2002

Design and implementation of a content filtering firewall

Rongbo Du

University of Wollongong, rongbo@uow.edu.au

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author.

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Design and Implementation of A Content Filtering Firewall

A thesis submitted in partial fulfilment of the requirements for the award of the degree

Master of Computer Science (Honours)

from

UNIVERSITY OF WOLLONGONG

by

Rongbo Du

School of Information Technology and Computer Science
2002
Acknowledgement

I am grateful to my principal supervisor Professor Rei Safavi-Naini and co-supervisor Dr. Willy Susilo for their constant guidance and support.
Abstract

Firewall is a component or a set of components that restricts access between a protected network and the Internet, or between two networks. A traditional firewall tries to protect the internal network from outside threats by permitting or forbidding network connections between the external network and the internal network. The rules used by the firewall to determine whether a connection should be permitted or not are usually based on the connection type, source address or destination address of the connection, or user authentication, and not on the actual content of the network traffic.

Content filtering is to monitor and analyse data in order to filter specific content that is forbidden by an organisation’s policy. Previous content filtering approaches include: use blacklists and whitelists to keep lists of sites that should be blocked or allowed, search the content for keywords that frequently appear in undesired sites and block the content if such keywords are found, or utilise rating systems that provide rating of sites. Most previous Internet content filtering programs reside and run on the end users’ machines, and not on a central point of the network such as firewall. Therefore, such content filtering could be circumvented or disabled by dishonest end users. In addition to that, maintaining or updating of such programs need to be done for each end user’s terminal machines individually.

The purpose of this thesis is to investigate, propose and experiment how to combine the content filtering and firewall together, in order to solve problems that cannot be solved either by content filtering or by firewall individually.

We propose a method of adding content filtering functionality to the firewall and describe its implementation. We also propose and implement an advanced content filtering method based on text categorisation techniques to replace the basic keyword-matching filtering method. We discuss using content filtering firewall to prevent computer virus propagation through the Internet. Then we propose and describe the implementation of a new attack using encryption to get around the content filtering firewall, hence showing the difficulty the content filtering firewall encounters when end-to-end encryption for network traffic is used.
Related Publication

CERTIFICATION

I, Rongbo Du, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Master of Science (Honours), in the School of Information Technology and Computer Science, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Rongbo Du

March 2002
# Table of Contents

ACKNOWLEDGEMENT ........................................................................................................... II

ABSTRACT ................................................................................................................................. III

RELATED PUBLICATION ........................................................................................................ IV

TABLE OF CONTENTS .............................................................................................................. VI

LIST OF TABLES ....................................................................................................................... X

LIST OF FIGURES .................................................................................................................... XI

CHAPTER 1 OVERVIEW OF THE THESIS ............................................................................. 1
  1.1 OBJECTIVES .................................................................................................................. 1
  1.2 OUTLINE ......................................................................................................................... 1

CHAPTER 2 INTRODUCTION TO CONTENT FILTERING .................................................. 3
  2.1 NEEDS FOR CONTENT FILTERING ............................................................................... 3
  2.2 PREVIOUS WORKS ON CONTENT FILTERING AND THEIR LIMITATIONS ............... 5
    2.2.1 Blacklists and Whitelists ...................................................................................... 5
    2.2.2 Keyword Blocking ............................................................................................... 5
    2.2.3 Rating Systems ................................................................................................... 6
  2.3 CURRENT CONTENT FILTERING SOFTWARE .............................................................. 6

CHAPTER 3 INTRODUCTION TO FIREWALL ..................................................................... 8
  3.1 NEEDS FOR FIREWALL .................................................................................................. 8
    3.1.1 Protecting Internal Networks ............................................................................. 8
    3.1.2 Enforcing Internal Security Policy ..................................................................... 10
    3.1.3 Use of Internal Firewall ..................................................................................... 10
  3.2 TYPES OF FIREWALLS .................................................................................................. 11
    3.2.1 Packet-Filtering Firewall .................................................................................. 11
    3.2.2 Application-Level Firewall ............................................................................... 12
    3.2.3 Circuit-Level Firewall ........................................................................................ 13
  3.3 FIREWALL ARCHITECTURES ......................................................................................... 13
    3.3.1 Dual-Homed Host Architecture ....................................................................... 13
    3.3.2 Screened Host Architecture .............................................................................. 14
3.3.3 Screened Subnet Architecture ......................................................... 15
3.4 LIMITATIONS OF TRADITIONAL FIREWALL ......................................... 16

CHAPTER 4 DESIGN OF A CONTENT FILTERING FIREWALL ....................... 17

4.1 COMPARISONS OF FILTERING AT USERS’ TERMINAL MACHINES AND FILTERING AT A CENTRAL FIREWALL ........................................... 17
  4.1.1 Filtering at Users’ Terminal Machines ........................................... 17
  4.1.2 Filtering at Firewall ................................................................. 18
4.2 EXTENDING A FIREWALL FOR CONTENT FILTERING FUNCTIONALITY .......... 19
  4.2.1 Filtering for Text Files .............................................................. 20
  4.2.2 Filtering for Non-text Files ...................................................... 21

CHAPTER 5 AN ADVANCED CONTENT FILTERING METHOD ....................... 23

5.1 NEED FOR ADVANCED FILTERING ..................................................... 23
5.2 INTRODUCTION TO AUTOMATIC TEXT CATEGORISATION .................... 24
  5.2.1 Observations ............................................................... 24
5.3 MAJOR AUTOMATIC TEXT CATEGORISATION APPROACHES ................. 26
  5.3.1 Naïve Bayes (NB) ............................................................... 26
  5.3.2 K-Nearest Neighbour (KNN) .................................................. 27
  5.3.3 Decision Tree (DTree) ......................................................... 28
5.4 PROPOSED ADVANCED CONTENT FILTERING METHOD ...................... 30
  5.4.1 The Categorisation Method .................................................... 31
  5.4.2 Choosing the Threshold ......................................................... 33
  5.4.3 How the Entire Content Filtering Component Works ....................... 33
5.5 IMPLEMENTATION AND ANALYSIS OF THE ADVANCED CONTENT FILTERING METHOD ............ 35
  5.5.1 Data ................................................................. 35
  5.5.2 Experiment Results ............................................................ 36
  5.5.3 Comparison with Existing Filtering Systems .................................. 38

CHAPTER 6 FILTERING INAPPROPRIATE IMAGES .................................. 40

6.1 INTRODUCTION ............................................................... 40
6.2 PREVIOUS INAPPROPRIATE IMAGE FILTERING SYSTEMS .................... 41
  6.2.1 Fleck-Forsyth-Bregler System ................................................. 41
  6.2.2 WIPE System ............................................................... 41
6.3 COMMENTS ON PREVIOUS IMAGE FILTERING SYSTEMS .................... 42
APPENDIX B TESTING OF THE CONTENT FILTERING FIREWALL ................. 71
APPENDIX C SOURCE CODES OF THE IMPLEMENTATION OF THE PROPOSED ADVANCED CONTENT FILTERING METHOD IN CHAPTER 5 ..................... 73
APPENDIX D SOURCE CODES OF THE IMPLEMENTATION OF THE PROPOSED ATTACK ON CONTENT FILTERING FIREWALL IN CHAPTER 8 ..................... 92
List of Tables

Table 1 Results for filtering Web sites with adult contents.......................37
Table 2 Effectiveness of utilising the 'alt' attribute..............................43
List of Figures

Figure 1 Dual-Homed Host Architecture ................................................................. 14
Figure 2 Screened Host Architecture ........................................................................ 14
Figure 3 Screened Subnet Architecture ..................................................................... 15
Figure 4 Architecture of A Content Filtering Firewall ............................................... 20
Figure 5 K-Nearest Neighbour .................................................................................. 28
Figure 6 An Example of A Simple Decision Tree ...................................................... 29
Figure 7 Building Decision Tree .............................................................................. 30
Figure 8 Experiment results of Adult URLs and Non-Adult URLs .............................. 37
Figure 9 Evaluation of existing filtering systems ...................................................... 39
Figure 10 Architecture of Filtering Viruses At Firewall ........................................... 51
Figure 11 Applet and XML Attack Demonstration .................................................... 56
Figure 12 Different Content Filtering for Different User Groups ............................... 62
Figure 13 Experimental Network Architecture ...................................................... 70
Figure 14 Test for Text Files .................................................................................... 71
Figure 15 Test for Non-text Files ............................................................................. 72
Chapter 1

Overview of the Thesis

This chapter firstly introduces the objectives of this research, and then briefly describes the outline of the thesis.

1.1 Objectives

We propose and implement a content filtering firewall. The objectives of this thesis are as follows:

- To evaluate current content filtering methods and firewalls, and propose how to combine content filtering and firewall together.
- Investigate problems with the basic keyword-matching method used in content filtering and propose an advanced content filtering method.
- Analyse and experiment possible attacks on the content filtering firewall.

1.2 Outline

Chapter 2 describes the application of content filtering, introduces previous work and some unsolved problems in this field. Chapter 3 introduces firewall and some limitations of the traditional model. Chapter 4 firstly compares filtering at the end users' terminal machines, and filtering at a central firewall, and then describes the design of a content filtering firewall. Finally, legal and ethical issues of content filtering are discussed. Chapter 5 proposes and implements an advanced content filtering method based on text categorisation techniques to replace the basic keyword-matching filtering method. Chapter 6 discusses how to use content filtering firewall to prevent computer virus propagation through the Internet. Chapter 7 proposes and
describes the implementation of an attack using encryption to get around the content filtering firewall, hence showing the difficulty that content filtering firewall encounters when end-to-end encryption for network traffic is used. Based on the architecture proposed in Chapter 4, a content filtering firewall is implemented that uses the firewall toolkit (FWTK). Chapter 8 gives details of the implementation. Chapter 9 draws some conclusions about the design of content filtering firewall and introduces possible future work in the area of content filtering firewall.

Details of the experimental network used in the implementation are in Appendix A. Details of the testing of the content filtering firewall are in Appendix B. Appendix C includes the source codes of the implementation of the proposed advanced content filtering method in Chapter 5. Error! Reference source not found. includes the source codes of the implementation of the Naïve Bayes categorisation method used in Chapter 5. Error! Reference source not found. includes the source codes of the implementation of the K-Nearest Neighbour categorisation method used in Chapter 5. Error! Reference source not found. includes the source codes of the implementation of the Decision Tree categorisation method used in Chapter 5. Appendix D includes the source codes of the implementation of the proposed attack on content filtering firewall in Chapter 8.
Chapter 2
Introduction to Content Filtering

This chapter describes the applications of content filtering, introduces previous works and some unsolved problems in this field.

2.1 Needs for Content Filtering

Content filtering is to monitor and analyse data in order to filter specific content that is forbidden by the organisation’s policy. There are a number of reasons why content filtering is needed:

- To prevent inappropriate contents.

  While the Internet can provide people with valuable information, it could also be hosts for pornographic, violent or other inappropriate contents. Content filtering can be used to prevent people from accessing such inappropriate contents. In some cases, it is compulsory to filter such inappropriate contents. The America’s Children's Internet Protection Act requires public schools and libraries that receive federal funds to install content filtering programs for Internet access [24].

- To prevent computer viruses or malicious codes.

  Computer virus is a computer program that can infect other computer programs by modifying them in such a way as to include a (possibly evolved) copy of itself [6]. In recent years, with the wide spread of the Internet, the Internet has replaced floppy disks as the primary media for computer viruses propagation. Computer viruses or malicious codes could come into the internal network via
E-mail attachments, or be downloaded by inexperienced users via HTTP or FTP protocol.

- To prevent employees’ misuse of the network.

During office hours, some employees might waste time on browsing leisure Web sites, downloading MP3 or other types of streaming video/audio files, chatting with friends, etc. Such misuses of the network not only decrease productivity but also impose unnecessary load on the traffic of the entire network, impeding other users' legitimate activities. In some cases such as downloading illegal music files or software programs, these might be legal implications not only for the user, but also for his or her organisation since the network devices is used for illegal downloading.

- To prevent sensitive or classified documents from being disclosed without being authorised.

Users could intentionally send out sensitive or classified documents to the external network without being authorised. Besides that, malicious programs such as computer viruses could also unintentionally disclose sensitive or classified documents. For example, if the Sircam virus infects a computer, the virus searches the hard drive for Microsoft Word, Excel and Zip documents. Then the virus randomly chooses one of the documents found and sends the document chosen as email attachment to a number of randomly chosen email addresses from the address book. Once sensitive or classified documents are sent out, the consequences could be serious because there are no means to retract them.
2.2 Previous Works on Content Filtering and Their Limitations

This section briefly introduces previous approaches used in content filtering and their limitations. Previous approaches include the following.

2.2.1 Blacklists and Whitelists

Blacklists are lists of all sites that should be blocked by the filtering program. Whitelists are lists of all sites that should be allowed. They are usually composed by manually examining of each Web page to decide whether or not the page is within certain undesired categories, such as "Full Nudity," "Music," "Stock Market," etc. Sites can also be included to the blacklists because their domain name contains certain keywords like "sex", "xxx", etc. For example, if www.nasdaq.com is classified into the category of “Stock Market” and is recorded in the blacklist, the filtering system will deny users' request to access www.nasdaq.com.

The problem with this method is that because numerous new sites continue to emerge everyday and the content of sites could change overtime, it is very hard to keep the lists complete and up-to-date.

2.2.2 Keyword Blocking

Some content filtering programs consider a Web page to be undesired if the page contains certain words in a keyword list. The keyword list contains selected words that frequently appear in undesired sites.

The problem with keyword blocking method is that since the meanings of some keywords depend on the context, the filtering system may wrongly block Web pages that should not have been blocked. For example, sites about breast cancer research could be blocked just because their pages contain a number of the keyword “breast” that could exist in the undesired “pornography category” keyword list.

Another problem with keyword blocking is that keywords in the undesired pages could be misspelled intentionally or unintentionally, which may cause the undesired pages
pass through the filtering system without being found. For example, malicious sites that want to get around the content filtering can intentionally omit or modify some letters in the keywords, such as modify “pornographic” to “pornogaphic”. Such minor ommittances or modifications do not cause much misunderstanding for interested audience, but do impose significant difficulties on the filtering programs trying to find the original keywords.

