods [8, 7] for classification on URLs. Without the original implementations, we use WEKA [14], an open source software containing a collection of machine learning algorithms for data mining tasks, to compare with our approach.

For the lexical feature extraction, we implement in the same way as [8, 7], that is, we treat each token (strings delimited by '/', '?', '.', '-', '=', and '_') in URLs as a binary feature. Then, we use two classification models in WEKA: Naïve Bayes and SMO [12, 6], a support vector classifier. The training data set consists of 50K malicious URLs and 200K white URLs. There are 592,829 binary lexical features in this training data set. For testing, we use a data set containing 1,028 malicious URLs and 60,680 white URLs. As for the GA algorithm, we set the malicious probability ratio threshold to 800.

Figure 5.10 shows the result. The GA algorithm takes much less training and testing time, and still gets good detection performance. We notice that the training and testing time of Naïve Bayes and SMO are extremely high. The possible reason is that the feature vectors are too long, about 592,829 in the experiments, and it takes too much time to do preprocessing. Although the SMO algorithm gains a higher recall than the GA algorithm, it sacrifices much on precision. In real applications, a high precision is critical for malicious

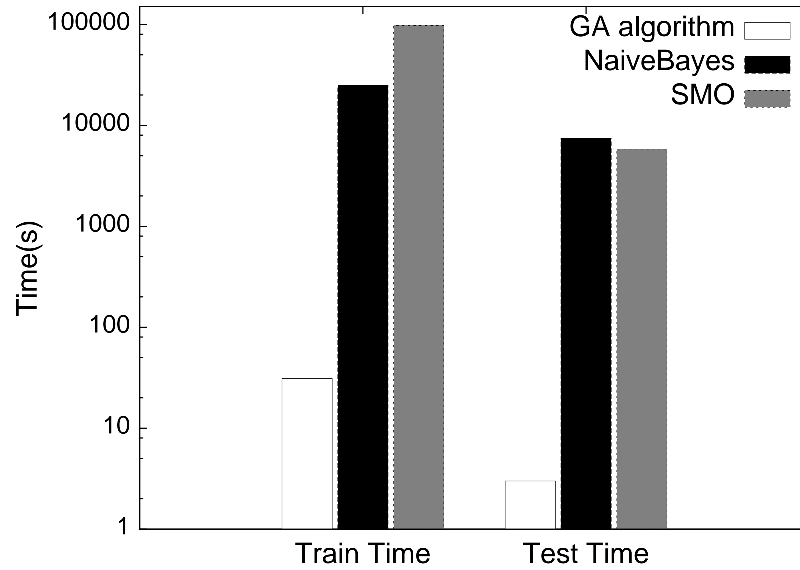


(a) Precision

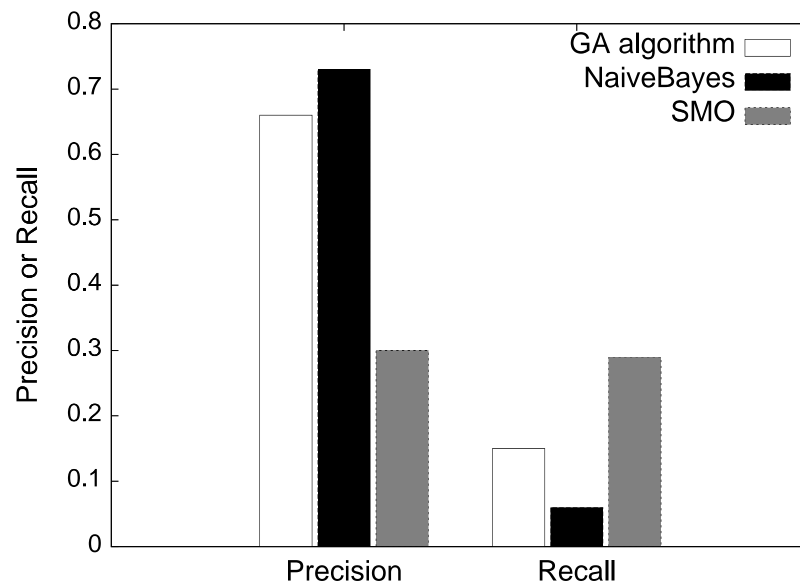


(b) Recall

Figure 5.9: Algorithms with and without dynamic features



(a) Run time



(b) Performance

Figure 5.10: Comparison between the GA algorithm and WEKA machine learning methods

URL detection because most URLs are not malicious in real data flow. Our algorithm captures more lexical features than the pre-defined delimiters method in the Naïve Bayes model, and has a higher recall.

