Transactional Behaviour Based Spam Detection

By

Thomas Choi

A thesis submitted to
The Faculty of Graduate Studies and Research
in partial fulfilment of
the degree requirements of
Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for
Computer and Electrical Engineering

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
May 2007

Your file Votre référence
ISBN: 978-0-494-26984-8
Our file Notre référence
ISBN: 978-0-494-26984-8

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Transactional Behaviour Based Spam Detection

Submitted by **Thomas Choi**
in partial fulfilment of the requirements for the degree of
Master of Applied Science in Electrical Engineering

Adrian D.C. Chan, Thesis Supervisor, Department of Systems and Computer Engineering

---

Tony White, Thesis Co-Supervisor, School of Computer Science

Victor Aitkin, Chair, Department of Systems and Computer Engineering

Carleton University
2007

Abstract

In this thesis, we propose a novel spam zombie detection method by presenting a new real-time machine learning based spam filtering technique that uses the Spamhaus blacklist to learn SMTP transactional behaviour of spam zombies. Specifically our technique was implemented as a single layer perceptron plug-in that learns the behaviour of spam zombies and makes decisions as to whether an incoming source is likely to send spam or not. We also created and integrated a reverse DNS module into our design to prevent spammers from forging legitimate domains and making it difficult for them to overcome our technique. Our technique was deployed on a large corporate network, where we were able to demonstrate that our technique was able to generalize the Spamhaus list. In addition, this was accomplished without any increase in false positives.

Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Carleton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature_____

I further authorize Carleton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature_____

Acknowledgements

I wish to thank my thesis supervisors Dr. Adrian Chan and Dr. Tony White for their encouragement, guidance and support. I am truly grateful for having two extremely knowledgeable and understanding supervisors who have each taken time out of their busy schedules to review, edit and provide valuable feedback on my thesis. In 2005, I switched my research topic from bio-medical engineering to digital security engineering. As part of this switch, Dr. White had agreed to co-supervise my work with Dr. Chan. I thank Dr. White for giving me the opportunity to study digital security with some extremely exceptional individuals in the Carleton Digital Security and Carleton Intrusion Detection Research groups. I also thank Dr. Chan for his assistance in this transition and for his patience and advice.

I also wish to thank my manager Pat Cottrell for allowing me to use Nortel's corporate resources and network to develop and test the new spam filtering technique described in this thesis. Finally, I wish to thank my mentor Chris Lewis whose guidance and encouragement led me to develop an extremely strong interest in the field of digital security.

Table of Contents

List of Tables

List of Figures

List of Equations

Chapter 1: Introduction

1.1 Motivation

E-mail is an efficient form of communication that has become widely adopted by both individuals and organizations. Today, more and more people are relying on e-mail to connect them with their friends, family, colleagues, customers and business partners. Unfortunately, as e-mail usage has evolved, so too has its threats. In particular, spam, which is also known as unsolicited bulk e-mail, has become an increasingly difficult threat to detect and is being delivered in incredibly high volumes. For example, according to MessageLabs, spam accounted for 67% of all e-mail traffic in October 2006, up from 57% the same time a year before [1].

Spam is a serious problem that potentially threatens the existence of e-mail services. In particular, it is a now a non-trivial task to find legitimate e-mails in an e-mail inbox cluttered with spam. Spam is also an expensive problem that costs service providers and organizations billion of dollars per year in lost bandwidth [2]. Further to the bandwidth cost, it is also estimated that each piece of spam costs an organization one dollar in lost employee productivity [2]. There are also published reports, which suggest that spam has resulted in lost opportunity costs of several billions of dollars [3] because of organizations that have lost faith in the security industry's ability to fight this problem.

1.2 Thesis Objectives

Significant investment and advances have been made within the past decade on anti-spam research but even with all this work, the spam problem appears to be getting worse [1]. One reason is that existing spam filtering techniques have been directed towards detection based filtering techniques such as content and network based filtering. Content filtering is based upon a set of static rules generated by analyzing the content of previously detected spam whereas network based filtering, such as DNS blacklists, list sources that have been previously detected sending spam. Spammers have proven their ability to overcome such techniques by modifying the content of their spams such that the spams can bypass existing content filtering techniques [4] or by creating a fresh supply of spam zombies to send spam for the purpose of bypassing network filtering techniques. With industry metrics indicating that approximately 80% of all spam comes from spam zombies [5], there is a need to improve existing spam filtering techniques to detect a wide range of spam from spam zombies.

The objective of this thesis is to introduce a new approach to spam filtering by first investigating the SMTP transactional behaviour differences between legitimate mail sources and illegitimate mail sources such as spam zombies. Next, our objective is to introduce a new machine learning filtering technique that can continuously learn efficiently in real-time and also identify differences in mail sending behavior between a legitimate and illegitimate-mail source. Specifically, we wish to make use of existing detection based techniques as a learning mechanism to produce a solution designed to identify illegitimate mail servers, such as spam zombies, and filter mail from said

sources. Our final objective is to conduct experimentation to prove that our new filtering technique provides an advantage over existing detection based filtering.

1.3 Contributions

This thesis makes two main contributions to the state of the art in research in spam filtering:

- A comparative behavioural analysis of the SMTP transaction between a legitimate mail server and a spam zombie according to the five-step SMTP transactional model defined in Chapter 4.
- A new spam filtering technique based upon our comparative SMTP transactional behaviour analysis that is able to learn in real-time using a single layer perceptron. The filtering technique is able to efficiently generalize the Spamhaus list in real-time, detect spam zombies that are not already being detected by said list and is difficult for the spammer to overcome.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of spam as well as a four-step model to describe how spam sources have evolved. We then use the model to explain how spam zombies have become a preferred method of sending spam.

Chapter 3 describes current efforts at combating spam and the state of the art of spam filtering. For each effort or filtering technique, we examine their shortcomings with respect to spam prevention or filtering.

Chapter 4 gives a general overview of Simple Mail Transfer Protocol (SMTP) transactional behavior. Specifically, Section 4.2 describes a five-step SMTP transactional model between legitimate mail servers. Section 4.3 provides a comparative analysis between two mechanisms of notifying the sender when an SMTP transaction has failed.

Chapter 5 applies the five-step SMTP transactional model presented in Chapter 4 to determine the differences in SMTP transactional behaviour between legitimate mail servers and spam zombies.

Chapter 6 analyzes the behavioural differences found in Chapter 5 and describes how these findings were used as part of the design of our new filtering technique. In particular, Section 6.4 describes the design of our filtering technique and section 6.5 illustrates how our technique was implemented.

Chapter 7 presents the test results from our design running in a live corporate mail infrastructure.

Finally, Chapter 8 presents our conclusions and discusses several possibilities for extending our work in the future.

Chapter 2: Background - SPAM

2.1 Introduction

In this Chapter, we begin by presenting an overview of spam. We then examine the e-mail delivery process and subsequently define a four-step model that describes the evolution of spam sources used to send spam. The model is then applied to explain the abundance of spam zombies currently used to send spam. In particular, we identify and discuss trends in the changing spam sources that have made spam zombies a preferred source for sending spam where approximately 80% of all spam comes from spam zombies [5].

2.2 What is Spam?

In this thesis, we define spam as all email the user does not want to receive and has not asked to receive [7]. Such unsolicited e-mails come in several forms such as advertisements [8], scams [9], and viruses [10].

2.2.1 Why does Spam Exist?

Spam exists because the recipients allow spammers to make spamming a profitable business. First, compared to physical bulk mailings, sending e-mail advertisements is about 100 times cheaper [11] because it costs the sender far less to send their messages

via e-mail than through regular mail since it is the recipient or their carrier who is responsible for having to pay for most of the spam delivery cost. For example, assuming that a company receives 100 000 spams per day and it takes their servers 10 seconds to identify and discard a message, a total of 278 hours of connect time is being used by their servers to filter spam. With growing volumes of spam, the receiver and their carrier will require more network and disk resources to cope with the extra load, which will result in further costs to the receiver. In contrast, the spammer simply requires a broadband Internet connection to send their spam. Not only do recipients cover most of the costs for delivering spam but a small number of them are known to purchase goods advertised in the spams or fall for scam or virus spam attacks thus generating a stream of income for the spammer [12].

2.2.2 Why is Spam Disruptive?

Spam is disruptive not only because it is unsolicited but also because spammers send it in bulk such that 67% of all e-mail is now defined as spam [1]. With so much e-mail being spam, end users will tend to agree that it is becoming more difficult and annoying for them to find legitimate e-mail in a sea of spam. In addition to being disruptive, spam is also considered offensive to users as well. For example, commercial spam may include advertisements of pornographic or sexual nature [13], which may be considered offensive to some users. As we will illustrate in the next section, in spite of the many efforts to block spam, spam continues to be a problem because for each major new filtering technique, spammers have found ways to evolve their spam to defeat it.