2.2.3 Rating Systems

Some content filtering programs utilise rating of sites. Systems such as PICS (Platform for Internet Content Selection) [30] developed by the Massachusetts Institute of Technology’s World Wide Web Consortium produce rating for sites. The PICS specification enables labels (metadata) to be associated with Internet content. The PICS standard describes two approaches to the rating of sites:

- **Self-Rating**: Web site publishers can evaluate their own content and put PICS rating information directly into the Web pages they create or distribute.

- **Third-Party Ratings**: Independent third parties can use PICS ratings systems to evaluate Web sites and publish their own results that could be used by other users for content filtering purpose.

The problem with rating systems is that:

- Rating is not always available. Since rating for Web pages is not compulsory, the content filtering systems still need to use other filtering approaches for those Web pages without ratings.

- Even if available, the ratings are not always reliable or accurate, especially in the case of self-rating. Some malicious sites may intentionally provide false ratings for their Web pages in order to get around the content filtering system.

2.3 Current Content Filtering Software

Current content filtering programs include Cyber Patrol from Microsystems Software [26], SurfWatch from SurfWatch Software [32], MIMEsweeper from Baltimore
However, few of such filtering programs are built directly into firewall. Most of them are individual applications or plug-in applications that reside on the end users’ terminal machines. Plug-in applications are third-party programs that are installed as part of the Internet browsers and used for presenting the Web pages. Therefore, such content filtering programs have the following problems:

- Dishonest users could circumvent the content filtering programs. For example, even though a content filtering plug-in program has been installed for the Microsoft Internet Explorer browser in order to prevent the users from browsing leisure sites, users can install the Opera Internet browser or any other Internet browsers and use such browser instead to browse leisure sites.

- In those cases where the users own administrator level rights for their own machines (which are very common for personal computers, especially for those running Microsoft Windows operating systems), dishonest users can easily disable the filtering by simply terminating the current running processes of filtering programs or by removing the filtering programs from their machines.

- It is hard to maintain or update such content filtering programs because the maintaining or updating need to be done for each end user’s terminal machines.
This chapter introduces firewall and some limitations of the traditional model.

Firewall is a component or a set of components that restricts access between a protected network and the Internet, or between two networks. A firewall sits between an external network and an internal network and serves as a guard for the internal network. A carefully designed and deployed firewall can deny unauthenticated requests or requests with potential threats, while permitting authenticated requests, thus protecting the internal network. Firewall is effective only when direct communications between the internal network and the external network do not exist or are restricted.

### 3.1 Needs for Firewall

This section explains why firewall is needed for protecting networks. The uses of firewall include the followings.

#### 3.1.1 Protecting Internal Networks

It is well known that the Internet is far from secure. There are hundreds of millions of people connected to the Internet. Some might maliciously try to penetrate into private networks to view or modify data, or even to damage the computer systems. According to Computer Emergency Response Team’s (CERT) statistics, the number of incidents reported in year 2001 is 52,658, while the total number of incidents from 1988 to 2001 is 100,369 [25]. Such threats can be roughly divided into two categories [22].
1. Attacks that require establishing connections between client and server, such as:

- **Command-channel attacks.** In such attack, attacker either sends valid commands that do undesirable things to the server, or sends invalid commands that exploit the server’s bugs in handling invalid input.

- **Data-driven attacks.** Examples of such attack are computer viruses appended to E-mail messages, or malicious programs downloaded that claim to be safe.

- **False authentication of clients.** In such attack, an attacker masquerades as one of the legitimate users to access resources.

2. Attacks that do not need connections between client and server, including:

- **Denial of service.** Such attacks try to prevent other people from accessing the resources by sending massive requests to the server.

- **Packet sniffing.** Packet sniffing is to watch packets going through the network in order to find useful information such as user passwords.

- **Replay.** An attacker saves past authentication information and sends it again in the future.

- **Data injection and modification.** In such an attack, the attacker intercepts the traffic between the client and server, then modifies the traffic and dismisses the original traffic.

- **Hijacking.** In a hijacking attack, an attacker takes over an open terminal or login session from a user who has successfully authenticated to the system.

No matter what kind of attacks they are, the consequences of successful attacks are the same: they could not only lead to substantial financial losses, but could also harm the reputation of the victims and hence reduce their clients’ confidence.

Firewall is an important tool to reduce such threats.
3.1.2 Enforcing Internal Security Policy

Firewall can also be used to enforce internal security policy. Organisations usually have security policy controlling users’ accesses to resources and communication services. For example, some organisations forbid outbound FTP or E-mail service for classified networks, in order to protect sensitive data stored in those networks.

3.1.3 Use of Internal Firewall

Generally firewall is deployed between the Internet and the internal network. However, in some situations, firewall can also guard parts of the internal network against its other parts. The reasons to keep one part of the internal network separate from other parts by internal firewall could be:

- Different parts of the internal network may have different responsibilities, thus need to be protected with different degrees of security assurance.

  Some parts of the private network may be more important than the rest. Examples of such domains are accounting department’s network where important financial data are stored and processed, or research & development networks that may have sensitive technical secrets, etc.

- Different parts of the internal network may face different level of security challenges.

  Some parts of the private networks may be more subject to attacks than the rest, such as the demonstration networks that any outsider can freely use.

- Different parts of the internal network may have different kinds of usages.

  For example, some parts of the private networks are used for networking experiments or testing that could affect other parts of the network if there is no protection between them.

Internal firewall is especially useful for organisations with large private networks and many divisions at different physical places.
3.2 Types of Firewalls

This section introduces three types of firewalls: Packet-Filtering, Application-Level and Circuit-Level firewall. It should be noted that in practise firewalls are seldom built on just one of these basic types, usually they are combinations of these basic types.

3.2.1 Packet-Filtering Firewall

Packet-Filtering firewall works by filtering packets based on their source and/or destination address or ports [5]. To send information across a packet switching network, the information must be broken up into small pieces called packets. Each packet is sent separately so that multiple communications among computers can proceed concurrently.

An IP packet usually contains only a few hundred bytes of payload and a set of headers containing information including the source address, destination address, source port number and destination port number. Servers for particular Internet services usually utilise certain port numbers; for example, port 80 is normally used for HTTP connections. Hypertext Transfer Protocol (HTTP) is the protocol used for communication between a browser and a Web server or between intermediate machines and Web server [7]. Therefore, according to the security policy rules, the Packet-Filtering firewall can block or allow certain types of connections by examining the source or destination port number within packet header.

Examples of the security policy rules are:

- Block all connections to or from certain distrusted hosts.
- Block insecure services such as RPC (Remote Procedure Call), rlogin, rsh, etc.
- Allow E-mail services to and from external networks, but block HTTP and FTP to and from external networks.

Packet-Filtering firewall has the advantage of being fast, but it cannot provide context about the network traffic because it considers each packet separately and does not record interaction between packets. Therefore, Packet-Filtering firewall is limited and is usually used in conjunction with other firewall components such as bastion hosts.
A bastion host is a computer system that is exposed to the Internet and highly secured. Packet filtering rules can be set up so that packets from outside can only go to the bastion host, but not to any other machines within the internal network directly.

### 3.2.2 Application-Level Firewall

Another commonly used firewall is Application-Level firewall, which usually is a dual-homed host. A dual-homed host is a computer system that has two or more network interfaces. One network interface is connected to the external network and the other interface is connected to the internal network. Application-specific server programs called proxies run in the firewall host and do the forwarding of network traffics between the internal network and the external network. The proxy program takes internal users' requests for Internet services (or external requests for services in the internal network) and forwards them, if such requests comply with the security policy rules, to the actual services.

Depending on the security policy, requests might be allowed or refused. For example, the FTP proxy might refuse to export files from secure networks, or it might allow importing files only from certified external sites. More sophisticated security policy might be to enforce different restrictions for different users or machines, rather than to enforce the same restrictions for all users or machines.

Application-Level firewall can provide better security than Packet-Filtering firewall. Because Application-Level firewall is application-specific, it can understand the traffic passing through. Application-Level firewall can also provide user authentication [5]. In addition, to an external host, it must deal directly with the proxy server, other than talk directly to the internal real host. Therefore, by disclosing less information about the internal network to the outside, the proxy server reduces threats to the internal network to some extent.

The disadvantage of Application-Level firewall is that a separate proxy is required for each supported protocols such as FTP, Telnet, HTTP, etc. Therefore, the Application-Level firewall must understand detailed knowledge of each supported protocol.

An example of Application-Level firewall is the Firewall Package Toolkit from...
3.2.3 Circuit-Level Firewall

The third type of firewall is the Circuit-Level firewall that creates a circuit between the client and the server without interpreting the application protocol [22]. The difference between Circuit-Level firewall and Application-Level firewall is that Circuit-Level firewall does not need to understand or interpret each application protocol. The most popular example is the SOCKS package, which is designed to make it easy to convert existing client/server applications into proxy versions of those same applications.

The advantage of Circuit-level firewall is that it can support any program using TCP, without the need to modify each program, because it does not need to understand the application protocols.

The disadvantage of Circuit-level firewall is that it can only control connections based on their source and destination addresses, but cannot determine whether the traffics of the connections are legitimate or not.

3.3 Firewall Architectures

This section introduces three firewall architectures: dual-homed host architecture, screened host architecture and screened subnet architecture.

3.3.1 Dual-Homed Host Architecture

In dual-homed host firewall architecture, a dual-homed host computer sits between the Internet and the internal network and acts as a router between the networks, routing IP packets from one network to another. The dual-homed host only routes those IP packets that comply with the firewall’s security policy. Because the Internet and the internal network cannot communicate directly with each other, those IP packets that breach the firewall’s security policy cannot arrive at their destinations.

An example of the dual-homed host architecture is shown in Figure 1.
Dual-homed host architecture has a high level of control over the network traffics passing through the firewall. The disadvantage to the dual-homed host architecture is that the bastion host is the single point of failure. If attackers compromise the bastion host, nothing else can protect the internal network.

### 3.3.2 Screened Host Architecture

Screened host architecture uses an external screening router attached to the Internet and a bastion host attached to the internal network. The screening router routes IP packets coming from the Internet only to the bastion host, if the packets comply with the firewall’s security policy. Therefore, any external machines want to connect to the internal network have to connect to the bastion host first. The screening router also routes IP packets coming from the bastion host to the Internet if the packets comply with the firewall’s security policy.

An example of the screened host architecture is shown in Figure 2.
The disadvantage to the screened host architecture is similar to that of the dual-homed host architecture -- the bastion host and the screening router both become single point of failure. Attackers from the Internet only need to compromise either the bastion host or the screening router in order to access resources in the internal network.

3.3.3 Screened Subnet Architecture

The screened subnet architecture has a bastion host attached to two screening routers. One screening router connects to the Internet while the other screening router connects to the internal network, as shown in Figure 3.

![Figure 3 Screened Subnet Architecture](image)

The advantage to the screened host architecture is that there is no single point of failure. To access resources in the internal network, attackers would have to compromise both routers. Even if the attackers compromise the bastion host, they still need to compromise the interior router.

Another more advanced screened subnet architecture is to have a perimeter network between the exterior router and the interior router. The perimeter network provides more vulnerable services and the internal network provides less vulnerable services. Even if the attackers break into the perimeter network via more vulnerable services, they still cannot break into the internal network so easily.
3.4 Limitations of Traditional Firewall

Although a few firewalls (such as the Check Point FireWall-1 from Check Point Software Technologies Ltd [23]) provide content examination functionality, most traditional firewalls do not examine whether the actual content of the network traffic comply with the organisations’ policy or not. The firewall policy is not based on the payload of the traffic, but on information in the IP packet header such as the connection’s type, source address or destination address of the connection, or user authentication, etc.

Another problem with traditional firewalls is that they concentrate on how to protect the private network from external threats without monitoring the behaviours of the internal users. Without this monitoring, dishonest internal users may abuse the network.

In the next chapter, we propose the design of a content filtering firewall that can enforce access control policy by analysing the contents of the network traffic passing through the firewall.
Chapter 4
Design of a Content Filtering Firewall

This chapter firstly compares filtering at the end users' terminal machines and filtering at a central firewall, and then describes the design of a content filtering firewall. Finally, legal and ethical issues of content filtering are discussed.

4.1 Comparisons of Filtering at Users' Terminal Machines and Filtering at a central Firewall

Most previous content filtering is done at the end users' terminal machines. We propose a design in which the content filtering is done at the firewall that is between the end users' terminal machines and the external network. These two different approaches both have their advantages and disadvantages.

4.1.1 Filtering at Users' Terminal Machines

Content filtering at the end users' terminal machines has the following advantages:

- Because content filtering happens at different machines, every machine computes content filtering for itself. Therefore, the speed of the network traffic will not be delayed by the computation for content filtering, making this approach faster than content filtering at firewall.

- If the network traffic is encrypted and can only be decrypted at the terminal machines, filtering programs at terminal machines may be able to check the plaintext of the encrypted traffic, while filtering programs at firewall may fail to do so. To accomplish this, the filtering programs must be configured to do content filtering when the decryption finishes.
But the disadvantages of filtering at users' terminal machines include:

- Dishonest users may be able to disable or circumvent content filtering. Because the content filtering programs reside and execute at the users' terminal machines, if users own administrator level rights for their terminal machines, dishonest users may terminate running processes of filtering programs or remove the filtering programs in order to disable filtering.

- Filtering programs are hard to maintain and update. Maintaining or updating the content filtering programs or modification to the filtering rules need to done for each of the end users' terminal machines at different places.

- It is unsuitable for terminal machines with weak computation power or limited storage space. Content filtering programs could take up considerable computation power and storage space, thus are unsuitable to be installed or executed in those terminals with weak computation power or limited storage space.

4.1.2 Filtering at Firewall

Content filtering at firewall has the following advantages.

- Users cannot disable or circumvent content filtering. Firstly, the content filtering programs are part of the firewall and execute at the bastion host. Normal users do not have administrator level rights at the bastion host, therefore cannot disable running processes of the filtering programs or remove the filtering programs. Secondly, because all the network traffic between the end users and the external network must pass through the firewall, users have no means to escape the content filtering by communicating with the external network directly.

- It is easy to maintain and update the filtering programs. To maintain or update the filtering programs and to modify the filtering rules need to done at one place – the bastion host where the firewall resides.
• It is suitable for terminal machines with weak computation power or limited storage space. Because the content filtering is done at the firewall, there will be no extra requirement for the terminal machines.

But the disadvantages of content filtering at firewall include:

• If content filtering is computationally expensive, it will slow down the entire network. Because the firewall is at the central point of the network, the delay caused by the computation of the filtering for an internal machine’s traffic will affect all other internal machines.