The recall of all those three algorithms are low in our experiments, while [8, 7] reports a recall above 0.8. The possible reasons are as follows. First, in [8, 7], the training data and test data are just different splits of the same data source, but in our experiments, the test data is from the unknown URLs set and the training data is from the rated URLs set. Thus, the training data and test data in our case may follow very different distribution. Many malicious URLs in the test data set do not contain any malicious lexical features available in the training data set. Our test data is closer to reality because in the real applications, the model should be able to detect malicious URLs from unknown URLs. Second, the data in [8, 7] is typical malicious URLs and white URLs, while our URLs are from the real daily URL flows, where the data is much sparser.

In summary, comparing to the machine learning approaches, our algorithm is more practical and more efficient, and the detection performance is competitive.

Chapter 6

Conclusions

In this thesis, we tackle the problem of malicious URL detection and make two new contributions beyond the state-of-the-art methods. We propose to dynamically extract lexical patterns from URLs, which can provide new flexibility and capability on capturing malicious URLs algorithmically generated by malicious programs. We develop a new method to mine our novel URL patterns, which are not assembled using any pre-defined items and thus cannot be mined using any existing frequent pattern mining methods. Our extensive empirical study using the real data sets from Fortinet clearly shows the effectiveness and efficiency of our approach. The data sets are at least two orders of magnitudes larger than those reported in literature.

As future work, there are following directions we can further work on.

- Better methods to select patterns for detection are needed, because the malicious probability ratio threshold method used in our experiments may filter out not only noise patterns but also some useful patterns. More sophisticated methods to assess the quality of a URL pattern can be applied. For example, some heuristic laws can be used in the assessment.

- Our algorithm may generate too many URL patterns when the training data is huge. As we can see from the experiments, even in GA algorithm, the URL patterns still overlap with each other. In other words, there may be many redundant features in the result URL pattern set. How to remove those overlaps is a problem need to be solved.

- How to further increase the recall of malicious detection is also critical. To increase the recall, more lexical features beyond the common substrings can be used in the detection, such as the statistics of segments, and the meaning of segments (e.g., numerical or letter).

- All our algorithms in this thesis are batch based algorithms, but in the real application, the training data is updated everyday. Online algorithms can be developed to handle the evolving features over time.

Bibliography

[1] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, pages 39–, Washington, DC, USA, 2000. IEEE Computer Society.

[2] Gartner. Gartner survey shows phishing attacks escalated in 2007; more than $3 billion lost to these attacks. `http://www.gartner.com/it/page.jsp?id=565125`.

[3] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15:55–86, August 2007.

[4] ICT Applications and Cybersecurity Division, Policies and Strategies Department, and ITU Telecommunication Development Sector. ITU study on the financial aspects of network security: Malware and spam. `http://www.itu.int/ITU-D/cyb/cybersecurity/docs/itu-study-financial-aspects-of-malware-and-spam.pdf`.

[5] Sittichai Jiampojamarn and Nick Cercone. Biological named entity recognition using n-grams and classification methods. In *In Proceedings of PACLING*, pages 191–8506, 2005.

[6] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Comput.*, 13(3):637–649, March 2001.

[7] Anh Le, Athina Markopoulou, and Michalis Faloutsos. Phishdef: Url names say it all. In *Proceedings of the 30th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies*, pages 191–195, Shanghai, China, April 2011. IEEE.

[8] Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 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*, KDD '09, pages 1245–1254, New York, NY, USA, 2009. ACM.

[9] David Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, April 1978.

[10] D. Kevin McGrath and Minaxi Gupta. Behind phishing: an examination of phisher modi operandi. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 4:1–4:8, Berkeley, CA, USA, 2008. USENIX Association.

[11] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A crawler-based study of spyware in the web. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'06)*, San Diego, California, USA, 2006. The Internet Society.

[12] John C. Platt. Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[13] Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, and Fabian Monrose. All your iframes point to us. In *Proceedings of the 17th conference on Security symposium*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[14] Weka. Weka. `http://www.cs.waikato.ac.nz/ml/weka/`.

[15] Wikipedia. Web threat. `http://en.wikipedia.org/wiki/Web_threat`.

[16] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A.L. Narasimha Reddy, and Supranamaya Ranjan. Detecting algorithmically generated malicious domain names. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 48–61, New York, NY, USA, 2010. ACM.