2.2.3 Where does Spam come from?

Spam sources have evolved from simplistic sources such as a spammer's e-mail account to more sophisticated means such as the use of third party relays, proxies and compromised machines. The motivation behind this evolution comes from anti-spam efforts such as anti-spam laws, broadband service provider policies and filtering techniques [14]. However, in spite of the changing spam sources, very little research has been conducted on spam sources. In the following sections, we present a model that highlights how spam and e-mail is delivered from a sending source and how spam sources have evolved.

2.2.4 Spam and e-mail Delivery Process

The e-mail delivery process can be best explained with the help of Figure 2-1, which illustrates a step-by-step process on how e-mail is delivered. The main elements of Figure 1-1 are the sending client, sending server, receiving server, receiving client, e-mail/spam and a DNS server used by sending server. In this section, we will describe the basics of e-mail delivery but for a more in-depth look at the delivery process along with the terminology used in this thesis, please refer to [16].

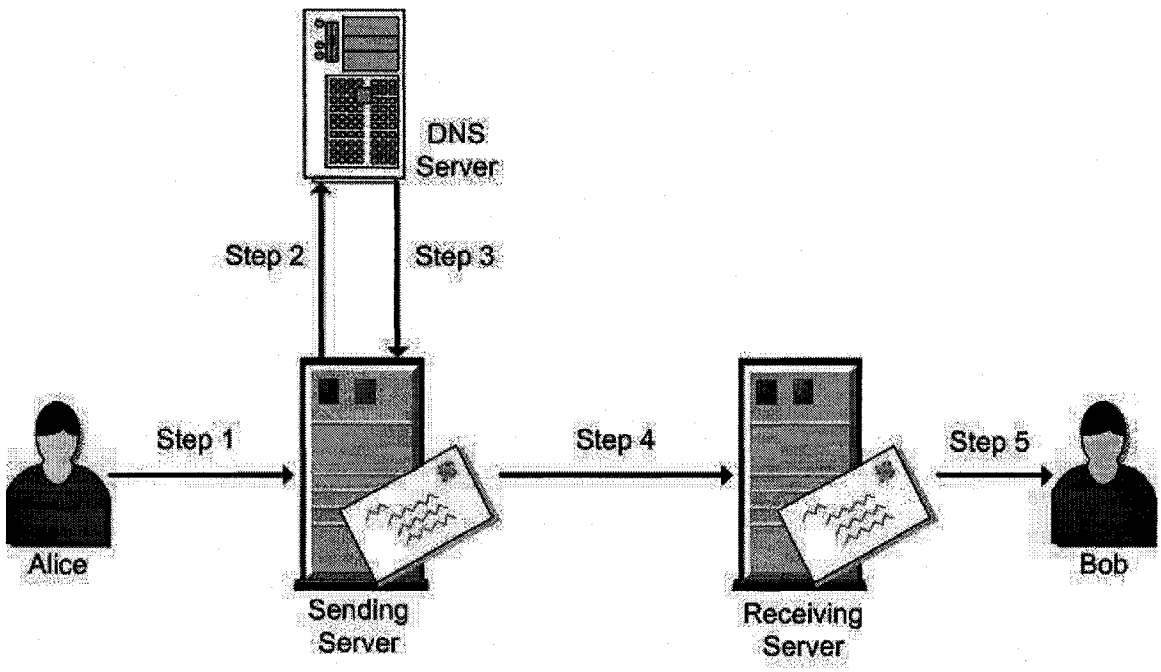

Figure 2-1: E-mail Delivery Process

For purposes of this example, we will assume that the sender's domain is `sender.com` and the receiver's domain is `receiver.com`. In order the sender Alice (`alice@sender.com`) to send e-mail to Bob (`bob@receiver.com`), the sender's (Alice) server must determine the receiver's (Bob) inbound mail server. This is done via a Domain Name Server (DNS) query. Specifically, Step 1 of the e-mail delivery process requires the sender, Alice, to compose a message and request its e-mail server to deliver the message to the recipient Bob. In Step 2, the sending server asks the DNS server to lookup the DNS record of type MX of the receiving server. In Step 3, the DNS server responds to the sending server with the Internet Protocol (IP) address of the receiving server. In Step 4, the sending server connects to the receiving server and performs a Simple Mail Transfer Protocol (SMTP) transaction with the receiving server to deliver

the Alice's message to Bob. In Step 5, the receiving server delivers Alice's message to Bob's e-mail client. The SMTP transaction performed between the sending and receiving servers is described in greater detail and modeled in Chapter 4.

2.3 Evolution of Spam Sources

In this section, we present a four-step model that identifies how spam sources have evolved in response to anti-spam measures. Our intent is to use this model to help identify the key features that spammers look for in a spam source.

1. *Spammers are moving away from having to use their own sources to send spam* [14].

   One of the first sources of spam was the spammer's Internet Service Provider's (ISP) mail servers where spammers would send mail from their ISP assigned account. Spammers quickly moved away from these sources once ISPs adopted acceptable usage policies and terminated accounts of those who abused their Internet privileges.

2. *Spammers are moving away from sources that do not protect their identity* [14].

   Open relays are mis-configured mail servers that allow any non-authenticated user on the Internet to relay mail to any domain. Open relays were once a heavily utilized source for spam. In recent years however, spammers moved away from these sources because mail server software companies were issuing configuration patches to prevent mail servers from being abused in this fashion and also because spams sent from these sources could be traced back to the spammer's machine [15].

3. *Spammers are moving away from sources that require a non-trivial amount of manual labour* [14].

Spammers are also known to abuse free-mail mailing sources such as those from yahoo, hotmail and gmail, by creating bogus free-mail accounts and using them to send spam. In recent years, free-mail providers have introduced a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) [17], which uses challenge-response type tests to determine whether the user is a human. CAPTCHAs help prevent automated creation of accounts thus making free-mail sources less attractive to spammers.

There are however, a few spammers today such as the Nigerian 419 scam spam gang that still use freemail sources to send spam. Specifically, they hire people from poorer parts of the world to manually create and send spam from freemail accounts, thus overcoming the graphical CAPTCHA technique [17]. The hired workers are also able to protect their identity by performing their work on public machines such as those in an Internet café.

In this type of scam, an e-mail is sent from an alleged 'official' who needs to transfer millions of dollars in funds from his/her foreign nation to a bank account in the recipient's home country. For reasons outlined in the e-mail, the 'official' cannot conduct the transfer him/herself and as such, an offer is made whereby the recipient will receive a share of the funds. Although the offer will sound enticing and the need to transfer these funds overseas will sound legitimate, responding to such scams is costly, dangerous and in some cases life threatening [18]. An

example of what a Nigerian 419 scam e-mail looks like is provided in Appendix A. As a whole, free-mail is not a heavily utilized source for sending spam due to the high cost involved in hiring people to do the work and the fact that the amount of spam that can be generated is dependent upon how fast the hired workers can send e-mail. As such, spammers are moving away from sources that require a non-trivial amount of manual labour.

4. *Spammers are moving away from using third party sources that are difficult to find* [14].

   Open proxies are typically non e-mail servers such as web servers, proxy servers, socks servers that have been abused to send spam to third parties. One way such servers can be abused is through accidental mis-configuration. For example, spammers could connect to the proxy and have it connect to an e-mail server to send spam. An advantage of using open proxies over open relays is that spam sent from open proxies cannot be traced back to the spammer's machine [15]. However, in recent years, spammers have moved away from using these sources in favor of spam zombies [14].

Although we recognize that some of the above mentioned sources are still being used to send spam, industry anti-spam metrics show that the spam they send represent a relatively decreasing percentage of total spam [1]. What is missing is a source that is consistent with the above-mentioned trends. With such a source, spammers can safely send a significant amount of spam in an efficient manner. In the next sub-section, we will examine a commonly used source known as a spam zombie.

2.4 Spam Zombies

In recent years, a substantial amount of spam has been detected from one particular source known as spam zombies, where according to a leading anti-spam service provider, approximately 80% of all spam comes from spam zombies [16]. In this section, we provide some background information about spam zombies and apply our four-step model against spam zombies to help understand why they have become a preferred means to send spam.

2.4.1 What is a Spam Zombie?

A spam zombie is an end-user system that has been compromised by malware [14] where malware is defined as any software program developed for the purpose of causing harm to a computer system, similar to a virus or trojan horse [19]. Once malware is installed on the system, the spammer is able to use the compromised system as a spam distribution channel without the knowledge of the system owner. Many of these end-user systems are insecure and have broadband Internet access. Specifically, these machines are typically missing critical operating system or application level patches and are hence vulnerable to the many corresponding exploits being spread across the Internet. The spread of such exploits are performed by compromised systems called bots, which scan the Internet for hosts possessing a vulnerability for the given exploit. Once such a host is found the exploit is applied, which discreetly opens a backdoor port on the host so that the attacker can access the machine. All this activity is done in the background such that the owner of the compromised machine has no idea what is going on their machine. One of the things that an attacker does with a compromised host is install spamware, an SMTP engine

designed specifically to send spam, and then sells the use of such hosts to spammers [14]. According to CipherTrust, a vendor of messaging security solutions, research findings confirm that 300 000 new zombies are created daily [20].