• If the network traffic is encrypted and cannot be decrypted by the firewall, or more generally, if the network traffic is in a form that the firewall cannot correctly interpret, content filtering at the firewall will fail. This topic will be further discussed in Chapter 8.

4.2 Extending A Firewall for Content Filtering

Functionality

To perform content filtering, the firewall cannot be only a packet-filtering firewall. It should be either an application-level firewall, or in general be able to understand the network traffic.

The firewall access control component decides whether the connection should be permitted or not, according to a security policy. Then,

• For incoming traffic, content filtering is done after the firewall’s access control component finishes, and before the traffic is sent to the internal users.

• For outgoing traffic, content filtering is done after the firewall’s access control component finishes, and before the traffic is sent to external network.

The proposed architecture for a content filtering firewall is shown in Figure 4.
The content filtering component accepts contents in text form, does the filtering and outputs the filtered result. The filtering methods used by this component can be as simple as a basic keyword-matching method, or other more sophisticated methods. We propose an advanced content filtering method to replace the basic keyword-matching method and discussed this method in detail in the next chapter.

In the design, content filtering happens after the access control component finishes, i.e., after the firewall has decided whether the connections should be allowed to pass or not. Only network traffics permitted to pass through the firewall will be checked for undesired content. In this way, because content filtering can be avoided for those traffics not allowed to pass, the performance is better than to check all network traffics.

### 4.2.1 Filtering for Text Files

Most content sent over the Internet is in text form, such as HTML (HyperText Markup Language) files. HTML is the standard document format for Web pages [5]. A file in
text form can be read word by word and therefore calculating statistical values such as word frequency is possible.

In our design, the content filtering components firstly read keywords from the files, count the word frequency, and use those frequency for determining whether the files should be allowed or not. Because content filtering components can read keywords from text files directly, no extra pre-processing work is needed.

### 4.2.2 Filtering for Non-text Files

Some files may be non-text files. For example, a file may be in Portable Document Format (PDF) or Microsoft Word format, or some may be in compressed forms using applications such as `zip` and `gzip`. Having different formats, non-text files can only be processed by the corresponding applications that understand the specific formats of the non-text files. Without knowledge about the formats of non-text files, content filtering components cannot read keywords from non-text files directly. Therefore some pre-processing work is needed to convert non-text files to text files that are acceptable for the content filtering components.

Therefore, content filtering follows such steps:

1. When a file passes through the firewall, the firewall will try to determine whether it is a non-text file. This can be done by analysing the header of the file or by calling external programs such as `gunzip`. The firewall does not use the suffix of the file to determine the file type but looks into the real content of the file. In this way, the firewall can detect undesired files whose suffix are deliberately renamed to other types of suffix or undesired files that have no suffix at all.

2. If the file is a non-text file, then the firewall calls an appropriate program to extract the text. For example, gz files will be extracted by the program `gunzip`. If the file is a text file, the firewall will feed this file to the content filtering component directly.

3. The firewall feeds the extracted text into the content filtering component.
4. After the content filtering component finishes, the firewall calls an appropriate program to repack the filtered result into its received format.

5. The firewall transfers the repacked non-text file to the destination.

It should be noted that in some cases password might be needed to extract password-protected files. The firewall does not know the password thus cannot extract the files for checking. Therefore, such files should be retained at the firewall and users are required to extract the files for the firewall.

In section 4.2, we mentioned that the filtering methods used by the content filtering component can be as simple as a basic keyword-matching method, or other more sophisticated methods. However, as we will show in Chapter 5, there are several drawbacks with the basic simple keyword-matching method. We will discuss those drawbacks and propose an “advanced” filtering method using automatic text categorisation techniques in Chapter 5.
Chapter 5

An Advanced Content Filtering Method

This chapter proposes and implements an advanced content filtering method based on text categorisation techniques to replace the basic keyword-matching filtering method.

5.1 Need for Advanced Filtering

In the basic keyword-matching filtering method discussed in chapter 2, the filtering system considers a Web page to be undesired if the page contains some keywords that frequently appear in undesired sites.

The problem with the basic keyword-matching filtering method is that words in Web pages could be deliberately modified to avoid being detected by the basic keyword-matching filtering. The simplest cases are: When a character in the keyword is, 1) substituted by another visually similar character 2) or omitted. An example of substitution is to substitute the letter ‘l’ with the number ‘1’, therefore the word “violent” becomes “viol ent”. An example of omission is to modify the word “pornographic” to “pomogaphic”. In this way, interested audiences hardly notice any difference when browsing the Web pages but the basic keyword-matching program may fail to detect the undesired content.

Although the filtering program may use pattern-matching algorithm in order to find undesired keyword out of the modified keyword, if the substituted or omitted characters are more than just one, such pattern-matching algorithm will inevitably become complex. Furthermore, such matching algorithm increases the possibility of “false positive”, which may cause the content filtering program to wrongly block legitimate Web pages.
Another problem with the basic keyword-matching method is that it is hard to construct a keyword list that can sufficiently identify Web pages we want to block, at the same time cause few "false positive" blocking. For example, we want to block pornographic contents and put the keyword "vagina" to the list, and then the filtering program may wrongly block Web pages of vagina monologues, which are true stories of women's experiences with domestic violence and gender discrimination.

To improve the filtering, we propose an advanced content filtering method based on automatic text categorisation techniques.

5.2 Introduction to Automatic Text Categorisation

Automatic text categorisation arose due to the ever-expanding amount of text documents available in digital form. With huge amount of digital documents, it is not only time-consuming but also costly for people to manually classify such documents. Therefore, research on how to automate the text categorisation attracts more and more attention recently.

Automated text categorisation is a supervised learning task, defined as assigning pre-defined category labels to new documents based on the likelihood suggested by a training set of labelled documents [21]. It should be noted that in traditional automatic text categorisation, documents do not have hyperlinks pointing to other documents, not like the Web pages.

The applications of automatic text categorisation include data mining, natural language processing, reorganising huge documents collections, information retrieval, Web searching, etc. For example, in data mining, automatic text categorisation can be used to remove invalid or irrelevant documents. Another example is that automatic text categorisation can be used to catalogue news articles so that a user may easily find articles he might be interested in.

5.2.1 Observations

There are several important observations making the problem of content filtering for
firewalls different from traditional text categorisation problems:

1. In traditional text categorisation, the system is required to assign documents to different categories (usually more than 10).

In the problem of content filtering at firewall, the filtering program only needs to make a binary decision, that is, it only needs to answer “yes” or “no” to this question: “Does the URL that users want to access belong to certain forbidden category thus should be filtered?” Uniform Resource Locator (URL) is a unique name used to identify a Web page [7]. For example, the URL www.ieee.org is used to identify the Web page of Institute of Electrical and Electronic Engineers.

2. In traditional text categorisation, most systems such as K-Nearest-Neighbour and Support Vector Machine require ‘positive’ and ‘negative’ training documents. ‘positive’ training documents are examples of documents having the same criteria with the documents to be classified. ‘negative’ training documents are examples of documents that are not in the same category as the documents to be classified.

The filtering system we propose only requires ‘positive’ training documents without using any ‘negative’ training documents. For example, if we want to filter pornographic contents, we only need to collect some examples of pornographic contents and use the examples to train our filtering system.

3. In traditional text categorisation problem, documents to be categorised are considered to be independent from each other. Successful categorisation of one document does not provide hints to determine the categories of other documents.

In the problem of content filtering firewall, most Web pages have hyper-links pointing to other related Web pages. Let a user wants to access a Web page $U$, if $U$ is not considered to be in the undesired category, the filtering program can follow hyper-links (if hyper-links are available) within $U$ and examine whether the followed pages are in the undesired category. If most of the followed pages belong to the undesired category, the first page $U$ may be considered to be in
the undesired category thus should be filtered too. Although it is possible that a Web page contains hyper-links that are not very much related to the page itself, in the real world most Web pages have hyper-links to other related pages.

5.3 Major Automatic Text Categorisation Approaches

Major automatic text categorisation approaches include Naïve Bayes (NB) [13], K-Nearest Neighbour (KNN) [21], Decision Tree (DTree) [1]. Other text categorisation approaches include Support Vector Machines (SVM) [12], Neural Network (NNet) [18], Rocchio [11], First Order Inductive Learner (FOIL) [19], etc.

5.3.1 Naïve Bayes (NB)

The Naïve Bayes probabilistic classifiers are commonly studied in machine learning [15]. The Naïve Bayes approach calculates a probability for each word in the document by calculating its relative frequency and uses these probabilities to assign a category to the document. The Naïve Bayes approach is widely used because of its simplicity and computational efficiency.

Before classification, training documents need to be fed into the filtering system. The training documents are split into different parts based on subjects. Each of these parts contains similar documents and becomes a predefined category. Let $c_k$ denotes a predefined category, $d_i$ denotes the document being classified and $W$ denotes the words in the document, given a document $d_i$ to be classified, the Naïve Bayes approach uses the joint probabilities of the document’s words and the predefined categories to estimate the probabilities of the document’s categories. Naïve Bayes assumes that the conditional probability of a word $w_j$ given a category $P( w_j | c_k)$ is independent of the conditional probabilities of any other word $w_j$ given the same category $P( w_j | c_k)$. The assumption of word independence makes the computation efficient because it does not use word combinations as prediction. But it should be noted that the assumption of word independence is not true for most documents in the real world. Most words in real world documents are related. The assumption of word independence allows the conditional probability of the category given a document to be computed [14]:
To calculate the conditional probabilities of presence of each word $w_j$ given the category $P( w_j \mid c_k )$, the filtering system only needs to count the number of occurrence of each word under each category.

Although the Naïve Bayes approach has been studied and used for many years [13], many other recent approaches such as K-Nearest Neighbour (KNN) and Support Vector Machines (SVM) outperform the Naïve Bayes approach in terms of classification performance [21]. Therefore, the Naïve Bayes approach is often used as baseline for comparisons with different automatic text categorisation approaches.

### 5.3.2 K-Nearest Neighbour (KNN)

K-Nearest Neighbour is a well-known statistical approach, which has been intensively studied in pattern recognition for over four decades [8]. K-Nearest Neighbour is among the top-performing automatic text categorisation methods.

In general, given a document to be classified, the KNN categorisation system selects $k$ most similar documents to the document being classified from the training documents, and uses the categories of the $k$ documents to judge the categories of the document being classified.

In the K-Nearest Neighbour approach, documents are represented by vectors of the words that appear in the documents. The similarity between the document being classified and the training documents can be measured by the Euclidean distance between the two documents' vectors or other functions [21]. For example, in Figure 5, because the training document $A$ is closer to the document being classified $X$, $A$ is considered to be more similar to $X$ than $B$ is. The similarity values between the document being classified and the $k$ most similar training documents are used to calculate the possibility that the document being classified belongs to the $k$ documents' categories. If two or more documents out of the $K$ documents are in the same category,
the similarity values are summed up. The possible categories are sorted by the sum of the similarity values. Only those categories with the sum of the similarity values higher than a certain threshold value are considered to be the categories that the document being classified belongs to.

![Figure 5 K-Nearest Neighbour](image)

The optimal threshold value can be determined through experiments using a subset of the training documents as documents to be classified.

Let \( x \) denotes the document being classified, \( d \) denotes the training document set, \( c \) denotes the categories, \( \text{sim}(x, d_i) \) denotes the similarity between the document being classified \( x \) and a training document \( d_i \), \( y(x,c_j) \in \{0,1\} \) denotes whether the document \( x \) belongs to the category \( c_j \) (\( y = 1 \) for YES, and \( y = 0 \) for NO), \( y(d_i,c_j) \in \{0,1\} \) denotes whether the document \( d_i \) belongs to the category \( c_j \), \( b_j \) denotes the category-specific threshold for the category \( c_j \), the decision rule in KNN can be written as [21]:

\[
y(x,c_j) = \sum_{d_i \in \text{KNN}} \text{sim}(x,d_i) y(d_i,c_j) - b_j
\]

The K-Nearest Neighbour approach is simple and effective. Because KNN's computation time is linear to the number of training documents, KNN is especially suitable for applications that require frequent updating of the training documents [20].

5.3.3 Decision Tree (DTree)

Decision tree is a well-known machine learning approach to automatic induction of classification trees based on training data [15].

In the decision tree approach, each internal node of the decision tree has a test involving an attribute, and an outgoing branch for each possible outcome. Each leaf is
associated with a predefined category. In order to use the decision tree to classify
documents, beginning with the root node, successive internal nodes are visited until a
leaf is reached. At each internal node, the test for the node is applied to the document
being classified. The outcome of the test at an internal node determines the branch
traversed, and the next node visited. The category for the document being classified is
the same category of the final leaf node. Thus, the conjunction of all the conditions for
the branches from the root to a leaf constitutes one of the conditions for the category
associated with the leaf [17].

Figure 6 (b) is a simple decision tree for the training set shown in Figure 6 (a). It can
be observed from the training set that only documents which contain both the word
"abstract" and the word "bibliography" should be classified into the category "Paper",
and other documents should be classified into the category "Non-paper".

<table>
<thead>
<tr>
<th>Document</th>
<th>Contains word &quot;abstract&quot;</th>
<th>Contains word &quot;bibliography&quot;</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>YES</td>
<td>YES</td>
<td>Paper</td>
</tr>
<tr>
<td>B</td>
<td>YES</td>
<td>NO</td>
<td>Non-Paper</td>
</tr>
<tr>
<td>C</td>
<td>NO</td>
<td>YES</td>
<td>Non-Paper</td>
</tr>
<tr>
<td>D</td>
<td>NO</td>
<td>NO</td>
<td>Non-Paper</td>
</tr>
</tbody>
</table>

(a)

(b)

**Figure 6 An Example of A Simple Decision Tree**

To build the decision tree used for categorisation, the training documents set is
recursively partitioned until all the documents in a partition have the same category. A
new node is added to the decision tree for every partition. Initially, the decision tree
only has a single root node for the entire training documents set. For a set of
documents in a partition \( P \), a test criterion \( T \) for further partitioning the set into \( P_1, \ldots, P_m \) is first determined. New nodes for \( P_1, \ldots, P_m \) are created and these are added to
the decision tree as children of the node for \( P \). Also, the node for \( P \) is labelled with test
\( T \), and partitions \( P_1, \ldots, P_m \) are then recursively partitioned. A partition in which all
the documents have identical category labels is not partitioned further, and the leaf
Therefore, the steps for building a decision tree with the training documents shown in Figure 6 (a) are illustrated in Figure 7.

![Decision Tree Diagram](image)

**Figure 7 Building Decision Tree**

If the decision tree in Figure 6 (b) is used to classify the documents in the training set, only document A is classified into the category “Paper” by the decision tree. All the other documents B, C and D are classified into the category “Non-paper” by the decision tree.

Although its classification performance is not among the best, the decision tree algorithms are efficient for processing huge amounts of data [19]. Compared with the neural networks approach that can take large amounts of time and thousand of iterations, decision tree is efficient and is thus suitable for large training sets [17].

### 5.4 Proposed Advanced Content Filtering Method

There have been some works in categorising Web pages using naïve Bayes (NB), K-Nearest Neighbour (KNN), etc. Yiming Yang, S. Slattery and R. Ghani compared the effectiveness of using NB, KNN and First Order Inductive Learner algorithms to classify Web pages [38]. They also investigated the use of hyperlinks, content of linked documents, and meta data about related Web sites and found that meta data can be extremely useful for improving classification accuracy [39]. Norbert Govert, Mounia Lalmas and Norbert Fuhr used an enhanced document representation that incorporates the structural and heterogeneous nature of Web documents [40]. But few
works have been done in how to filter undesired Web contents using automatic text categorisation methods.

5.4.1 The Categorisation Method

Based on the observations above and previous automatic text categorisation approaches, we propose a categorisation method for content filtering firewall to determine whether the URLs that a user requests belong to an undesired category or not.

Each document is represented by a vector of word frequencies. The more similar two documents are, the closer the two documents' vectors are, thus the smaller angle between the two document's vectors is, and therefore, the higher the cosine value of the angle between the two documents' vectors is. Therefore, the similarity of two documents can be measured by the cosine of the angle between the two document vectors, which can be calculated by $\cos( X, Y ) = \frac{\sum X_i Y_i}{\sqrt{\sum X_i^2} \times \sqrt{\sum Y_i^2}}$, where $X$ and $Y$ are the two document vectors. Training documents that belong to the undesired category must be fed into the filtering program. The filtering program calculates the similarity value to the other documents for each training document. Based on the similarity values of all training set, a threshold value is chosen. When a user wants to access a Web page, the filtering program calculates the Web page’s similarity value to the undesired category and compares it with the threshold value to determine whether the Web page is in the undesired category. To calculate the page’s similarity value to the undesired category, the system calculates its similarity value to each training document in that category, and then calculates the average of the n% highest similarity values, where n is a number such as 30. If the Web page’s similarity value to the undesired category is less than the threshold value, the filtering program continues to get the next hyper-link within the Web page and calculates its similarity value to the undesired category, which will repeat for $r$ times. If majority of the $r$ links belong to the undesired category, the first Web page is considered to be in the undesired category thus should be blocked.
Following are the detailed steps:

1. Before the filtering program can be used to determine whether a URL belongs to certain undesired category or not, training documents that belong to the undesired category must first be fed into the filtering program. This needs to be done once when the filtering program runs for the first time and requires to be redone when a new category is requested (hence the training documents set changes).

When the filtering program processes these documents,

- Some common words are ignored. The reason is to avoid common words that might appear in any kind of categories, such as “the”, “and”, “for”, etc.
- Words with frequency less than certain threshold or a fixed number such as 2 are ignored. The reason is to avoid "noise" words.
- Words shorter than 2 letters are ignored. The reason is to avoid words like “a”, “to”, “of”, “in”, etc. Another reason is to avoid special symbols like ‘@’, ‘?’, etc.

During the training time, the filtering program reads in the training documents that belong to the undesired category. The program counts the frequency of distinct words for each training document, adjusts the frequency weighting for keywords within different HTML tags, and then calculates and stores their frequency into different vectors. Each document has its own vectors to store the frequency for each distinct word appears in that document.

2. Then the filtering program calculates the similarity value to the other documents for each training document:

The similarity value $sim(X, Y)$ of any two documents $X$ and $Y$ is $cos(X, Y) = \frac{\sum X_i Y_i}{\sqrt{\sum X_i^2} \times \sqrt{\sum Y_i^2}}$, where $X$ and $Y$ are vectors of the frequency for distinct words appears in the documents [21].

The similarity value of one training document $t$ to the other training documents can be calculated by averaging the similarity values of the document $t$ and each of the
other training documents, as \( \frac{\sum \text{sim}(t, D_i)}{\text{Total number of training documents} - 1} \), where \( D \) is set of the other training documents except the document \( t \).

3. The next step is to empirically set the threshold value \( (T) \) for this undesired category. The threshold value \( T \) will be compared with the similarity value of the document to be checked to the training documents. The document to be checked is considered to be in the undesired category if its similarity value to the training documents is greater than \( T \). The chosen threshold value \( T \) should be less than most similarity values of training documents to the others.

4. Suppose a user wants to access a Web page \( U \), if the similarity value of \( U \) to the training documents of the undesired category is less than the threshold, i.e. \( U \) is not considered to be in the undesired category, the filtering program continues to get the next hyper-link within \( U \) and examines its similarity value to the undesired category, which will repeat for \( r \) times. If majority of the \( r \) links belong to the undesired category, the first page \( U \) is considered to be in the undesired category thus should be blocked.

5.4.2 Choosing the Threshold

We want to maximise the number of undesired pages that are successfully blocked by the filtering program, at the same time minimise the number of legitimate pages that are wrongly blocked. It can be observed that there exists trade-off between these two goals. To maximise the number of undesired pages that are successfully blocked, we need to decrease the threshold \( T \). But to minimise the number of legitimate pages that are wrongly blocked, we need to increase the threshold \( T \).

We suggest that the threshold \( T \) should be higher than the similarity values of most training documents to the others. For example, the threshold \( T \) can be chosen so that the similarity values of 99% training documents to the others are higher than \( T \).

5.4.3 How the Entire Content Filtering Component Works

Using the categorisation method above, the entire content filtering component works
as the following:

Suppose a user requests to access the URL W,

Step 1: The content filtering component checks the “allowed” URL list, which records URLs considered to be free of undesired contents. If W is in this list, the request is allowed without further examination. Be aware that the construction of such “allowed” list should be extremely careful; otherwise malicious sites could relay undesired contents from other sites even if those sites themselves contain no undesired contents.

Step 2: The content filtering component checks the “undesired” URL list. The “undesired” URL list records URLs that contain undesired contents. If W is in this list, the filtering component will block the user’s request for W.

Step 3: The content filtering component retrieves W from the Internet and checks whether there are information descriptors such as PICS (Platform for Internet Content Selection) or meta-data descriptors available in W. The information descriptors and meta-data descriptors have been discussed in Chapter 2.

If there are descriptors available and according to such descriptors the URL W contains undesired content, the filtering component will block the user’s request.

But if there are descriptors available and according to such descriptors the URL W contains no undesired content, the filtering component still needs to do further examination of W. The reason is that some meta-data are self-regulated by the site owner, as discussed in Chapter 2; therefore malicious sites could make up false descriptors for their pages in order to get around content filtering.

Step 4: The content filtering component feeds W into the categorisation component. If the categorisation component reports that W belongs to any of the undesired categories, the filtering component will do the followings:

1) Add W into the training set of the undesired category for the categorisation component, in order to increase accuracy of the further categorisation.

2) Add W to the “undesired” URL list, in order to improve performance of further filtering because other users’ request for W will be blocked without examination for W’s category.
3) Block the user’s request for W and record relevant information to the log file.

Step 5: The content filtering component allows the user’s request.

To summarise, the content filtering component’s action to a user’s request for a Web site W is:

\[
\text{Action to request for } W = \begin{cases} 
\text{Allow, if } W \in \{\text{"allowed.URL list"}\} \\
\text{Deny, if } W \in \{\text{"undesired URL list"}\} \\
\text{Deny, if } \forall \text{ descriptors for } W \land \text{ descriptors } \in \{\text{"undesired content"}\} \\
\text{Deny, if } W \text{ does not pass the categorisation component} \\
\text{Allow, otherwise}
\end{cases}
\]

5.5 Implementation and Analysis of the Advanced Content Filtering Method

We have implemented the advanced content filtering method as a separate program with Java 1.4. We also collect pages from various Web sites and use them to test the filtering system. The experimental results have been very encouraging. Appendix C includes the source codes of the implementation.

5.5.1 Data

In the following experiment, our system aims to filter Web sites with Adult contents.

We collect 487 URLs from the Adult category of Google (URL: http://directory.google.com/Top/Adult/). Web sites in this category are reviewed and classified as containing adult contents by human beings editors. Within this category, we obtain the list of top 500 adult Web sites by searching with the keyword porn. Out of the 500 URLs, 13 are invalid or unavailable and we exclude them. We use the remaining 487 URLs as training data.

For all the URLs (385 when the data were collected) under the Adult category of Yahoo! (URL: http://dir.yahoo.com/Business_and_Economy/Shopping_and_Services/Sex/Directories/), we exclude those URLs that are also in the training set or are
invalid. We use the remaining 329 URLs as test data. The filtering system is used to recognise those URLs as containing adult contents.

Also, to determine how good our system is in recognising Web sites without adult contents, URLs that do not contain Adult contents are collected from 10 top directories of Google. Those categories are: arts, business, science, computers, news, shopping, games, society, health, and sports. For each category, we obtain a list by searching with the keyword same as the category name (keyword *arts* for the arts category, keyword *business* for the business category, etc.). Then for each list, we select 20 URLs at the top, 20 URLs in the middle and 20 URLs at the bottom. For the selected 600 URLs, we exclude those invalid or unavailable URLs and get 587 URLs.

Totally, 1,403 URLs are collected and used for the experiment.

### 5.5.2 Experiment Results

The experiment is run for multiple times. For each run, we randomly select 10% of the test adult URLs and non-adult URLs as the test data for this run. Using the same training data, the filtering program calculates the similarity values to the training data (adult URLs) for every test documents. Then with different thresholds, the filtering program calculates the corresponding blocking rate and over-blocking rate. When the multiple runs finish, the program calculates the 95% confidence intervals of blocking rate and over-blocking rate for different thresholds.

The following figure is the result for a typical test run. The Y-axis is the similarity value to Adult category calculated by the filtering system and its range is between 0 and 1. The X-axis includes the set of adult URLs tested and the set of non-adult URLs tested. Both sets of URLs are separately sorted ascending by their similarity values to Adult category. Test URLs are given high similarity values if the filtering system believes those URLs contain adult contents. Test URLs are given low similarity values if the filtering system believes those URLs do not contain adult contents.
Figure 8 Experiment results of Adult URLs and Non-Adult URLs

It can be clearly observed that our filtering system gives Web sites that actually contains adult contents quite high similarity values to the Adult category (most are higher than 0.2), while giving Web sites free of adult contents quite low values (generally less than 0.2). Therefore, with different threshold values, we can obtain the following results:

Table 1 Results for filtering Web sites with adult contents (100 test runs)

<table>
<thead>
<tr>
<th>Blocking Rate (mean)</th>
<th>Over-Blocking Rate (mean)</th>
<th>Threshold</th>
<th>95% Confidence Interval of Blocking Rate</th>
<th>95% Confidence Interval of Over-Blocking Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.35%</td>
<td>4.09%</td>
<td>0.10</td>
<td>99.21% to 99.48%</td>
<td>3.75% to 4.43%</td>
</tr>
<tr>
<td>98.99%</td>
<td>2.03%</td>
<td>0.12</td>
<td>98.84% to 99.15%</td>
<td>1.79% to 2.28%</td>
</tr>
<tr>
<td>98.80%</td>
<td>0.92%</td>
<td>0.14</td>
<td>98.62% to 98.98%</td>
<td>0.74% to 1.09%</td>
</tr>
<tr>
<td>98.80%</td>
<td>0.59%</td>
<td>0.16</td>
<td>98.62% to 98.98%</td>
<td>0.46% to 0.71%</td>
</tr>
<tr>
<td>97.41%</td>
<td>0.48%</td>
<td>0.18</td>
<td>97.13% to 97.68%</td>
<td>0.36% to 0.59%</td>
</tr>
<tr>
<td>95.34%</td>
<td>0.34%</td>
<td>0.2</td>
<td>94.98% to 95.71%</td>
<td>0.24% to 0.44%</td>
</tr>
</tbody>
</table>
Total number of adult URLs tested by the filtering system: 329
Total number of non-adult URLs tested by the filtering system: 587
Total number of adult URLs used to train the filtering system: 487

Blocking Rate = \[
\frac{\text{Total number of adult URLs blocked by the filtering system}}{\text{Total number of adult URLs tested by the filtering system}}
\]

Over-Blocking Rate = \[
\frac{\text{Total number of non-adult URLs blocked by the filtering system}}{\text{Total number of non-adult URLs tested by the filtering system}}
\]

It can be observed that the performance of our filtering system is quite satisfactory.

### 5.5.3 Comparison with Existing Filtering Systems

The NetProtect project in Europe evaluates the filtering efficiency and the over-blocking rate of existing filtering systems. In year 2001, 2,794 URLs with pornographic content and 1,655 URLs with normal content were collected and used to test 50 commercial filtering solutions [37]. Because the theme of NetProtect’s test is very similar to that of our experiment, their results can be used to compare with ours. NetProtect’s test results for the 10 most popular filtering tools are as follows:
<table>
<thead>
<tr>
<th>Filtering Tools</th>
<th>Blocking Effectiveness</th>
<th>Overblocking rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>BizGuard</td>
<td>55%</td>
<td>10%</td>
</tr>
<tr>
<td>Cyber Patrol</td>
<td>52%</td>
<td>2%</td>
</tr>
<tr>
<td>CYBERsitter</td>
<td>46%</td>
<td>3%</td>
</tr>
<tr>
<td>Cyber Snoop</td>
<td>65%</td>
<td>23%</td>
</tr>
<tr>
<td>Internet Watcher 2000</td>
<td>30%</td>
<td>0%</td>
</tr>
<tr>
<td>Net Nanny</td>
<td>20%</td>
<td>5%</td>
</tr>
<tr>
<td>Norton Internet Security</td>
<td>45%</td>
<td>6%</td>
</tr>
<tr>
<td>Optenet</td>
<td>79%</td>
<td>25%</td>
</tr>
<tr>
<td>SurfMonkey</td>
<td>65%</td>
<td>11%</td>
</tr>
<tr>
<td>X-Stop</td>
<td>65%</td>
<td>4%</td>
</tr>
</tbody>
</table>

Figure 9 Evaluation of existing filtering systems

Comparing the above table with results for our system, it is clear that our filtering system outperforms the existing filtering systems in filtering Web sites with adult contents. It should also be noted that our filtering system does not use any blacklist or whitelist, therefore we avoid the huge costs introduced by manually classifying and maintaining such lists.
Chapter 6
Filtering Inappropriate Images

This chapter first introduces problems with inappropriate images, and then discusses previous image filtering systems. Finally, it covers the preliminary alternative method and the experimental results.