Spammers are able to control compromised hosts, through a central server known as the command and control server. With respect to terminology, each compromised host is referred to as a bot and the collection of bots controlled by the command and control server is called a botnet. Figure 2-2 illustrates a high-level architectural overview of a simple 1-to-many botnet architecture.

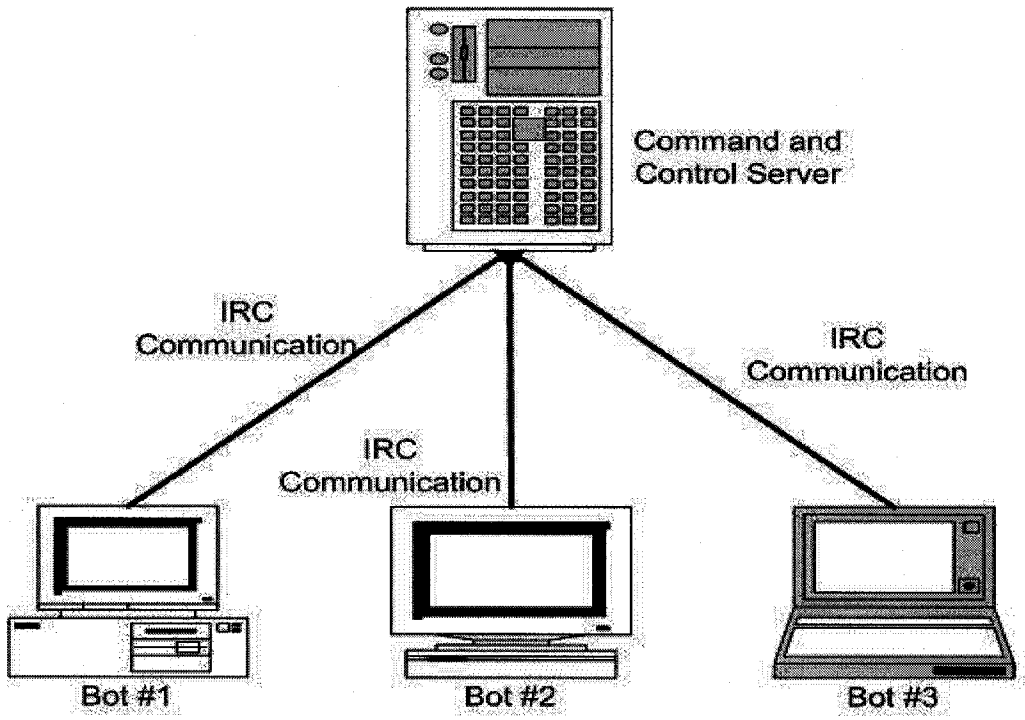

Figure 2-2: High Level Botnet Architecture

In the architecture of Figure 2-2 above, the command and control is able to communicate with each bots via a modified Internet Relay Chat (IRC) protocol.

Specifically, each bot executes a stripped down IRC client which allows them to authenticate on the proper IRC channel on the command and control server. The client also allows the bot to understand and perform commands given by the attacker through such a channel. Through the command and control server, the spammer can then control up to several thousand bots, allowing him or her to perform a mass spam campaign or scan the Internet to find more bots to add to the botnet [21]. For example, the IRCd server software allows an attacker to control up to 80 000 bots [21].

2.4.2 Spam Zombie as Spam Source

To help understand why spam zombies have become a preferred means of sending spam, we apply our four-step model to spam zombies.

1. *Spammers are moving away from having to use their own sources to send spam* [14].

   Spam zombies allow spammers to use other people's resources to send spam. Specifically, as mentioned above, a spam zombie is a compromised machine that usually provides spammers with enough processing power and Internet connectivity to send spam.

2. *Spammers are moving away from sources that do not protect their identity* [14].

   Spam zombies protect the spammer's identity since the spams sent from spam zombies can only be traced back to the compromised machine and not the spammer [13].

3. *Spammers are moving away from sources that require a non-trivial amount of manual labour* [14].

   Today, many users lack general computer security knowledge and either are not aware of the importance of patching and updating anti-virus signatures or simply do not do it. By exposing vulnerable computing systems to the Internet, these systems can be quickly transformed into spam zombies by various exploits and worms. Specifically, once an exploit encounters such a vulnerable machine, the machine then becomes compromised and the spammer may then have complete control over it. A benefit of this approach is that all the activity is automatically done in background such that the owner is unaware of what is happening to their machine and minimal effort is required on the part of spammers.

4. *Spammers are moving away from using third party sources that are difficult to find* [14].

   The combination of the booming popularity of broadband Internet, affordable computer hardware and a lack of general computer security knowledge, provides spammers with a significant number of potential systems that they can use to conduct their activities. According to Panda Software, approximately 35% of all Personal Computers (PCs) were virally infected as of February 2005, down from 50% in November 2004 [14]. One benefit of having many potential spam zombies is that it allows spammers to overcome today's anti-spam technology such as Domain Name Service Black Lists (DNSBL), which list IP addresses that are known to send spam. Specifically, a fresh supply of spam zombies can

consistently overcome such a filtering mechanism. DNS-blacklists are described in greater detail in Section 3.4.2.

As we can see from the analysis above, the use of spam zombies is consistent with the above-mentioned trends. We believe that this is a major factor that resulted in spam zombies becoming a preferred source for sending spam.

2.5 Summary

As spam has evolved so too have the sources that are used to send them. Unlike other sources used in the past, spam zombies provide spammers with an efficient means for sending spam while protecting their identities. Furthermore, with the growing popularity of the Internet combined with a lack of general computer security knowledge amongst end users, spammers are provided with a wealth of potential computing sources that can be compromised for malicious purposes. Once a machine is compromised and turned into a zombie, non-standard mail server software called spamware [14] is installed on the system and used to deliver spam. In the next Chapter, we examine the current state of the art with respect to spam prevention and filtering.

Chapter 3: Current State of the Art - Combating Spam

3.1 Introduction

The previous Chapter illustrated the evolution of spam sources where the majority of today's spam originates from spam zombies. This Chapter presents a literature review on current efforts at combating spam and the state of the art of spam filtering. This Chapter is divided into four general sections: legislation, preemption, filtering and remediation. For each approach, we will outline their goals, uses and shortcomings. Once the current state of the art is presented, we will outline the goals of a new message filtering solution addressing the shortcomings of current filtering efforts and illustrate these ideas in subsequent Chapters.

3.2 Legislation

The goal of legislation is to create an enforceable law, which would make it illegal to send spam and impose heavy penalties for those caught doing so. We consider legislation as an anti-spam solution because imposing legal consequences to spamming should discourage people from spamming and there are those who believe that no technological solution will solve the spam problem [22].

In the United States, anti-spam legislation called the CAN-SPAM act [23] was introduced to help combat the spam problem. According to the CAN-SPAM act, in order to send unsolicited commercial e-mails, the e-mail must include, among other things, a means for the end user to opt-out of the distribution, a postal address where the user can contact the sender, true memo headers and a subject line header that does not mislead the end user as to what the content of the e-mail contains. In addition, all sexually explicit e-mail content needs to be sent with a label indicating such content in the subject line.

Unfortunately, the CAN-SPAM act has had little effect on spam volumes since its inception in 2003 [24] and may have promoted spam by legalizing certain types of spam [25]. At the time when the CAN-SPAM act went into effect, industry estimates put the total spam volume at around 35% of all e-mails [26]. This is in comparison to 2006/2007 where spam volumes have reached 67% of all e-mails [1]. One major problem with CAN-SPAM and other anti-spam legislation is the difficulty in tracing spam to the actual spammer [27] and hence enforcing the law. Specifically, with over 80% of all spam sent from compromised machines, tracing spams to the originating source will often lead to the innocent hi-jacked owner and not the spammer themselves. Furthermore, the distributed nature of the Internet makes it easy for people to send spam from anywhere around the world. Therefore, anti-spam laws have the effect of forcing spammers send spam from jurisdictions where laws are not in force.

3.3 Pre-emption

The goal of pre-emption is to prevent non-dedicated mail servers from becoming spam sources.

3.3.1 User Education

One solution that has been proposed is a mass educational campaign to educate end users on how to secure their computers and prevent them from becoming spam zombies [14]. The goal of such campaign is to make end users aware of the risks involved in not applying security patches, updating anti-virus signatures, opening suspicious e-mail attachments, downloading spyware infested peer-to-peer programs, and clicking on malicious links obtained via e-mail or instant messaging.

Unfortunately, user education can only go so far and no matter how much effort and money is invested in such campaigns, the method is not fool proof [28]. So while there is value in user education, the battle against spam zombies is unlikely to be fully resolved by this method alone.