6.1 Introduction

The method proposed in the previous chapter only utilises text information but ignores images within the Web pages. That could lead to the following problems:

- Inappropriate text can be converted to images in order to get through the filtering system. For example, a Web site designer can convert a forbidden article to one (or multiple) image file(s) in JPEG, GIF or other format, and embed such images in a Web page. Inappropriate content in image form is able to get through text-based filtering systems. However, it is troublesome and time-consuming for Web site designers to consistently convert all the text content to images. Besides that, the converted images are usually much larger than the original texts in file size. Therefore, such Web sites are less preferable for users, especially for those users with low-band Internet connection, such as modem.

- Watermarking techniques can be used to embed inappropriate content into images to get through filtering. Embedded information cannot be observed visually or be extracted without knowing the key, which makes it hard for the filtering system to detect the embedded inappropriate content. However, to retrieve the inappropriate content, the viewer must collaborate with the outsider to obtain the key, and then use specific watermarking tools to extract embedded
information from the watermarked images. Nowadays, trying to thwart such elaborate collaboration between the insiders and the outsiders is still far beyond the capacity of any existing Internet filtering system.

- For certain inappropriate categories, such as pornographic contents, Web pages may have many inappropriate images and few texts by nature, thus making text-based filtering methods less ineffective.

6.2 Previous Inappropriate Image Filtering Systems

6.2.1 Fleck-Forsyth-Bregler System

This system [33] classifies images as pornographic or non-pornographic. The published results are 52 percent sensitivity (i.e., 48 percent false negatives) and 96 percent specificity (i.e., 4 percent false positives). It has an average processing time of 6 minutes per image, which makes it unsuitable for real-time applications.

The system first locates images containing large areas of skin-colour region. Then, within these areas, it finds elongated regions and groups them into possible human limbs and connected groups of limbs. Images containing sufficiently large skin-colour groups of possible limbs are reported as potentially containing naked people.

6.2.2 WIPE System

WIPE system [34][35][36] analyses image content and classifies it as pornographic or non-pornographic. It has 96 percent sensitivity, and 91 percent specificity, and the average processing time is less than 1 second per image.

The system used feature extraction using wavelet transform and compares the result with pre-stored vectors if the image is a photograph (versus drawing). Manually generated images are distinguished on the basis of tones: sharp tones for manually generated images versus continuous tones for photographs.

The system uses a database containing thousands of objectionable images and thousands of benign images (10,000 photographic images) to find all images that are
close to the test image. Every image has a feature vector that encompasses information such as texture, colour, etc. Images are classified according to those vectors.

6.3 Comments on Previous Image Filtering Systems

Although some existing image filtering systems produce considerately good results, for as real-time filtering systems, it is crucial to take into account of the time delay introduced by the image analysis. If the average delay for a Web page were longer than 5 to 10 seconds, it would be unbearable for most users.

In Fleck-Forsyth-Bregler system, the average processing time per image is 6 minutes. In WIPE system, the average time per image is around one second. For a typical Web page that contains 5 or 10 images, the delay could vary from 5 seconds to 30 minutes. That might not be satisfactory enough for a real-time filtering system.

Besides that, those systems initially aim at filtering pornographic images only. There are no statistics or experiments to determine whether those systems can be easily applied in filtering other undesired content, such as violence, racism, etc.

Finally, undesired contents embedded in watermarked images cannot be easily extracted if the key is unknown. It is also very time-consuming to try to detect whether there are embedded information within images, what are the embedded information and whether the embedded information are undesired.

6.4 A Preliminary Alternative Method to Complicated Image Analysis Methods

In order to avoid long delay and complexity of the filtering algorithm, instead of using complicated image analysis techniques, we may use meta data in HTML documents to provide some initial filtering. In particular, we may use a preliminary alternative method of utilising the 'alt' attribute of HTML <img> tag.

The 'alt' text of an image will be displayed if one of the followings is true:
The corresponding image cannot be displayed, either because the viewer disables the ‘display image’ option of the Internet browser or because the image is not available at that moment.

The Viewer is using a text-based Internet browser such as lynx; therefore, the image is not displayed.

The viewer’s mouse is positioned over the corresponding image.

The purpose of 'alt' attribute is to provide the viewers with alternative information to the corresponding image. Because most 'alt' texts are related to their corresponding images, it is possible for the filtering system to utilise the 'alt' texts in order to interpret the content of the corresponding image, without the need to analyse the image content by using complicated image analysis methods.

The advantage of this method is its simplicity and high speed. The filtering system is still text-based; therefore, the system avoids the costs or delay introduced by computation-intensive image processing. In addition, to accomplish the 'alt' attribute handling, only a few modifications needed to be done to the HTML parsing component of the original filtering system.

Following is the comparison table of experimental results between utilising the 'alt' attribute and not utilising the 'alt' attribute. We test with the same data used in the precious section. Our filtering system utilises the 'alt' attribute by including the 'alt' texts as part of the page’s texts, which only introduces few processing costs.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Not utilising 'alt'</th>
<th>Utilising 'alt'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking Rate</td>
<td>Over-blocking Rate</td>
<td>Blocking Rate</td>
</tr>
</tbody>
</table>

Table 2 Effectiveness of utilising the 'alt' attribute
<table>
<thead>
<tr>
<th>0.12</th>
<th>99.08%</th>
<th>1.53%</th>
<th>99.08%</th>
<th>2.21%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.14</td>
<td>98.78%</td>
<td>1.20%</td>
<td>98.78%</td>
<td>1.53%</td>
</tr>
<tr>
<td>0.16</td>
<td>98.48%</td>
<td>0.68%</td>
<td>98.48%</td>
<td>0.51%</td>
</tr>
<tr>
<td>0.19</td>
<td>94.53%</td>
<td>0.51%</td>
<td>94.83%</td>
<td>0.34%</td>
</tr>
<tr>
<td>0.21</td>
<td>93.62%</td>
<td>0.34%</td>
<td>93.92%</td>
<td>0.17%</td>
</tr>
<tr>
<td>0.22</td>
<td>92.10%</td>
<td>0%</td>
<td>92.40%</td>
<td>0%</td>
</tr>
</tbody>
</table>

The results above show that the preliminary method archives slightly better results than the original filtering system at most cases.

Despite of its simplicity and high speed, the limitations of this method are:

- Since the 'alt' attribute is optional, not all images have such attributes.
- Not all the 'alt' texts are accurate or meaningful.
Chapter 7
Using Content Filtering Firewall to Prevent Computer Virus Propagation

This chapter discusses how to use the content filtering firewall to prevent computer virus propagation through the Internet.

7.1 Introduction to Computer Virus

Computer virus is a computer program that can infect other computer programs by modifying them in such a way as to include a (possibly evolved) copy of itself [6]. Although some computer viruses are comparatively harmless, most computer viruses may cause damage to or destroy files and disks. Harms caused by computer viruses also include the following:

- Occupying disk space and/or main memory.
- Using up CPU processing time
- Costing the time and efforts in detecting and removing the computer viruses.
- Disclosure of user data.

7.2 Previous Anti-Virus Approaches

Major previous anti-virus approaches include [4]:

- Virus signature Scanner

Virus signature scanners search suspected files for certain signatures that are known to exist in computer viruses. More advanced computer virus scanners
use heuristic technique to find suspicious operation instructions within executable files in order to detect possible new computer viruses. An example of the suspicious operation instructions is to modify the first few bytes of an executable file, which is a common operation and symptom of computer viruses.

The disadvantage of virus signature scanner is that it requires frequently updating of the virus signatures. Otherwise, even with the heuristic technique, the virus scanner still cannot find all newly emerged computer viruses. But nowadays many virus signature scanners have the ability to automatically update virus signatures through the network at regular intervals, thus alleviate this problem to some extent.

- Integrity Checker

Integrity checkers record the signatures of executable files beforehand while they are not infected by computer viruses, then regularly compare those original signatures with the signatures of the existing executable files. If an executable file’s signature is found to be different with its original signature, the executable file may has been modified, which is a symptom of virus infection. Note that integrity checkers could give false alarm of virus infection because not all modifications to the executable files are caused by computer viruses. Therefore, integrity checkers may prompt the users for what actions should be performed to the modified executable files. Users may need to have some knowledge of their computer systems in order to discern legitimate modifications to executable files from virus infection.

To record the original signatures, the integrity checker programs need to run beforehand when the executable files are in the state of not being infected. But in the case of filtering computer viruses at firewall, it is hard to get such original signatures of the executable files to be checked, thus making it hard to use the integrity checking technique for filtering computer viruses at firewall.

- Behaviour/Activity monitor
Behaviour/activity monitors try to detect suspicious activities of the running programs. Suspicious activities are common activities of virus-infected programs such as modifying the boot sector of the hard disks so that every time when the operating system boots, the virus-infected program will execute.

The major difficulty with using behaviour/activity monitor for filtering virus at firewall is that the firewall bastion host may have different architecture and operating systems from internal users’ machines. For example, in a scenario where parts of the internal users’ machines are IBM-PC compatible computers running Microsoft Windows operating systems and the other parts are Macintosh Apple machines, and the firewall bastion host is a Unix or Linux machine, it is very hard to execute suspicious programs (which could be Microsoft Windows executables or Macintosh executables) in the bastion host to do behaviour/activity monitoring.

Therefore, this thesis considers virus signature scanner to be the most preferred approach for filtering computer viruses at firewall. Behaviour/activity monitor could also be an option if most internal users’ machines and the firewall bastion host have the same architecture and operating systems.

It should be noted that there is no algorithm exists that can precisely detect all possible viruses, which was proved by Fred Cohen in “Computer Viruses - Theory and Experiments” 1984 [6]. No matter how good the virus signature scanner is, filtering computer viruses at firewall cannot filter all computer viruses without false positive. What can be expected is that the filtering is practically good enough to filter as many computer viruses as possible.

7.3 Virus Propagation Through The Internet

This section introduces the major two ways of computer virus propagation through the Internet: E-mail and HTTP. Since HTTP is much simpler than E-mail, this section mainly describes how the E-mail system works.

In those early years when computer viruses firstly appeared, the major media of computer virus propagation is floppy disk. But with the widely use of the Internet, the
Internet has become the major media of computer virus propagation.

E-mail and HTTP are the major two ways for computer viruses to come from the Internet into the internal networks.

- Computer viruses can propagate through E-mail if users inadvertently choose to send to other people virus-infected files as E-mail attachments. Besides that, some computer viruses such as Nimda can automatically send themselves through the Internet as E-mail attachments. Such computer viruses are able to complete the entire E-mail sending process, without the need for any intervention or notice of the users. Such kind of computer viruses can propagate more actively since they have the ability to complete the E-mail sending process by themselves.

- For HTTP, inexperienced users may download “free” or “trial” programs from the Internet, which could possibly be virus-infected. When such downloaded programs are executed, the user’s machine is infected by the computer virus. In addition to being able to propagate through E-mail, some computer viruses such as the Nimda virus can also infect computers used as Web servers. When users browse such virus-infected Web servers via HTTP, their machines may also be infected.

### 7.3.1 Mail Standard

With the original Internet mail standard, only ASCII text was supported. The MIME (Multipurpose Internet Mail Extensions) protocol is a set of extensions to the original Internet mail standard. MIME provides a standard for encoding mail using different language character sets, for creating mail that contain non-text content, and for segmenting mails into pieces that can be sent separately. Under MIME, mail attachments can not only be text files, but also be executable, graphics and music files, etc. The set of data at the beginning of a MIME mail is called MIME headers. The MIME headers contain information about what type of data the mail contains, directives for the mail server, etc.
7.3.2 Email Delivery

1. Sending E-mail Messages

When a user sends E-mail messages, the messages are firstly sent from his/her client computer to the mail server. The mail server usually is an internal mail server in the same network as the user, but it could also be an external mail server outside the user's network.

In most cases, Simple Mail Transfer Protocol (SMTP) is used to send E-mail messages from the user's computer to the mail server. The SMTP protocol specifies how the mail delivery system passes messages across from one machine to another.

Another way to send E-mail messages is through Web-based E-mail interface, such as Hotmail. In this case, the Web is used as an interface and the Web server receives the user's input then delivers the messages to the mail server. In this case, the protocol used between the user's client computer and the Web server is HTTP.

2. Relaying E-mail Messages

Normally E-mail messages need to be relayed between many mail servers before the messages can finally arrive at their destination. The protocol used to transfer E-mail messages between the relaying mail servers is SMTP.

3. Retrieving E-mail Messages

When a user tries to retrieve E-mail messages from the mail server, the protocols used by his/her client computer and the mail server can be one of the following:

- Post Office Protocol (POP). POP is used to transfer E-mail messages from a permanent mailbox to a local computer possibly without permanent connectivity [7].

- Internet Message Access Protocol (IMAP). IMAP is an alternative to POP but with more powerful functions. IMAP allows a user to dynamically
create, delete, or rename mailboxes and provides other extended functionality such as obtain information about a message or examine header fields without retrieving the entire message [7].

Another way to retrieve E-mail messages is through Web-based E-mail interface. The Web server retrieves E-mail messages from the mail server and sends the messages (in form of Web pages) to the user’s computer via HTTP protocol.

7.4 Proposed Architecture of Using Content Filtering Firewall to Prevent Computer Viruses Propagation

In addition to educating users on anti-virus knowledge and deploying frequently updated real-time anti-virus software to users’ machines, to protect against computer virus, we can use the content filtering firewall to filter computer virus so as to provide further protection for users. Filtering computer virus at firewall is especially useful for the most vulnerable user group -- inexperienced users who do not have enough competent computer knowledge on computer virus protection.

This thesis considers three cases. They are: 1) When no encryption is used for the network traffic; 2) When users use encryption for the network traffic; 3) When computer virus encrypts itself before passing through the firewall, then decrypts itself at the user’s machine.