3.3.2 Port 25 Filtering

In order for a machine to send spam, it must be able to connect to a relay or a destination mail server. More specifically, it must initiate an outbound port 25 request to one of these servers [29]. With the consequences of having compromised machines on one's network, many Internet service providers and corporations have blocked outbound port 25 access on all hosts on their network except those that are explicitly allowed to perform SMTP relay functions. That way, even if a machine is infected and behaves as a spam zombie, it will not be able to send spam [30]. In addition to blocking outbound port 25, the service provider or company may also want to block inbound port 25 access.

Although this is a simple yet effective means of stopping spam, it has its shortcomings. Specifically, not all service providers or corporations have adopted this policy since their users do not wish to have such blocks implemented. For example, some users may wish to use their own mail servers to send and receive mail rather than having to use a dedicated mail server provided by their provider or company. Such a situation could arise when a new business customer of a service provider requires their e-mails be stored on their own servers only and not on their service provider's dedicated mail server. It should also be noted that the port 25 block does not prevent computers from becoming zombies. Rather it prevents zombie computers from being able to send spam. As such, an attacker will still be able to control the zombie computer and use it for other purposes such as storing illegal software or scanning the network for other machines to infect.

3.3.3 Anomalies in DNS traffic

Another pre-emptive anti-spam solution is to detect abnormal DNS MX query requests [31]. Such requests are typically performed by an SMTP mailing agent that wishes to send mail to a specific domain. In order to find the mail server accepting mail on behalf of that domain, the agent must do a DNS lookup to find that domain's mail server's IP address. These types of lookups are called MX queries and should not be initiated on a machine that does not perform mail transport functions.

In an ISP network, most SMTP traffic is usually routed through the provider's mail server. In such cases, the end user's machine should never perform an MX DNS lookup; rather it is the provider's mail server that is responsible for this. As such, if an end user's

machine is detected performing DNS MX query lookups, there is reason to be suspicious. Such a machine can be flagged as possibly being infected and isolated from the network.

One of the main drawbacks to this approach is false positives with respect to the alarms being generated by such an approach. Specifically, some e-mail clients are configured without an outbound SMTP server such that when a user tries to send mail to a user from an external domain, the sending user's machine will perform an MX query lookup in order to find the mail server it should be communicating with. Therefore, alarms generated by the anomalous DNS approach described above will not only detect spam zombies but will also falsely detect user machines where the e-mail client is configured to send mail directly to the recipient's server. False positives are a problem because each alarm requires a security officer to inspect and analyze it and the more false alarms that get generated, the greater the amount of time and effort that is wasted by the officer.

3.3.4 Command and Control Server Detection

As described in section 2.4.1, when a machine is infected with malware [19], it is common to have the host under the attacker's control via a command and control (C&C) botnet server. By using such a server, the attacker can control a plurality of zombies (also known as bots) by performing tasks such as uploading spamware software [14] to send spam. The bots connect to the C&C server by either IP address or domain name. It should be noted however that if the C&C server's IP address is dynamically assigned, it will most likely be contacted by its domain name.

One way to help reduce the number of spam sources is to detect upstream botnet IRC communication between the C&C and the compromised machine where said IRC communication is illustrated in Figure 2-2 [14]. By finding the IP address or domain name of the hosting server, one can then filter the IP address or domain name so that any other machines on the network that become compromised cannot be controlled by that C&C. However, one problem with this approach is that the C&C server may reside on a dynamic IP address where blocking the IP address could potentially cause problems later on if the filter for that address is not eventually removed and is later reassigned to an innocent owner or company. Although terminating communication with the botnet server is helpful in preventing new spam zombies from appearing in one's network, it does not necessarily prevent existing compromised machine from sending spam.

3.4 Filtering

The goal of anti-spam filtering is to filter messages by identifying which messages are likely to be spam and which are not. As we will see in Section 4.3, anti-spam filtering typically occurs towards the end of the SMTP transaction. In the following sections, we explain how spammers have found ways to modify the content of their e-mails such that some of these e-mails may overcome existing content filtering approaches. We also indicate how the content filtering techniques described below cannot detect the presence of new spam zombies nor prevent spam from said sources.

3.4.1 Content Filtering

Content filtering was one of the first types of anti-spam filters to be used. Such filters use hard coded rules where each rule has an associated score and is periodically updated [32] [33]. An example of such filter is SpamAssassin [34], which works by scanning the textual content of the e-mail against each rule and adds the scores for all matching rules. If the total score of the e-mail exceeds some set threshold score then the message is considered spam. In order to generate these scores, a single perceptron is used [35] where the inputs to the perceptron indicate whether a particular rule was matched and the weight for the corresponding input indicates the score for each rule. More detailed information with respect to machine learning algorithms such as the single perceptron is presented in Chapter 6.

The simplest of content-based rules flag e-mails if a specific string or expression was matched. For example, one type of content filtering rule would be to match all e-mails that contained a variation of the word VIAGRA in the body or subject line. Content-based filters can also match keywords or expressions in other parts of the e-mail such as the headers or the base64 encoding of e-mail attachments and embedded images. E-mail headers are typically used to determine where an e-mail came from and how it was delivered from the initial source to the current destination.

Although content filters are still being used, they are becoming less effective at blocking spam. First, spammers have found ways of getting around such filtering approaches by modifying their text. For example, if spammers are aware that a certain filter is filtering e-mails based on a variation of the term CIALIS excluding the variation

SPECIALIST then they may send the e-mail with a modified version of CIALIS such as SPE**CIALIS**T is unlikely to hit any exact match or regular expression based CIALIS content filtering rule due to the SPECIALIST exemption. Even the combining of regular expressions from a variety of sources is unlikely to overcome this modification [36]. The modified term is still read as CIALIS by the end user but looks like SPECIALIST to the content filtering system. Since content filtering is based on a static rule set, spammers can get a significant volume of spam to bypass said filter by crafting a message that will not trigger any content filtering rules.

Another approach to content filtering is called collaborative spam detection [37]. Specifically, when a receiving server receives an e-mail, it generates a signature for that particular e-mail and compares it against the signatures in a global server list to see how frequent the message has been reported as spam. If a match is found and was reported as spam above a specific threshold, the message is considered spam. Examples of this approach include Vipul's Razor [38] or Distributed Checksum Clearinghouse (DCC) [39]. One problem with this approach is that it detects bulk e-mail and not necessarily spam. For example, some bulk e-mails such as legitimate corporate communications from outsourced vendors may be seen as spam by such techniques and as such these e-mails may end up getting blocked. Another problem is that a simple change in the content by the spammer such as re-adjusting an image size or changing a few words can generate a completely different signature that would not already be listed in the global server list and as a result, such spams would get through this filtering technique.

For more information regarding the drawbacks of using content filtering, we refer the reader to [4].

3.4.2 Network Based Filtering

An alternate approach to content based filtering is network level filtering, which typically comprises of a blacklist. A blacklist comprises of known bad IP addresses that have been used to send spam. A mail server uses a blacklist when it receives a connection from a sending server. Specifically, the receiving server checks the sending server's IP address against the blacklist to determine if the sending server's IP address is listed.. If it is listed then the mail is rejected. However, if the sending server's IP address is not listed in the blacklist then it is inconclusive as to what the filtering system should do with messages from the sending server.

There is a wide range of different publicly available blacklists where each list has a specific focus. For example, the Composite Blocking List (CBL) [40] and Not Just Another Bogus List (NJABL-PROXY) [41] are focused on compromised machines whereas the Spamhaus [44] blacklist [6] list is focused on known spam sources that are not necessarily compromised machines.

When a receiving mail server receives an incoming SMTP connection, the receiving server can query a blacklist to determine whether the incoming mail server is listed on that blacklist or not. The query is performed by doing a reverse DNS lookup on the blacklist server. The response from the blacklist server helps the receiving server know whether to block or allow such e-mails. For example, if a receiving mail server received a connection from a sending server with an IP address of `1.2.3.4` and wanted to know

whether the sending server was listed on the Spamhaus blacklist then the receiving server would perform the following DNS query:

```
4.3.2.1.xbl.spamhaus.org
```

If the IP address is listed in the Spamhaus blacklist then the response would be

```
127.0.0.2
```

If the IP address is not listed in the Spamhaus blacklist then the response would be

```
NXDOMAIN.
```

By examining the response received from the blacklist server, the receiving server will be able to determine whether the IP is listed and whether or not they should accept the e-mail.

One problem with blacklists is that they are detection based and suffer from lag times. For example, in [42] of the 4295 spam zombie hosts IP addresses queried against the Spamhaus blacklist, only 255 were blacklisted after 46 days of continual activity. This implies that there is a lag time from when a new spam source starts to send spam, to the time when the IP address of the source is listed in the blacklist [45]. The actual criteria used to determine whether an IP gets added to the list and how long that IP remains in that list is unclear and as a result, false positives can also result from using such lists. In fact, some blacklists are poorly maintained such that legitimate users are unable to effectively use IPs once used by spammers [46]. Furthermore, since the list is detection based and zombies are known to generate a large amount of spam immediately upon infection, it is possible that hundreds of spams can get through a filter that utilize such blacklisting approaches [42]. Another problem with blacklists is that the major blacklists [39][40][41] are IP-based. In particular, since many spam zombies are known to reside

on end user machines with broadband connection, typically on dynamic IP space [14], using IP addresses to list spam zombies may not be effective solution especially if said zombies are continuously assigned new IP addresses. With thousands of new malicious sources being generated everyday, such reactive based approaches are always a step behind the attacker.