7.4.1 No Encryption is Used for the Network Traffic

When no encryption is used for the network traffic, the virus signature scanner at firewall can search the network traffic passing through the firewall for virus signatures. If certain virus signature is found in the traffic, the traffic is blocked by the firewall. It is very similar to content filtering for text files using the keyword-matching method, but in this case, the keywords are computer virus signatures, not undesired strings.

The proposed architecture of filtering viruses at firewall for E-mail and HTTP is as shown in Figure 10.
It can be observed from Figure 10 that there are three possible channels through which computer viruses can pass through the firewall:

1. Traffic between the external relaying SMTP mail server and the internal mail server
2. Traffic between the external SMTP mail server and the internal user's client computer
3. Traffic between the external Web-based mail server and the internal user's client computer

Therefore, in order to prevent computer viruses propagation through E-mail, solely filtering computer virus for traffic between the external SMTP mail servers and the internal network is not enough. We also need to filter computer virus for HTTP traffic.
to provide virus protection for Web-based E-mail.

It should be noted that filtering computer virus for HTTP traffic not only provides virus protection for Web-based E-mail, but also provides virus protection for downloading files via Web.

7.4.2 Users Use Encryption for the Network Traffic

If users use encryption for the network traffic, the traffic passing through the firewall is in an encrypted state. Not knowing the decryption key, the virus signature scanner cannot decrypt the encrypted traffic, thus cannot check the traffic for virus signatures.

There are several possible approaches in which encryption is used for network traffic and filtering virus at firewall is still possible:

- Do not allow the internal users to do encryption themselves but let the firewall to do encryption, such as using VPN (Virtual Private Network). The network traffic between the firewall and the external network is encrypted, but the network traffic between the firewall and the internal users’ machines is not encrypted. Therefore, the virus signature scanner at the firewall can still search the network traffic for virus signatures. The disadvantage is that since the internal network traffic between the firewall and the users’ machines is not encrypted, an insider has the chance to sniff others’ network traffic.

- Let the firewall records all the users’ decryption keys to decrypt the network traffics. This might be less preferred because if the firewall is compromised and the attacker captures the decryption keys, all the users’ network traffic will be known to the attacker.

7.4.3 Computer Virus Encrypts Itself before Passing Through the Firewall, then Decrypts Itself at User’s Machine

Some computer virus may encrypt themselves before passing through the firewall, then decrypt themselves at the user’s machine. In this case, the computer virus actually is a
polymorphic virus, which can produce varied versions of itself.

Strictly, if the computer virus can only use one encryption scheme with a variable encryption key, it is not a polymorphic virus. Virus signature scanner may easily find this kind of computer virus because its decryption part is always the same and thus can be used as the signature. Like in the following picture, the decryption part is always the same.

<table>
<thead>
<tr>
<th>Decryption Part</th>
<th>Encrypted Body</th>
</tr>
</thead>
</table>

Even if the computer virus is a true polymorphic virus, which can use different encryption schemes, the virus signature scanner may still find this virus, provided that the virus scanner knows all signatures for each of the encryption schemes used by this virus.

Like in the following pictures, the computer virus could be like

| Decryption Part A | Encrypted Body |

or like

| Decryption Part B | Encrypted Body |

To detect such virus, the virus signature scanner needs to know that decryption part A and decryption part B are both possible signatures of the virus.

### 7.5 Limitation

Solely filtering computer virus at firewall is not enough to eliminate computer virus propagation because computer viruses could also propagate via floppy disks or modem dial-in connections, which do not go through the firewall.

Therefore, filtering computer virus at firewall should not be used as the only approach
to protect against computer viruses, though it does provide some additional protection. Other anti-virus approaches such as deploying real-time anti-virus programs to users’ client machines should also be used.
Chapter 8

Proposed Attack on Content Filtering Firewall

In previous chapters’ discussion for content filtering, there are two assumptions: 1) the network traffic is not encrypted or can be decrypted by the filtering program at the firewall; 2) the traffic is in recognised format such as HTML or can be converted to recognised format by external programs. In this chapter, we discuss the situations where any of the two assumptions do not exist.

8.1 Proposed Attack on Content Filtering Firewall

In this section, we propose an attack to get around the content filtering firewall. This attack shows the difficulty the content filtering firewall encounters when end-to-end encryption for the network traffic is used.

This attack utilises techniques including Java Applet and XML (Extensible Markup Language). Java Applet is a program written in Java language that can be sent along with a Web page to a user and executes (with restrictions) in the user’s machine. XML is used to describe, store and exchange data [10]. In XML, tags are not predefined and must be defined by the user.

8.1.1 Using Java Applet to Get Around the Content Filtering Firewall

Suppose an outsider wants to display pornographic content to internal users behind a content filtering firewall, he can first encrypt the pornographic content, put the key and
the encrypted text into a XML file, and then create a Java applet capable of decrypting such XML file. When the internal users try to access that pornographic site, the XML file and the Java applet can pass through the content filtering firewall without being blocked. Because the firewall cannot decrypt the encrypted text (actually the firewall does not even know that the XML file contains encrypted text), the firewall cannot determine that the XML file should be blocked. Then at the internal users' machines, the encrypted content will be decrypted by the Java Applet and displayed in the user's Internet browsers.

This kind of attack has been successfully implemented and tested, as shown in Figure 11.

![Applet and XML Attack Demonstration](image)

Figure 11 Applet and XML Attack Demonstration

Usually Internet browsers such as Internet Explorer or Netscape Communicator use Extensible Stylesheet Language (XSL) to display the content of XML files. But to
provide the decryption ability needed in this attack, this thesis uses Java API for XML Processing (JAXP) from Sun Microsystems, Inc [2, 3, 9, 16, 28].

In Figure 11, the Internet browser displays the file xml.html, which includes a Java Applet (as a separate file). Once started, this Java Applet first connects to the external server and retrieves the XML file ciphertext.xml then decrypts the encrypted text in it using the decryption key in the same file. The XML file ciphertext.xml contains both the encrypted text and the decryption key.

The XML document ciphertext.xml:

```xml
<?xml version="1.0"?>
<secret>
  <encryptmethod>Additive ciphers</encryptmethod>
  <key>3</key>
  <ciphertext>pbu</ciphertext>
</secret>
```

From Figure 11, it can be observed that this attack is successful. The reason that the content filtering firewall cannot prevent the string ‘sex’ being displayed to the users is because the decryption from ‘pbu’ to ‘sex’ happens in the users’ terminal machines within the internal network.

Although it is not compulsory to use XML for such kind of attack, combining Java and XML makes the attack more extensible.

### 8.1.2 Possible Solutions to this Attack

The simplest solution to this attack is to let the firewall block all Java applets. To block Java applets, the firewall can find and rewrite the `<applet>` tags in the HTML files to empty tags, so that the Java Applets will not be downloaded to the users’ machines. This solution totally eliminates this kind of attack, but the obvious disadvantage is that other legitimate Java applets are also blocked.

A better solution is to block applets from distrusted sites and to accept applets from
trusted sites only. Although it is possible that even trusted sites could contain malicious Java applets, the chance is much lower than distrusted sites.

8.2 Observation From the Attack Proposed

It can be observed from the attack proposed that content filtering firewall would fail to find undesired content if the network traffic is encrypted and cannot be decrypted at the firewall, or more generally, if the network traffic is in a form that the firewall cannot interpret correctly.

For example, if a document is in a specific format that can only be processed by certain program, the content filtering firewall cannot check the content of this document unless the firewall have detailed knowledge about this specific format, or have the tool to convert this document into recognised format to the firewall.

It should also be noted that the content filtering firewall could choose to block the network traffic if the filtering firewall finds that the network traffic is encrypted and the content cannot be examined, which has been discussed in 7.4.2. But for the attack we propose, the content filtering firewall is not even aware that the network traffic is encrypted. The content filtering firewall only knows that the XML file is a text file without undesired content in it. It is the Java applet who interprets the XML file into undesired content and displays the content at the user’s terminal machine. Internal users do not need to understand or intervene how the encryption and decryption work.
Chapter 9
Implementation of A Content Filtering Firewall

Based on the architecture proposed in Chapter 4, a content filtering firewall has been successfully implemented using the firewall package toolkit (FWTK). This chapter introduces details of the implementation. Details of the experimental network used in the implementation are in Appendix A. Details of the testing of the content filtering firewall are in Appendix B.

9.1 Introduction to Firewall Package Toolkit (FWTK)

The firewall package toolkit (FWTK) is from Trusted Information Systems, Inc [27]. It is a set of components, which can be used to create an application-level firewall system. The toolkit is chosen for this implementation because its source code is publicly available. Squid [31], a Web proxy cache that is an open-source software package, is also used to replace the FWTK's HTTP proxy.

9.2 Installation and Configuration of FWTK

The first step of the implementation is to use FWTK to build up a basic firewall system (without content filtering functionality), which includes the following procedures:

1. Clean up the bastion host
The bastion host is the machine in which the firewall resides and runs. The bastion host has two network interfaces, one connects to the internal network and the other connects to the external network. This procedure includes:

- Disabling automatic IP forwarding between the two network interfaces of the bastion host, so that the bastion host does not automatically forward network traffics between the internal and external network.
- Disabling all unnecessary services or processes on the bastion host. Since the firewall program resides and runs on the bastion host, attackers may try to compromise the firewall by attacking other more vulnerable services on the bastion host. Therefore, the bastion host should provide as less services as possible.

2. Build the executable files

Because FWTK was originally designed for running in the UNIX family platform, to build the firewall in the Linux platform used by this implementation, some adjustments to the FWTK’s source codes are needed.

3. Integrate authentication into the firewall

S/KEY is used as the authentication method. S/KEY is a one-time password system to provides secure authentication over network that are subject to eavesdropping and reply attacks.

4. Set up the access control policy used by the firewall

Access control policy is the set of rules about who can use the network services, what resources the users can use, what privileges the users can have. Network access control rules are specified in the file netperm-table in the directory /usr/local/etc of the bastion host.
5. Test

Network services through the firewall including HTTP, FTP, TELNET, RLOGIN and FINGER are tested.

9.3 Integrating the Content Filtering Functionality

The next step is to integrate content filtering functionality with the firewall. Modifying the firewall program should be very cautious, because this involves many pitfalls such as concurrency problems, consistence problems and lots of socket level operations. If such issues are not handled with careful consideration, the firewall program will not operate properly.

Following are the procedures to integrate content filtering with firewall:

1. Locate the corresponding source codes files of FWTK for different network services, such as HTTP, FTP and TELNET.
2. Modify such source codes files to add in content filtering components.
3. Restart the firewall program.

9.3.1 Refining Filtering for Different Groups

This section describes why different content filtering for different user groups is needed and presents a proposal.

9.3.1.1 The Need of Different Content Filtering for Different Groups

Usually organisations give different access controls for different groups of internal users. For example, universities may require students to use the Internet for academic purposes only; however, academic staff and network administrators may have less restriction on what sites they can access. The ability to apply different content filtering for different user groups provides greater flexibility.

9.3.1.2 A Proposal and Implementation

We propose that different content filtering for different user groups can be built using
the architecture shown in Figure 12.

![Diagram](https://via.placeholder.com/150)

**Figure 12 Different Content Filtering for Different User Groups**

All the users are required to authenticate to the firewall first before their network requests can get through the firewall. The firewall records the user's ID in a table when the user authenticates, then uses the table to determine what content filtering rules should be applied for the user's subsequent traffic.

For example, when a network administrator or staff authenticates to the firewall, content filtering may not be applied to his/her subsequent network traffic. But when a student authenticates to the firewall, content filtering may be applied to his/her network traffic.

### 9.3.2 Configuration File for Filtering

To allow the firewall administrators to specify content filtering rules such as what types of files should be checked for inappropriate content, this implementation modified the firewall program so that when the firewall is started, it firstly reads a configuration file and sets up the corresponding rules for content filtering.

In this implementation, administrators can specify the following in the configuration file:

- The strings to be checked or replaced
The types of files to be checked

The external programs to be used for extracting or re-packing non-text files

The user groups whose network traffic should be checked

The user groups whose network traffic should not be checked

The configuration file has syntax similar to the configuration files of Microsoft Windows operating systems, such as win.ini or system32.ini.

An example of the configuration file is given below:

```plaintext
[FILTERS]
sex // the string to be checked
***** // the replacing string

[TAR]
tar -xf OLDFILE // the program used to extract tar files
tar -cf NEWFILE * // the program used to repack tar files

[ZIP]
unzip OLDFILE // the program used to extract zip files
zip NEWFILE * // the program used to repack tar files

[GROUPS-NOT-TO-CHECK]
admin // traffic of users in admin group should not be checked
staff // traffic of users in staff group should not be checked

[GROUPS-TO-CHECK]
student // traffic of users in student group should be checked
```

This configuration file example specifies that the firewall should check for the string "sex" and replace it with the string "*****". It also tells the firewall that `tar` and `zip`
files should be checked and specifies the external programs used to extract or repack such files. In the configuration file, the string OLDFILE denotes the original file to be checked and the string NEWFILE denotes the file that has been checked.
Chapter 10

Conclusions and Future Works

By integrating the content filtering with firewall, we provided an alternative solution in order to overcome some problems of content filtering at terminal machines. To make the filtering more accurate, we proposed and implemented an advanced content filtering method based on text categorisation techniques to replace the basic keyword-matching filtering method. We also discussed using content filtering firewall to protect computer virus propagation through the Internet.

We also found the difficulties that the content filtering firewall encounters when end-to-end encryption for network traffic is used. We implemented and described an attack using encryption to get around the content filtering firewall.

10.1 Fulfilment of Objectives

The objectives of this thesis were briefly outlined in Chapter 1. The major contributions of this thesis can be summarised as follows.

- Design and Implementation of a content filtering firewall (in Chapter 4 and Chapter 9).

- Proposal and implementation of an advanced content filtering method based on text categorisation techniques (in Chapter 5).

- Finding the difficulties that the content filtering firewall encounters when end-to-end encryption for network traffic is in use (in Chapter 8).
10.2 Future Works

There still exist many unsolved problems in order to improve the content filtering firewall. Such problems include:

- How to correctly and effectively filter Web pages that consist of both images and texts. The method this thesis proposes only utilises texts in the Web pages to do filtering. But most Web pages contain not only texts but also images. Filtering undesired content based on images will make content filtering more complete.

- How to combine filtering at firewall and filtering at terminal machines. As discussed, filtering at firewall and filtering at terminal machines both have their advantages and problems. Combining filtering at firewall and filtering at terminal machines may help to alleviate these problems.

- How to deal with encrypted network traffic properly. We discussed that filtering firewall will fail if the traffic is encrypted (and cannot be decrypted at the firewall). The topic of dealing with encrypted traffic at firewall still needs further study.

It is hoped that this thesis can introduce further research on the topic of integrating content filtering with firewall.
Bibliography


[23] Check Point Software Technologies Ltd.

[24] CHILDREN'S INTERNET PROTECTION ACT.


[27] FireWall ToolKit (FWTK).


[29] MIMEsweeper, Clearswift Limited.


[31] Squid Web Proxy Cache.


[33] Margaret Fleck, David Forsyth, and Chris Bregler. Finding Naked People. Proceedings of


Appendix A

The Experimental Network

The work of this thesis is implemented in a private network with three computers running Linux (Red Hat 7.0) operating systems, as shown in Figure 13. Linux is chosen because it provides an excellent programming environment, which is essential for the thesis's implementation.

![Diagram of experimental network]

**Figure 13 Experimental Network Architecture**

The computer Seclab5 is the bastion host where the firewall resides and runs. It has two network interfaces. One network interface has the IP address 192.168.1.254 and connects to the computer Seclab1. The other network interface has the IP address 192.168.2.254 and connects to the computer Seclab7. Computer Seclab1 is used to simulate the internal machine and computer Seclab7 is used to simulate the external server. Seclab1 and Seclab7 are not directly connected but they are both connected to Seclab5. Therefore, all network traffic between Seclab1 and Seclab7 must go through the firewall Seclab5 first.
Appendix B

Testing of the Content Filtering Firewall

The implementation of content filtering firewall has been tested for both text files and non-text files, as shown in Figure 14 and Figure 15.

![Page Before Content Filtering](image1)

Page Before Content Filtering

![Page After Content Filtering](image2)

Page After Content Filtering

Figure 14 Test for Text Files
It can be observed from Figure 14 and Figure 15 that according to the content filtering rules, all the text files and non-text files passing through the firewall have been checked and the string "sex" has been replaced by the string "*****".
Appendix C
Source Codes of the Implementation of the Proposed Advanced Content Filtering Method in Chapter 5

/*
Author: Rongbo Du
University of Wollongong, Australia
Email: rd12@uow.edu.au
Purpose: Automatically filters HTML documents
Version: 3.0
Date: 2002/010/25
*/
import java.io.*;
import java.util.*;
import javax.swing.text.*;
import java.net.*;
import org.w3c.dom.*;
import org.w3c.dom.html.*;
import com.docuverse.dom.*;
import com.docuverse.dom.util.*;
import com.docuverse.dom.html.*;
public final class AdvancedContentFilteringMethod {
    static Vector training_documents = new Vector();
    static Vector training_set = new Vector();
    static Hashtable all_keywords = new Hashtable(20000);
    static Hashtable commonwords = new Hashtable(2000);
    static PrintWriter files_statistics;
    static PrintStream result, test_pos, test_neg;
    final static String COMMON_WORDS = "stoplist.txt";
    public final static String DELIMETER = "0123456789!@#$%^&*()_+\[\]{};':"\,./?o"
    final static int MIN_WORD_FREQUENCY = 2;
    final static int MAX_WORD_FREQUENCY = 20;
    final static int MIN_WORD_LENGTH = 3;
    final static int CONSIDER_LAYOUT = 0;
    final static double N_HIGHEST = 0.3;
    final static int USE_TF_IDF = 0;
    static int CONFIDENCE_INTERVAL_RUNS = 100;
    public static double trainingset_threshold = 1;
}
public static void main(String[] args) {
    if (args.length < 4) {
        System.out.println("training positive negative result gui\batch");
        System.exit(1);
    }
    try {
        files_statistics = new PrintWriter(new FileOutputStream("files_statistics.txt"));
        result = new PrintStream(new FileOutputStream(args[3]));
        test_pos = new PrintStream(new FileOutputStream("pos.txt"));
        test_neg = new PrintStream(new FileOutputStream("neg.txt"));
    } catch (IOException e) {
        System.out.println("Cannot open files "+args[3]);
    }
    AdvancedContentFilteringMethod wordcounter = new AdvancedContentFilteringMethod();
    wordcounter.loadCommonWords();
    dom.setProperty("sax.driver", com.docuverse.html.swing.SAXDriver);
    dom.setFactory(new com.docuverse.dom.html.HTMLFactory);
    File directory = new File(args[0]);
    if (!directory.exists()) {
        System.out.println("Wrong directory!!");
        System.exit(1);
    }
    File[] files = directory.listFiles();
    for (int i = 0; i < files.length; i++) {
        Hashtable keywords = new Hashtable(2000);
        try {
            BufferedReader in = new BufferedReader(new FileReader(files[i]));
            HTMLDocument doc = (HTMLDocument) dom.readDocument(in);
            if (doc != null) {
                long initial_words = loadFile(doc, keywords);
                long wordsaftercommonwords = 0,
                wordsafterlowfrequency = 0;
                Enumeration enum = keywords.elements();
                while (enum.hasMoreElements()) {
                }
double precomputed_training_set[] = new double[training_documents.size()];
for( int i = 0; i < training_documents.size(); i++ ) {
    
    Hashtable hash = new Hashtable( all_keywords );
    HasTable temp = (Hashtable) training_documents.get(i);
    // cal_tf_idf( temp, training_documents );
    hash.putAll( temp );
    training_set.add( hash );
    // presentData( (Hashtable) training_set.get(i) );
    // files_statistics.println( "+files[i]+"+count+"--");
    precomputed_training_set[i] = precompute( hash );
}

double final_sim_test1[]=null, final_sim_test2[]=null;
int highest_k = (int)(training_set.size() * N_HIGHEST);
for( int k = 1; k <= 2; k++ )
{
    File test_directory = new File(args[k]);
    if( !test_directory.exists() ){
        System.out.println("Wrong directory!");System.exit(1);
    }
    File[] test_files = test_directory.listFiles();
double final_sim[] = new double[test_files.length];
    for( int i = 0; i < test_files.length; i++ )
    {
        Hashtable test_doc_keywords = new Hashtable();
        Enumeration e1 = all_keywords.elements();
        while( e1.hasMoreElements() )
        {
            WordCountStruct w =
            new WordCountStruct( (WordCountStruct) e1.nextElement() );
            test_doc_keywords.put( w.getWord(), w );
        }
        try{
            BufferedReader in =
            new BufferedReader(new FileReader(test_files[i]));
            HTMLDocument test_doc =
            (HTMLDocument)dom.readDocument(in);
            if( test_doc != null )
            {
                wordcounter.loadFile( test_doc, test_doc_keywords );
                Enumeration enum = test_doc_keywords.elements();
                while( enum.hasMoreElements() )
                {
                    WordCountStruct w =
                    (WordCountStruct) enum.nextElement();
                    if( w.getCount() < MIN_WORD_FREQUENCY && !all_keywords.containsKey(w.getWord()) )
                        test_doc_keywords.remove(w.getWord());
                }
            } else
            {
            }
        } catch( Exception e ) {
            System.out.println("Error reading file");
        } 
        if( CONSIDERLAYOUT == 1 )
            Webclassifier.considerLayout(
            test_doc, test_doc_keywords, commonwords );
        double precomputed_value_for_test_doc =
        precompute(test_doc_key, words);
    }
}
double simi_distance = 0;
double temp[] = new double[training_set.size()];
for( int j = 0; j < training_set.size(); j++ )
{
    temp[j] = measureDistance( (Hashtable) test_doc_keywords,
                   precomputed_value_for_test_doc,
                   (Hashtable) training_set.get(j),
                   precomputed_training_set[j] );
    sum_distance += temp[j];
}
final_sim[i] = get_final_sim( temp, highest_k );
if( k == 1 )
    test_pos.println(test_files[i]+","+final_sim[i]);
else
    test_neg.println(test_files[i]+","+final_sim[i]);
}
in.close();
}
catch(IOException ioe){
    System.out.println("Exception in reading test files.");
}
}
Arrays.sort(final_sim);
if( k == 1 ){
    final_sim_test1 = new double[final_sim.length];
    for( int c=0;c<final_sim.length;c++)
        final_sim_test1[c] = final_sim[c];
}
else{
    final_sim_test2 = new double[final_sim.length];
    for( int c=0;c<final_sim.length;c++)
        final_sim_test2[c] = final_sim[c];
}
}
cal_confidence_interval();
result.close();
test_pos.close();
test_neg.close();
files_statistics.close();
//if( args[4].equals("gui") )
//gui(wordcounter,dom);
private static void cal_cofididence_interval() {
    double pos[] = null, neg[] = null;
    try {
        RandomAccessFile pos_file = new RandomAccessFile("pos_input.txt", "r");
        String s = pos_file.readLine();
        pos = new double[Integer.parseInt(s)];
        int i = 0;
        while ((s = pos_file.readLine()) != null) {
            pos[i++] = Double.parseDouble(s);
        }
        RandomAccessFile neg_file = new RandomAccessFile("neg_input.txt", "r");
        s = neg_file.readLine();
        neg = new double[Integer.parseInt(s)];
        i = 0;
        while ((s = neg_file.readLine()) != null) {
            neg[i++] = Double.parseDouble(s);
        }
    } catch (Exception e) { System.out.println("Exception in reading file: "+e); }

    PrintWriter output = null;
    try {
        output = new PrintWriter(new FileOutputStream("confidence_interval.txt");
    } catch (IOException e) {
        System.out.println("Cannot open confidence_interval file!!");
    }

    int run_times[] = { 10, 20, 30, 40, 50, 60, 70, 100, 150, 200, 300, 500, 700, 1000, 1500, 2000 };
    for (int r = 0; r < run_times.length; r++) {
        CONFIDENCE_INTERVAL_RUNS = run_times[r];
        output.println("t--------------- Number of runs: +CONFIDENCE_INTERVAL_RUNS+"--");
        double pos_t = 2.048; // 5%
        double neg_t = 2.045;
        //double pos_t = 2.0025; //10%
        //double neg_t = 2.0017; //10%
        //double pos_t = 1.981; //20%
        //double neg_t = 1.981; //20%
    }

}
//double pos_t = 1.974; //30%
//double neg_t = 1.974; //30%
double threshold[] = {0.10, 0.12, 0.14, 0.16, 0.18, 0.20, 0.22};
int pos_test_size = (int) (pos.length * 0.2)+1;
int neg_test_size = (int) (neg.length * 0.2)+1;

int [CONFIDENCE_INTERVAL_RUNS][threshold.length];
int [CONFIDENCE_INTERVAL_RUNS][threshold.length];
double pos_test_mean, neg_test_mean;
int pos_test_sum = 0, neg_test_sum = 0;

double pos_test[] = new double[pos_test_size];
double neg_test[] = new double[neg_test_size];
double pos_temp[] = new double[pos.length];
double neg_temp[] = new double[neg.length];
Random random = new Random();

for( int i = 0; i < CONFIDENCE_INTERVAL_RUNS; i++ ){
    for( int c=0;c<pos.length;c++)
        pos_temp[c] = pos[c];
    for( int c=0;c<neg.length;c++)
        neg_temp[c] = neg[c];

    for( int j = 0; j < pos_test_size;){
        int temp = random.nextInt(pos.length);
        if( pos_temp[temp] != -1 ){
            pos_test[j] = pos_temp[temp];
            pos_temp[temp] = -1;
            j++;
        }
    }
    for( int j = 0; j < neg_test_size;){
        int temp = random.nextInt(neg.length);
        if( neg_temp[temp] != -1 ){
            neg_test[j] = neg_temp[temp];
            neg_temp[temp] = -1;
            j++;
        }
    }
}
Arrays.sort( pos_test );
Arrays.sort( neg_test);

for( int t = 0; t < threshold.length; t++ ){
    NUM_OF_BLOCKED[i][t] = 0;
    NUM_OF_OVERBLOCKED[i][t] = 0;
    for( int j = 0; j < pos_test_size; j++ ){
        if( pos_test[j] >= threshold[t] ){
            NUM_OF_BLOCKED[i][t] =
            pos_test_size - j;
            break;
        }
    }
    for( int j = 0; j < neg_test_size; j++){
        if( neg_test[j] >= threshold[t] ){
            NUM_OF_OVERBLOCKED[i][t] =
            neg_test_size - j;
            break;
        }
    }
}

for( int t = 0; t < threshold.length; t++ ){
    pos_test_sum = 0;
    neg_test_sum = 0;
    for( int i = 0; i < CONFIDENCE_INTERVAL_RUNS; i++ ){
        pos_test_sum += NUM_OF_BLOCKED[i][t];
        neg_test_sum += NUM_OF_OVERBLOCKED[i][t];
    }
    pos_test_mean = (double)(pos_test_sum) *100/(CONFIDENCE_INTERVAL_RUNS*pos_test_size);
    neg_test_mean = (double)(neg_test_sum) *100/(CONFIDENCE_INTERVAL_RUNS*neg_test_size);
    double a = 0, b = 0, temp = 0;
    for( int j = 0; j < CONFIDENCE_INTERVAL_RUNS; j++){
        temp = (((double)(NUM_OF_BLOCKED[j][t])*100)/pos_test_size
        - pos_test_mean);
        a += temp*temp;
    }
    for( int j = 0; j < CONFIDENCE_INTERVAL_RUNS; j++)

temp = ((double)(NUM_OF_OVERBLOCKED[j][t] * 100 / neg_test_size) - neg_test_mean); b += temp * temp;

double sterr_a = pos_t * Math.sqrt(a / (CONFIDENCE_INTERVAL_RUNS * (CONFIDENCE_INTERVAL_RUNS - 1))); double sterr_b = neg_t * Math.sqrt(b / (CONFIDENCE_INTERVAL_RUNS * (CONFIDENCE_INTERVAL_RUNS - 1)));

output.println((pos_test_mean - sterr_a) + "\t" + (pos_test_mean + sterr_a) + "\t" + (neg_test_mean - sterr_b) + "\t" + (neg_test_mean + sterr_b) + "\t" + threshold[t]);
result.println(pos_test_mean + "\t" + neg_test_mean + "\t" + threshold[t]);

private static double get_final_sim(double[] input, int k) {
    double[] temp = new double[input.length];
    for (int i = 0; i < input.length; i++)
        temp[i] = input[i];

    double sum_distance = 0;
    for (int i = 0; i < k; i++)
    {
        int max_index = 0;
        for (int j = 0; j < temp.length; j++)
        {
            if (temp[max_index] < temp[j])
            {
                max_index = j;
            }
        }
        sum_distance += temp[max_index];
        temp[max_index] = -1;
    }

    double result = sum_distance / k;
    if (Double.isNaN(result))
    {
        return 0;
    }
    else
    {
        return result;
    }
}
private static void cal_tf_idf(Hashtable hash, Vector training_documents) {
    double LOG_E_2 = Math.log(2);
    int N = training_documents.size();
    Enumeration e = hash.elements();
    while (e.hasMoreElements()) {
        WordCountStruct w = (WordCountStruct)e.nextElement();
        if (w.getCount() != 0) {
            String word = w.getWord();
            int num_of_docs_with_word = 0;
            for (int i = 0; i < training_documents.size(); i++) {
                Hashtable temp = (Hashtable) training_documents.get(i);
                WordCountStruct wl = (WordCountStruct) temp.get(word);
                if (wl != null) {
                    if (wl.getCount() != 0)
                        num_of_docs_with_word++;
                }
            }
            double d = w.getCount() * Math.log(N*1.0/num_of_docs_with_word) / LOG_E_2;
            if (!Double.isInfinite(d))
                w.updateWeight(d);
        }
    }
}

private static double precompute(Hashtable hash) {
    double sum = 0, x;
    Enumeration e = hash.elements();
    while (e.hasMoreElements()) {
        if (USE_TF_IDF == 1)
            x = ((WordCountStruct)e.nextElement()).getWeight();
        else
            x = ((WordCountStruct)e.nextElement()).getCount();
        sum += (x * x);
    }
    double result = Math.sqrt(sum);
if (Double.isNaN(result))
    return 0;
else
    return result;
}

private static double measureDistance(
    Hashtable test_doc_keywords, double precomputed_value_for_test_doc,
    Hashtable a_training_document, double precomputed_value_for_training_doc)
{
    if (USE_TF_IDF == 1)
    {
        double sum_xy = 0, sum_y2 = 0, y;