Since spammers tend to make efficient usage of their spam-related websites, it is not surprising that many spams contain similar Universal Resource Identifiers (URIs), which is a standard means of addressing resources on the web. This leads us to another type of blacklist called the URI blacklist, which lists URIs, found in spams. An example of such list is the Spam URI Realtime Blocklists (SURBL) [48].

Unfortunately, URI blacklists suffer from similar problems as IP based blacklists where the list is generated on a detection-basis and a spam containing a newly created URI will not be detected by such list. Furthermore, there is also the risk of false positives. For example, in cases such as phishing e-mails where a 'clean' URI is obfuscated and actually directs the user to the spammer's site, the 'clean' URI may end up getting listed in that list.

Since the above-mentioned blacklists are owned by third party volunteer organizations, it makes sense to have a local whitelist that can be used in the event of a false positive. In particular, such whitelists are typically designed to allow mail from a particular IP or e-mail address to bypass all spam filtering. One issue with this is that the machine using the whitelisted IP address may become compromised giving the spammer unfiltered access to recipient's inbox. Similarly, if the spammer knew that a particular e-

mail address was whitelisted, he or she could forge that e-mail address in their spam to ensure that the spam gets through the filter. As a result, whitelists need to be carefully maintained so that they do not get abused, which leads to additional overhead for the e-mail administrator.

For more information regarding the drawbacks of using network filtering, we refer the reader to [4].

3.4.3 Forging

With a significant amount of spam being forged, it makes sense to look at techniques that detect such forgeries. Email forging is where the 'From' line of an e-mail is forged such that the email appears to have been sent from someone else. For example, Alice could forge an e-mail to Bob to make it appears that the e-mail came from Jim by re-writing the 'From' line of the e-mail with Jim's e-mail address.

In the absence of technological detection means, one may discover that they have been spoofed when someone complains to them about an email they never actually sent or they receive an e-mail bounce from email that was generated elsewhere.

One technique to detect e-mail forging is called the Sender Policy Framework (SPF) [47]. SPF determines whether the sending mail server is allowed to send mail on behalf of the sender's domain. The success of SPF is dependent upon domains registering a DNS TXT record, which lists all the mail servers that are allowed to send mail on behalf of that domain. Although SPF is able to detect forged messages from domains that have SPF records such as Microsoft, SPF is not an effective means for filtering spam.

Specifically, spammers are known to register new domains with SPF records thus allowing their e-mails to bypass any SPF filter.

Domain keys [49] is another anti-forging technique where instead of listing allowed mail servers, a domain specific public key is listed in the TXT record. Domain keys works by having the sending mail server sign the message using its public key and include a hash of that signature in the header of the e-mail. The receiving server then finds the public key of the sender's domain, listed in the domain's DNS TXT record, to sign the message and produces a hash of the signature. If the hash value in the header and the hash value that the receiving server generated matches, then the receiving server can be confident that the message was not forged.

Another problem with current sender forgery techniques is that it does not handle relays or mailing lists very well. In particular, when a user sends an e-mail to a mailing list, the mailing list server receives the message and subsequently relays the message to the mail servers of all the recipients on the list. In such a case, the signature generated by the sender and the recipient servers would be different because of the additional headers added to the original message by the intermediate mailing list server.

3.4.4 Machine Learning

In this sub-section, we review two anti-spam based machine learning filtering techniques.

3.4.4.1 Client Based E-mail behavioral analysis

This technique uses an e-mail data mining toolkit to analyze offline email corpus including e-mails sent and received from a user [50]. Specifically, machine learning

algorithms are used to model the user's behavior to classify emails for spam detection. We note that this work does not conduct behavioral analysis of the sending mail server to classify mail and instead models the user's sending and receiving behavioral patterns to detect spam.

3.4.4.2 Bayesian Filtering

The Bayesian algorithm makes use of Bayes's rule, which uses prior probability knowledge to calculate the most likely classification of data. Given a set of features (f1,f2,f3..fn) to extract from the data, the Bayes theorem can determine the probability of the data being spam. Specifically, according to the Equation 3-1 [51][52], the probability that an e-mail is spam given that the e-mail contains a number of features, is equal to the probability that any e-mail is spam multiplied by the probability that an e-mail containing those features is found in spam divided by the probability of those features found in any e-mail.

$$P(SPAM|f_1,f_2,f_3,..f_n)=\frac{P(SPAM)P(f_1,f_2,f_3,....,f_n|SPAM)}{P(f_1,f_2,f_3,..f_n)}$$

Equation 3-1: Bayesian Formula

In the e-mail case, these features are words where each word has a certain probability of it occurring in a spam or legitimate e-mail. For example, the word VIAGRA is likely to be found in spam but not in legitimate e-mail. The filter learns by training it with good and bad e-mails. By knowing which words are in good e-mails and bad, it then developed probabilities for each word and stores it in a database. For example, the word VIAGRA would have a large probability in spam and low probability in non-spam. After

training, the word probabilities are used to determine whether an incoming email is spam or not. Each word in the e-mail contributes to the overall email's spam probability. The e-mail spam probability is computed over all words and if it exceeds a threshold then the filter will mark the e-mail as spam.

According to the Bayesian filtering approach, when new mail arrives, a message is scanned for all the features defined in the formula above such as words with a high probability of being in either spam or legitimate mail. The probability of each feature being in spam is calculated and used in the Bayesian formula to determine the overall probability of the e-mail being spam.

To better explain how Bayesian can be applied to the e-mail domain, we describe the existing use of Bayesian in content filtering. In such filtering, a corpus of mail is classified into two distinct groups and used for training purposes. The first group consists of legitimate mail while the second group consists of spam. For each group, the algorithm generates a number of features from the two groups and assigns a probability to each feature based on the number of times it appears in each group. In the e-mail domain, a feature could be considered a word. For example, a feature word such as 'conference' would appear in legitimate mail and not in spam and as a result, would have a conditional probability `P(f1=business|SPAM)= 0`. On the other hand, a feature word such as 'Viagra', would appear in spam and not in legitimate mail and hence would have a conditional probability `P(f2=Viagra|SPAM) = 1.`

One major flaw with the Bayesian filtering approach is that a large amount of training data are required to accurately determine the conditional probability of a feature being in

spam or legitimate mail. Specifically, in order to effectively use the Bayesian algorithm, we would need an extremely large corpus where each feature was used multiple times. Obtaining this type of corpus of training data is extremely difficult .

For more information regarding the drawbacks of using machine learning based filtering, we refer the reader to [4].

3.5 Remediation

The final anti-spam solution discussed in this Chapter is remediation where the malicious viral code known as malware [19] is removed from compromised machines. This approach however, requires the user to know that their machine is infected in the first place and know how to clean it up. In section 3.3 of this Chapter, we mentioned a few techniques to detect spam zombies. Specifically, one method is to look for anomalous DNS traffic and the other is to look for computers communicating with a known C&C server.

Since compromised machines are under the control of a botnet server, no one really knows what malicious software gets uploaded and installed on these hosts. As such, in order to remove such software from the machine, we recommend that the user re-install the operating system. Another challenge with cleaning infections is that viruses these days have become more difficult to kill. Some viruses disable key functionality such as anti-virus applications, pc firewalls, task manager, registry editor as well as access to important sites such as windows update and antivirus websites [14].

3.6 Summary

In this Chapter, we analyzed current efforts at combating spam and state of the art anti-spam solutions, which were categorized into one of four approaches: legislation, preemption, filtering, and remediation categories. By examining current filtering efforts and their shortcomings, we can begin illustrating our idea of a new message filtering solution that can address these issues.

Chapter 4: SMTP Transaction Model and Mail Rejection

4.1 Introduction

The previous Chapter presented related research on current efforts at combating spam and the state of the art of spam filtering. We note that existing anti-spam filtering techniques focus on distinguishing good and bad e-mail content while our filtering technique differs from existing solutions in that we look at discrepancies in mail transactional behavior between a legitimate mail server and a spam zombie. In this Chapter, we examine the SMTP transaction between a legitimate sending and receiving mail server and model the transaction in five sequential steps. We also examine the different ways that an SMTP transaction can be rejected and look at the different sender notification mechanisms used to notify senders that their mail has been rejected. Once the transactional model is presented, we will outline the machine learning algorithm goals of our new message filtering solution.