        Enumeration e = test_doc_keywords.elements();
        while (e.hasMoreElements())
        {
            WordCountStruct w1 = (WordCountStruct)e.nextElement();

            WordCountStruct w2 = (WordCountStruct)a_training_document.get(w1.getWord());
            if (w2 != null)
            {
                y = w2.getWeight();
                sum_xy += (w1.getWeight() * y);
                sum_y2 += (y * y);
            }
        }
        return sum_xy / (precomputed_value_for_test_doc * Math.sqrt(sum_y2));
    }
    else
    {
        long sum_xy = 0, sum_y2 = 0, y;

        Enumeration e = test_doc_keywords.elements();
        while (e.hasMoreElements())
        {
            WordCountStruct w1 = (WordCountStruct)e.nextElement();

            WordCountStruct w2 = (WordCountStruct)a_training_document.get(w1.getWord());
            if (w2 != null)
double result = sum_xy / (precomputed_value_for_test_doc * precomputed_value_for_training_doc);

if (Double.isNaN(result))
    return 0;
else
    return result;

Euclidean distance method

long sum = 0;
Enumeration e1 = test_doc_keywords.elements();
while (e1.hasMoreElements())
{
    WordCountStruct w1 = (WordCountStruct)e1.nextElement();
    int x = w1.getCount();
    int y = 0;
    WordCountStruct w2 = (WordCountStruct)a_training_document.get(w1.getWord());
    if (w2 == null)
        y = 0;
    else
        y = w2.getCount();
    sum += (x - y) * (x - y);
}
return Math.sqrt(sum);

private static void loadCommonWords()
{
    StringTokenizer token = null;
    BufferedReader buf = null;
    try
    {
        buf = new BufferedReader(new FileReader(COMMON_WORDS));

        String line;
        while ((line = buf.readLine()) != null)
        {
            token = new StringTokenizer(line);
            while (token.hasMoreTokens())
            {
                String word = token.nextToken();
                if (!isStopWord(word))
                    document.addWord(word);
            }
        }
        buf.close();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
}
catch(FileNotFoundException fne){
    System.out.println("Couldn't open " + COMMON_WORDS);
    System.exit(1);
}

while(true)
{
    try{
        String s = buf.readLine();
        if( s == null )
            break;
        token = new StringTokenizer(s,DELIMETER);
    }
    catch (IOException e){
        System.out.println("Error reading from input file.");
        System.exit(1);
    }

    while( token.hasMoreTokens() )
    {
        String s = new String(token.nextToken()).toLowerCase();
        WordCountStruct temp = (WordCountStruct) commonwords.get( s );
        if( temp != null ){
            temp.countPlusPlus();
        }
        else{
            WordCountStruct Word = new WordCountStruct();
            Word.updateCount(1);
            Word.updateWord( s );
            commonwords.put( s, Word );
        }
    }
}

try{
    buf.close();
}
catch(IOException e){
    System.out.println("Error closing input file.");
    System.exit(1);
}
private static long loadFile(HTMLDocument doc, Hashtable hash)
{
    BasicHTMLBodyElement body = (BasicHTMLBodyElement) doc.getBody;
    NodeList list = DOMUtil.getNodesByFilter(body, ContentFilter.getDefaultInstance());
    TextBuffer buffer = new TextBuffer();
    for (int i = 0; i < list.getLength(); i++)
    {
        Node node = list.item(i);
        if (node != null)
        {
            String parent_name = node.getParentNode().getNodeName();
            if (parent_name.equals("script") || parent_name.equals("comment"))
                buffer.append(node.getNodeValue());
        }
    }
    // process meta tag
    list = doc.getElementsByTagName("meta");
    for (int i = 0; i < list.getLength(); i++)
    {
        Node node = list.item(i);
        if (node != null)
        {
            NamedNodeMap attribute = node.getAttributes();
            if (attribute != null)
            {
                Node n1 = attribute.getNamedItem("name");
                if (n1 != null)
                {
                    String name = n1.getNodeValue();
                    if (name != null)
                    {
                        if (name.equalsIgnoreCase("description") ||
                            name.equalsIgnoreCase("keywords") ||
                            name.equalsIgnoreCase("description"))
                        {
                            Node n2 = attribute.getNamedItem("content");
                            if (n2 != null)
                            {
                                String content = n2.getNodeValue();
                                if (content != null)
                                {
                                    buffer.append("+
                                    buffer.append("+
                                    buffer.append("+
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
// process img tag
/*list = doc.getElementsByTagName("img");
for (int i = 0; i < list.getLength(); i++)
{
    Node node = list.item(i);
    if (node != null)
    {
        NamedNodeMap attribute = node.getAttributes();
        if (attribute != null)
        {
            Node n1 = attribute.getNamedItem("alt");
            if (n1 != null)
            {
                String alt = n1.getNodeValue();
                if (alt != null)
                    buffer.append(" + alt");
            }
        }
    }
}
*/

// process title tag
buffer.append(" + doc.getTitle()");
buffer.append(" + doc.getTitle()");
buffer.append(" + doc.getTitle()");

StringTokener token =
    new StringTokenizer(buffer.toString(), DELIMETER);

long initial_words = 0;
while (token.hasMoreTokens())
{
    String s = new String(token.nextToken()).toLowerCase();
    initial_words++;
    if (s.length() < MIN_WORD_LENGTH || commonwords.containsKey(s))
        continue; // ignore short words and common words

    WordCountStruct temp = (WordCountStruct) hash.get(s);
}
if (temp != null) {
    temp.countPlusPlus();
}
else{
    WordCountStruct Word = new WordCountStruct();
    Word.updateCount(1);
    Word.updateWord(s);
    hash.put(s, Word);
}
}
return initial_words;
}
private static void presentData(Hashtable hash) {
    long total_words = 0;
    LinkedList list = new LinkedList();
    Enumeration enum = hash.elements();
    while (enum.hasMoreElements()) {
        WordCountStruct w =
            new WordCountStruct((WordCountStruct) enum.nextElement());
        list.add(w);
    }
    Collections.sort((java.util.List)list, (Comparator)new WordComparator());
    ListIterator i = list.listIterator(0);
    while (i.hasNext()) {
        WordCountStruct tmp = (WordCountStruct) i.next();
        total_words += tmp.getCount();
    }
    ListIterator i = list.listIterator(0);
    while (i.hasNext()) {
        WordCountStruct tmp = (WordCountStruct) i.next();
        files_statistics.println(tmp.toString() + "\t" + ((float) tmp.getCount() * 100 / total_words) + ";
    }
}
public static double similarity(String input) {

double sum_distance = 0;
Enumeration e = all_keywords.elements();
while( e.hasMoreElements() )
    ((WordCountStruct) e.nextElement()).updateCount( 0 );

Hashtable test_doc_keywords = new Hashtable(all_keywords);

try{
    URL url = new URL(input);
    URLConnection conn = url.openConnection();

    if( input.indexOf("uow.edu.au") == -1 ) // not local web site
        {  
        String authentication = "Basic " + new sun.misc.BASE64Encoder().encode("rd12:type your password here".getBytes());

        System.getProperties().put("proxySet", "true");
        System.getProperties().put("proxyHost", "proxy.uow.edu.au");
        System.getProperties().put("proxyPort", "8080");
        conn.setRequestProperty("Proxy-Authorization", authentication);
    }

    HTMLDocument test_doc = (HTMLDocument)dom.readDocument(conn.getInputStream());
    if( test_doc != null )
        {
            loadFile( test_doc, test_doc_keywords );
            Enumeration enum = test_doc_keywords.elements();
            while( enum.hasMoreElements() )
            {
                WordCountStruct w = new WordCountStruct( (WordCountStruct) enum.nextElement() );
                if( w.getCount() < MIN_WORD_FREQUENCY && !all_keywords.containsKey(w.getWord()) )
                    // remove low frequency words
                    test_doc_keywords.remove( w.getWord() );
            }
        }

    if( CONSIDERLAYOUT == 1 )
}
double precomputed_value_for_test_doc = precompute(test_doc_keywords);
for (int j = 0; j < training_set.size(); j++) {
    sum_distance +=
        measureDistance((Hashtable) test_doc_keywords,
                        precomputed_value_for_test_doc, (Hashtable) training_set.get(j), 0);
}
} catch(Exception e1) {
    System.out.println("Exception removing HTTP header");
    System.out.println(e1);
}
return sum_distance / training_set.size();

private static void gui(AEAdvancedContentFilteringMethod filter, DOM dom)
{
    Gui app = new Gui(filter);
}

final class WordCountStruct{
    private String word;
    private int count;
    private double weight;
    WordCountStruct (WordCountStruct another)
    {
        word = new String( another.word );
        count = another.count;
        weight = another.weight;
    }
    WordCountStruct()
    {
        word = null;
        count = 0;
        weight = 0;
    }
    public void countPlusPlus()
    {
        count++;
    }
    public void updateWeight(double n)
weight = n;
}
public void updateCount( int n ){
    count = n;
}
public double getWeight( ){
    return weight;
}
public int getCount( ){
    return count;
}
public void updateWord( String newword ){
    word = new String( newword );
}
public String getWord( ){
    return word;
}
public String toString( ){
    if( word.length() <= 8 )
        return new String(word + 
                           \t\t\t\tcount + \t\tweight);
    else 
        return new String(word + \t\tcount + \tweight);
}
}
final class WordComparator implements Comparator {
{
    public int compare( Object obj1, Object obj2 )
    {
        WordCountStruct a = (WordCountStruct) obj1;
        WordCountStruct b = (WordCountStruct) obj2;
        int a_count = a.getCount();
        int b_count = b.getCount();
        if( a_count > b_count )
            return 1;
        else if( a_count < b_count )
            return -1;
        else
            return a.getWord().compareTo( b.getWord() );
    }
}
Appendix D
Source Codes of the Implementation of the Proposed Attack on Content Filtering Firewall in Chapter 8

The implementation of the proposed attack on content filtering firewall is a Java applet written using Java API for XML Processing (JAXP) from Sun Microsystems, Inc.

/*
   Author:  Rongbo Du
             University of Wollongong, Australia
   Email:    rd12@uow.edu.au
   Purpose:  Using Java API for XML Parsing to develop a Java applet that can decrypt and display the ciphertext inside XML files.
   Version:  3.0
   Date:     2001/04/30
*/

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.JEditorPane;
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;
import javax.swing.tree.*;
import javax.swing.event.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import java.io.File;
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;
import javax.swing.JEditorPane;
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;
import java.util.*
import javax.swing.*
import java.net.*
import java.io.*
import java.applet.*;

public class xmlApplet extends Applet {
    private DocumentBuilderFactory factory;
    static Document document;
    final String firstPage = "ciphertext.xml";

    public void init() {
        super.init();
        setBackground( new Color( 221, 221, 255));
        factory = DocumentBuilderFactory.newInstance();
        getThePage( firstPage);
    }

    private void getThePage( String location) {
        if( location.endsWith( ".xml")) {
            org.w3c.dom.NodeList list;
            org.w3c.dom.Node node;
            try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                document = builder.parse( location );
            } catch (SAXException sxe){
                Exception x  = sxe;
                if (sxe.getExceptionO != null)
                    x = sxe.getExceptionO;
            } catch (SAXException sxe){
                Exception x  = sxe;
                if (sxe.getExceptionO != null)
                    x = sxe.getExceptionO;
        }
    }

}
x.printStackTrace();
} catch (ParserConfigurationException pce){
    pce.printStackTrace();
} catch (IOException ioe){
    ioe.printStackTrace();
}

list = document.getElementsByTagName("encryt_method");
node = list.item(0);
String encryt_method = node.getFirstChild().getNodeValue();
list = document.getElementsByTagName("key");
node = list.item(0);
int key = Integer.parseInt( node.getFirstChild().getNodeValue() );

list = document.getElementsByTagName("ciphertext");
node = list.item(0);
String ciphertext = node.getFirstChild().getNodeValue();

String output = "Since this is an xml file with ciphertext in it, the program will decrypt the ciphertext then display the corresponding plaintext!!\n\n";
String plaintext = "";
if( encryt_method.equals( "Additive ciphers" ) ) {
    for( int i = 0; i < ciphertext.length(); i++ )
        plaintext += (char)(((int)ciphertext.charAt(i))+key);
    output += "\tThe plaintext is: " + plaintext + "\n";
    output += "\tThe encrypt method is: " + encryt_method + "\n";
    output += "\tThe key is: " + key + "\n";
}
else {
    output += "\n\n\tEncryption method not supported!!\n\n";
}
output += "XML file processed by JAVA browser developed by Rongbo Du.";