4.2 SMTP Procedure

The ability to exchange messages between a sending and receiving mail server is made possible by the SMTP protocol. In this section, we provide an overview of the SMTP protocol as defined in RFC 821 [53] and RFC 822 [54] developed in 1982 and extended

in RFC 2821 in 2001 [55]. An example of a successful SMTP transaction between a sending and receiving mail source is illustrated below.

```
220 mta554.mail.mud.receiver.com ESMTP YSmtp service ready
HELO mail.receiver.com
250 Hello mail.receiver.com [198.2.25.87], pleased to meet you
MAIL FROM: <alice@sender.com>
250 <alice@sender.com>... Sender ok
RCPT TO: <bob@receiver.com>
250 <bob@receiver.com>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
From: alice@sender.com
To: bob@receiver.com
Date: January 18, 2007
Subject: Test

This is a test
.
250 2.0.0 l0J5FYf01210 Message accepted for delivery (relayed by
MailShield
```

In the example above, the sending mail server `mail.receiver.com` engages in an SMTP transaction with the receiving mail server `a.mx.mail.receiver.com` for the purpose of delivering mail from the user account `alice@sender.com` to the user account `bob@receiver.com`. All communication from the sending mail server is in bold while

the responses from the receiving mail source are underlined. Each step within the SMTP transaction will be described in the following sub-sections.

4.2.1 Step 1: Session Initiation

An SMTP session begins when a sending mail server connects to the receiving mail server on the receiving server's SMTP port; typically, TCP port 25. If the receiving mail server accepts the connection from the sending mail server, the receiving mail server presents the sending server with a banner message starting with a `220` SMTP code followed by text identifying the receiving server's fully qualified domain name, SMTP software and version. In our example, the banner message that the sending server received was:

```
220 mail.receiver.com ESMTP YSmtp service ready
```

In the banner above, the `220` represents an SMTP code that indicates that the SMTP service is ready, `mail.receiver.com` is the fully qualified domain name of the receiving server and ESMTP is the mail software used by the receiving server. In this case, the receiving server likely concealed the version information of the mail software for security reasons.

It should be noted that instead of presenting a banner with SMTP code `220`, the receiving server can respond to an incoming connection with SMTP code `554` which indicates a transactional failure. This type of situation may occur if the receiving server is configured to consult a blacklist prior to accepting connections. Specifically, if the receiving server is configured to deny connections from all IP addresses listed in the

blacklist and the sending server's IP address is listed, then the receiving server can immediately reject the connection. According to RFC 2821, if the incoming connection is rejected, the server must wait for the sending server to send a QUIT command before closing the connection and should respond to all other commands from the sending server with SMTP code `503`, which indicates a bad sequence of commands. However, to prevent Denial of Service (DoS) attacks that connect to mail servers without issuing a QUIT, the receiving server may be configured to drop a connection after a pre-specified amount of inactivity has elapsed. Such a pre-specified time period varies for each mail server environment.

4.2.2 Step 2: Sending Server Identification

If the sending server is able to establish an SMTP connection with the receiving server and receives an SMTP code `220` banner message, the sending server must then identify itself to the receiving server. This is performed with either the HELO or the EHLO command. The HELO command is used to simply identify the sending mail server. In our original example, the sending server identified itself by issuing the following command:

```
HELO mail.receiver.com
```

This command can be interpreted as saying 'Hello, I am `mail.receiver.com`'. In response to this command, the receiving server responded with SMTP code `250` which indicates that the requested mail action was okay and completed:

```
250 Hello mail.receiver.com [198.2.25.87], pleased to meet you
```

The EHLO command is used when the sending server wishes to use a service extension of the SMTP protocol. An example of a sending server using EHLO communication is provided below:

```
EHLO mail.sender.com
The response to the above EHLO command is as follows:
250-mail.receiver.com Hello mail.sender.com [198.2.25.87], pleased to
meet you
250-ENHANCEDSTATUSCODES
250-EXPN
250-VERB
250-8BITMIME
250-SIZE 20000000
250-DSN
250-ONEX
250-ETRN
250-XUSR
250-STARTTLS
250 HELP
```

In the example above, the parameter associated with the EHLO command identifies the sending server (`mail.sender.com`) and results in the receiving server (`mail.receiver.com`) responding with a list of extensions it supports. At this point, the sending server can use any of the available service extensions. For example, if the sending server wishes to encrypt the SMTP transaction, it would use the `STARTTLS` service.

In some cases the use of the EHLO command will result in the receiving server returning a SMTP code 500 response which indicates that the command is not recognized and that the receiving server does not provide any service extensions. In such a situation, the sending server must use the HELO command instead.

According to RFC 2821, the HELO/EHLO parameter requires a fully qualified domain name of the sending server. If no meaningful domain name is found, the client should send an address literal which is an IP address enclosed in square brackets. It should be noted that majority of legitimate mail servers are assigned static IP addresses which typically map to a legitimate domain name. As such, very few servers should have address literals as its HELO/EHLO parameter and as we will see in Chapter 5, spam zombies will have unusual HELO/EHLO parameters. Therefore, the HELO/EHLO parameter values can help identity whether a server is legitimate or not. We will look at this in the subsequent Chapter.

4.2.3 Step 3: Sender Identification

After the sending server identifies itself to the receiving server, the sending server issues the MAIL command to identify the sender. The command is generally used in the following form:

```
MAIL FROM:<address>.
Note: address represents the sender's e-mail address (e.g.
alice@sender.com)
```

When the sending server issues the above command, the receiving server prepares for the start of a new mail transaction by resetting all state tables and buffers of previous MAIL transactions.

Now, referring to our original example, the sender server informs the receiving server that the sender is `alice@sender.com`:

```
MAIL FROM: <alice@sender.com>
250 <alice@sender.com>... Sender ok
```

In this case, the receiving server issued a response with SMTP code `250`, which means that the command was accepted and the associated e-mail address was stored in its buffer; however, if the command were not accepted then the receiving server must provide the sending server with a response to indicate whether the failure is permanent or temporary. If the failure is permanent, the sending mail server will not be able to send mail to the address under any circumstances. Such a situation would occur, for example, if the sender's e-mail address was listed on a blacklist, which lists all e-mail addresses that the receiving server does not accept. In contrast, a temporary failure means that the sender address might be accepted if the sending mail server were to try specifying the same sender address again.

4.2.4 Step 4: Recipient Identification

The RCPT command identifies the recipient of the sender's message and is followed by the MAIL command; however, if the receiving server receives a RCPT command without a previous MAIL command then the receiving server should send a response with SMTP code `504` (bad sequence of commands) to the sender. Unlike the HELO or MAIL

commands, the RCPT command can be used multiple times for each SMTP transaction. The reasoning behind this is that for each e-mail there is only one sending server, one sender address and potentially multiple recipients.

The general form of the RCPT command is similar to the MAIL command and is shown below:

```
RCPT TO:<address>.
Note: address is the recipient's e-mail address.
```

When the sending server issues the RCPT command, the receiving server verifies the address exists and belongs to its domain. For example, in our original example, the receiving server would only accept RCPT commands containing a valid address for the receiver's domain (i.e. `user@receiver.com`). Since our example met this criterion, the receiving server issued a response with SMTP code `250` and stored the address in its buffer.

```
RCPT TO: <bob@receiver.com>
250 <bob@receiver.com>... Recipient ok
```

However, if the address was not a valid address, then the receiver server would respond to the sending server with a SMTP code `550` reject response, which indicates that the requested action was not taken and the mailbox is unavailable.

4.2.5 Step 5: E-mail Content

The DATA command is used to define the content of the message to be sent to the recipient and is followed by the RCPT command. It is important that the DATA command be used after the RCPT, MAIL and HELO commands or else an out of

sequence failure will result. Specifically, if there was no MAIL or RCPT command issued before the DATA command then the receiving server will either issue a response to the sending mail server comprising SMTP code 503 or 554, which indicates that the requested action was not taken and that the mailbox name was not allowed. If the sending server receives either of these responses, the sending server must not send any message data. In fact, message data must not be sent unless an SMTP code 354 response is received. The typical form of the DATA command is as follows:

```
DATA <CRLF>
Note: CRLF represents a carriage return and line feed.
```

If the response to a DATA command is SMTP code 354, then all content provided by the sender server after DATA and up to but not including the end of mail data indicator is considered message text. The end of mail data indicator is a single line containing a period '.' and informs the receiving server to process stored recipients and mail data. For an illustration of such a transaction, we refer to our original example:

```
DATA
354 Enter mail, end with "." on a line by itself
From: alice@sender.com
To: bob@receiver.com
Subject: Test

This is a test
.
250 2.0.0 l0J5FYf01210 Message accepted for delivery (relayed by
MailShield)
```

In the above example, once the message data are received and stored by the receiving server, the receiving server issues an accept response comprising SMTP code 250. In addition to the out of sequence failure, a failure could also result if the message content triggers content related anti-spam rules. In such a situation, a permanent failure message with SMTP reject error code 550 should be sent to the sending server after the end of mail data indicator. It should be noted that the memo header items such as Date, Subject, To, Cc and From are all part of the message data. We expect that the e-mail content from spam zombies, otherwise known as spam, will be different from the e-mail content sent from legitimate mail servers. As such, we will examine the e-mail content from spam zombies in the subsequent Chapter.

4.3 Overall SMTP flow

According to RFC 2821 [55], a typical SMTP transactional flow sequentially follows each of the five steps as outlined in the five step model above and illustrated in Figure 4-1 below.

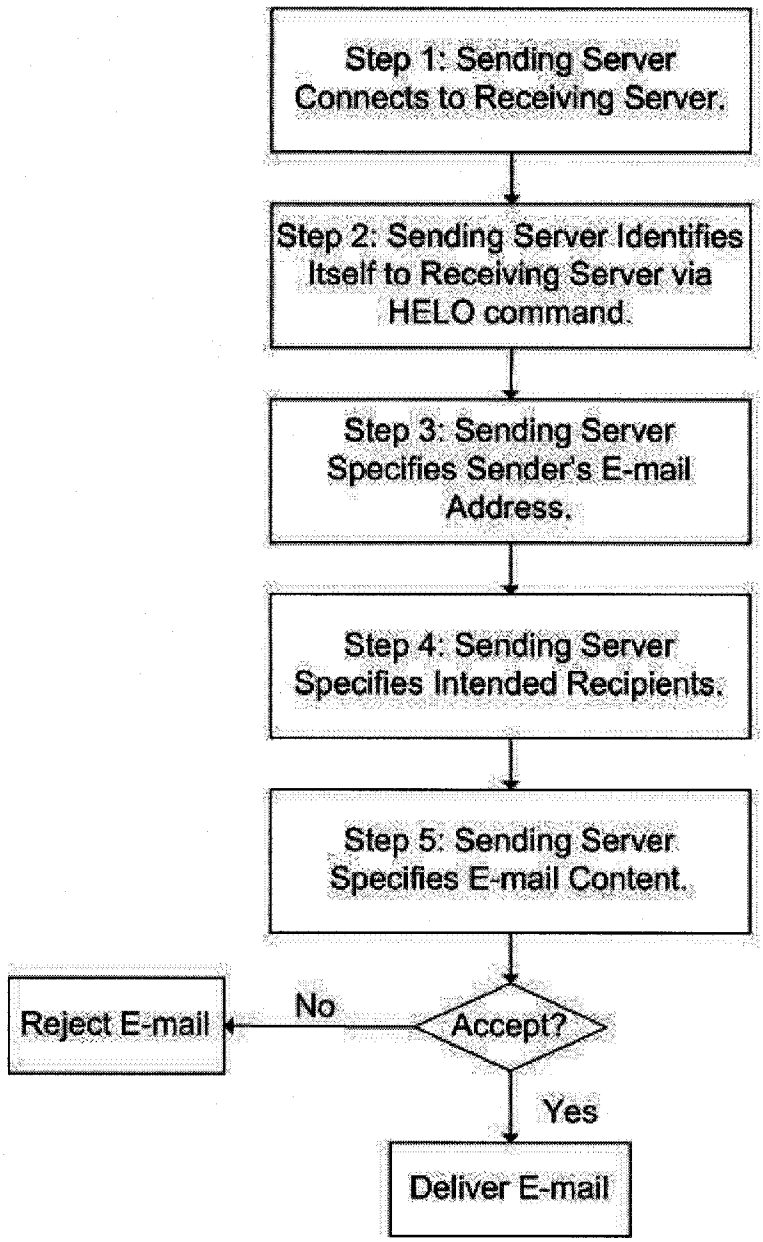

Figure 4-1: SMTP Transactional Flow

In Step 1 of Figure 4-1, a sending server initially establishes a connection with the receiving server,. In Step 2, the sending server introduces itself to the receiving server using the HELO or EHLO command with a parameter that identifies the fully qualified domain name of the sending host. Once the command is received and processed by the receiving server, the sending server then proceeds to Step 3 of the SMTP transaction and issues a MAIL FROM command to identify the sender. In Step 4, the sending server issues one or more RCPT TO commands to identify the recipients. While sending RCPT TO commands, the receiving server may verify that each recipient address is a valid address for their domain. In Step 5, the sending server may issue a DATA command and define the content of the message to be delivered. It is at this point where most spam filtering, such as content and network based filtering checks, usually occur. We note that the filtering servers we use in this thesis perform filtering at this point of the SMTP transactional model.

In summary, a typical SMTP transaction according to RFC 2821 follows the following flow: HELO->MAIL FROM->RCPT TO->DATA. We hypothesize that legitimate mail servers will abide by RFC 2821 with no guarantee that the same will hold true with respect to spam zombies. Therefore, we will examine the SMTP transactional flow of spam zombies in the subsequent Chapter.

4.4 Rejecting Mail

In the previous sections we illustrated how the receiving server can reject mail from the sending servers. In this section, we describe what happens once a receiving server rejects an e-mail. Specifically, when a receiving server responds to a sending server's command

with a reject response code, the sending server will generate a non-delivery receipt and send it to the sender address. The main problem with this approach is that such a non-delivery receipt provides very little information on what the user or their service provider's technical support staff can do to report or fix the problem. Since the SMTP reject issue was due to the receiving mail server's configuration, the e-mail administrators of the receiving server must be notified in order for this problem to be resolved. In such circumstances, the intended recipient must know that one of their incoming e-mails was not received and inform their e-mail administers that the e-mail did not get through. Although this is a feasible solution, a better approach would be to inform the sender that their e-mail did not get through and provide them with instructions on how they can report the false positives to the e-mail administers of the receiving domain. This is a more effective means of handling false positives because senders have a vested interest in ensuring their e-mails get delivered to their intended recipient. Since spam zombies are not legitimate mail servers, they typically do not go through the trouble of generating a non-delivery receipt to the specified sender address.

To implement such a mechanism, there are three things that are needed. First, the receiving server must be able to uniquely identify every e-mail it rejects so that the e-mail administrators can refer to their logs and determine why a particular e-mail got blocked. Second, the receiving domain must have a special e-mail address that external senders can report false positives to such that mail sent to that address can automatically bypass all filtering mechanisms used on the receiving server. Finally, the sender needs to be notified that their e-mail was blocked with instructions on how to report false positives.

In particular, the notification should contain the unique identifier along with the special e-mail address. The next section will provide details on how such a notification is sent to the end user.

4.4.1 In-line rejection

When a receiving server rejects a message from a sending server, the receiving server will issue an SMTP reject response during the SMTP transaction. To implement the sender notification solution described in the previous section, the SMTP reject response should include information such as the unique identifier and the unfiltered e-mail address that the sender can use to report the false positive. According to RFC2821, when a receiving server issues an SMTP rejection, the sending server will send a non-delivery receipt, including the SMTP reject response, to the user identified in the MAIL FROM. For example, in the SMTP transaction below, the receiving server rejected the message data based upon a simple content filtering rule (i.e. block mail without a Date). Note that the rejection response, which is highlighted in bold, includes the SMTP reject code `550` along with instructions for the sender to follow.

```
220 mail.receiver.com SMTP (UCE not wanted)
HELO mail.sender.com
250 Hello mail.sender.com [22.129.25.87], pleased to meet you
MAIL FROM: <alice@sender.com>
250 <alice@sender.com>... Sender ok
RCPT TO: <alice@sender.com>
250 <alice@sender.com>... Recipient ok
DATA
```

354 Enter mail, end with "." on a line by itself

From: alice@sender.com

To: alice@sender.com

Subject: Test

This is a test

.

550 Mail refused - by our email filters - If you believe this to be in error, please forward this whole bounce (including Session ID: 1169183674.02015825) to filtops@receiver.com ONLY

In response to the reject message, the sending server would generate and send the sender a non-delivery receipt, which would include the instruction provided by the receiving server. A sample non-delivery receipt with said instructions is presented below:

-----Original Message-----

From: Mail Administrator [mailto:Postmaster@receiver.com]

Sent: January 18, 2007 7:46 PM

To: alice@sender.com

Subject: Mail System Error - Returned Mail

This Message was undeliverable due to the following reason:

Your message was not delivered because the destination computer refused to accept it (the error message is reproduced below). This type of error is usually due to a mis-configured account or mail delivery system on the destination computer; however, it could be caused by your

message since some mail systems refuse messages with invalid header information, or if they are too large.

Your message was rejected by mail.receiver.com for the following reason:

Mail refused - by our email filters - If you believe this to be in error, please forward this whole bounce (including Session ID: 1169183674.02015825) to filtops@receiver.com ONLY

The following recipients did not receive this message:

<bob@receiver.com>

4.4.2 Bounces

E-mail bounces provide an alternate means of notifying senders that the receiving server rejected their e-mail. In this case, after the receiving server receives the entire message and rejects it, the receiving server creates a new e-mail called a bounce, encloses the content that the sender tried to send, and sends the bounce to the address listed in the MAIL FROM to inform the sender that the e-mail did not get through. An example of a bounce message is provided below:

Subject: Undelivered Mail Returned to Sender

Date: Thurs, 18 Jan 2007 16:31:44 -0500

From: "Mail Delivery System" <MAILER-DAEMON@a.mxmail.receiver.com>

To: <alice@sender.com>

```
This is the Postfix program at host box20.yyz1.setupahost.net.
I'm sorry to have to inform you that your message could not
be delivered to one or more recipients. It's attached below.
For further assistance, please send mail to <postmaster>
If you do so, please include this problem report. You can
delete your own text from the attached returned message.
                   The Postfix program
    user: "bob@receiver.com "
------_=_NextPart_001_01C73B8C.2AA0F400
Content-Type: application/octet-stream;
      name="Delivery report.txt"
Content-Transfer-Encoding: base64
Content-Description: Delivery report
Content-Disposition: attachment;
      filename="Delivery report.txt"
```

It should be noted that the main difference between this approach and the in-line rejection approach is the server that is used to notify the sender. Specifically, the in-line rejection method uses the sender's server while the bounce method uses the recipient's server. The problem with the bounce approach is that e-mail forging is a very simple technique that is widely used by spammers and spamware [14]. So having the receiving server send bounce e-mails to the supposed sender address, may result in an innocent user receiving a heavy volume of forged e-mail bounces and may also result in the receiving mail server becoming listed in public blacklists. In-line rejections, however, do not have to worry about e-mail forging since the sending server that was used by the sender to send the original e-mail generates their corresponding non-delivery receipts. Due to the

severe consequences of sending e-mail bounces, we have opted to use the in-line rejection method to notify users of SMTP rejections in our anti-spam solution.

4.5 Summary

In this Chapter, we modeled the SMTP transaction between a sending and receiving server. In particular, we examined how a legitimate mail server goes through five steps in an SMTP transaction to send, accept and reject mail. We will use this information in the next Chapter to help distinguish between legitimate and spam zombie transactions, as it is our hypothesis that zombies generate non-conformant SMTP transactions. We also looked at two different methods of handling SMTP rejections and explained why the in-line rejection notification method is the better of the two. Specifically, in comparison to e-mail bounces, the in-line rejection method is a safer means of notifying the user that their e-mail is blocked.

Chapter 5: Spam Zombie Behaviour Analysis

5.1 Introduction

In the previous Chapter, we examined the SMTP transaction according to RFC 821 [53], RFC 822 [54] and RFC 2821 [55] and modeled the transaction in five sequential steps. We also reviewed the different ways that an SMTP transaction could be rejected and looked at the different sender notification mechanisms used to notify senders that their mail could not be delivered. In this Chapter, we describe the environment and the methods we used to identify and analyze spam zombie behaviour. In particular, we capture three intervals of spam flow information from one of our filtering servers and compare its SMTP transactional flow with the five step model described in the previous Chapter. In the next Chapter, we will present our anti-spam technique that makes use of this comparative analysis.

5.2 Mail Environment

To study the behaviour of spam zombies, we use a live mail infrastructure consisting of two multi-layer inbound mail filtering servers running on identical platforms, where each server is equally responsible for receiving e-mail for one large global domain. The mail environment that we used is described in greater detail in Chapter 7.

We conducted our analysis in March 2007, when this domain had 65333 active users and our mail filtering servers blocked a total of 16,364,263 e-mails between February 9, 2007 to March 11, 2007. It should be noted that we elected to perform our analysis from the perspective of a receiving server of a large domain rather than the perspective of a spam zombie. This was done for several reasons. First, there are many different variants of spam zombies [56] that can be found on the Internet today, thus analyzing a few zombie types would not produce an accurate representation of general spam zombie behaviour. In contrast, by conducting the analysis on the receiving servers of a large domain, it is highly likely that our servers will interact with many distinct zombie sources and types. Second, in order to accurately find behavioral discrepancies between spam zombies and legitimate mail servers, we needed to ensure that the receiving server is RFC compliant. The simplest way to accomplish this is to perform the analysis from receiving servers that we run and manage ourselves.

As described in Section 2.4, a spam zombie is typically a non-standard mail server that is used to send spam by performing SMTP transactions with a receiving server. Such transactions are typically performed in a noticeably different manner from the five step transactional model described in Section 4.2. In the subsequent sections we will describe the method we used to identify and analyze spam zombie behaviour.

5.3 Packet Capturing

Our analysis begins by setting up a packet-capturing tool called Ethereal [57] on one of our mail filtering servers and running the application for three 10-minute intervals during the morning, afternoon and evening time periods of a regular business day. We selected

a 10-minute interval where, during that interval, our server handled over 10000 distinct connections providing us with a substantial amount of data. We also decided to use the three time periods where e-mail volumes fluctuated the most. For each packet capture, all packet streams were manually classified according to e-mail type. The different types included: legitimate, spam and miscellaneous. A brief description of each type is provided below.

5.3.1 Legitimate Packet Stream

The packet streams captured in this category consist of legitimate e-mail communications. It was observed that these captures correctly followed the five step SMTP transactional model described in the previous Chapter.

5.3.2 Spam Packet Stream

The packet streams captured in this category consist of unsolicited e-mail communications. We identified and distinguished spam captures from legitimate captures by looking for SMTP reject commands sent from the receiving server to the sending server. It was observed that the spam e-mails that we received could be further classified into three distinct groups, which includes scam spam, image spam and virus spam. We provide a brief overview of each spam classification and analyze why we were able to classify thousands of spams into three categories in Section 5.4.5.

5.3.3 Miscellaneous Packet Stream

The packet streams captured in this category includes network monitoring and filtering communications such as DNS blacklist look-ups where the receiving server queries an external DNS blacklist server to determine if the incoming IP address is listed on their blacklist. We distinguish these captures from both the legitimate and spam capture types by searching for non-TCP packets.

5.4 Packet Stream Analysis

For the purposes of this analysis, we will discard all legitimate and miscellaneous packet streams and focus on the spam packet streams. Specifically, we will compare the spam stream with the five-step SMTP transactional model we described in Chapter 4 and note key discrepancies between the two. The differences, along with relevant background, will be provided for each of the five steps in the sub-sections below.

5.4.1 Step 1: Session Initiation

When a user uses a sending server to send mail to a recipient, the sending server must find the recipient's mail server responsible for handling mail for the recipient's domain. The sending server is able to find the recipient's inbound servers by looking up a special DNS record of type MX for the recipient's domain, which provides a list of hostnames of the inbound mail servers. An example DNS MX record for the domain `foo` is as follows:

```
foo.com MX IN 600 mail.foo.com
```

In the above example, the first field (`foo.com`) specifies the recipient's domain, the second field indicates that the DNS record is of type MX, the third field indicates that the record belongs to the Internet class (IN), the fourth field indicates the time to live (i.e. 600 seconds) for that particular record before it expires and a new record should be retrieved. The final field indicates the hostname of the receiving server used to accept mail on behalf of that domain.

By initiating a port 25 connection to any of the recipient's inbound servers, the sending server can attempt to establish an SMTP transaction with the recipient's server for the purpose of delivering the mail to the intended recipient. If the connection is accepted then the sending server must wait for a banner with SMTP code `220` to be presented by the receiving server.

It was noticed that in some spam streams, the spam source was found sending commands to the receiving server prior to receiving the banner. Furthermore, it was noticed that if an MX record pointed to multiple hosts where one host (or potentially more) was non-responsive and the sending server tried to communicate with said host (or hosts), then the spam source would deviate from RFC2821 by simply quitting without trying to establish an SMTP communication with an alternate receiving host.

5.4.2 Step 2: Sending Server Identification

Our packet capture analysis presented some unusual SMTP transactional flow behaviour as well. In particular, several of the spam streams were found issuing a RST command after the HELO command. According to RFC2821, the RST command indicates that the

current mail transaction will be aborted and that any stored information such as senders, recipients, and mail data must be discarded from the buffers. Since nothing is stored in the buffer after a HELO command, it is illogical for any legitimate mail server to behave this way.

It was also observed that the spam streams had HELO parameter values that did not meet RFC 2821 [55] recommended host names. In fact, the spam stream HELO parameter values were noticeably different from the HELO values provided by legitimate servers. For example, many spam sources were found with a HELO parameter comprising a single word or character such as `friend` and `|`. Many other spam sources were caught lying about their identity. Specifically, we caught spam sources using our domain as their HELO parameter. This cannot be possible, since the only servers that can legally identify themselves with our domain name are our servers. We also caught a number of servers identifying themselves with freemail domain names such as gmail, yahoo and hotmail where their IP address did not belong to the identified domain name.

As noted in RFC2821, legitimate mail servers should be using their fully qualified domain name or an address literal as their HELO parameter. Although most spam sources did not use either of these two as their HELO parameters, we must still consider the possibility that spam zombies could eventually be developed as an RFC compliant mailer. We note that even if spam zombies become RFC compliant, the HELO parameter will still help us determine whether the sending server is a spam source or not. Specifically, a spam zombie on a home user machine would HELO with a generic fully qualified domain identifier that indicates that the machine resides on a dynamic IP pool

such as `dsl.ispserver.com`. We assume that legitimate mail servers should reside on a static IP address so that its clients may be able to communicate with the server without having to re-configure the server address each time the address changes. In other words, mail from a dynamic IP address should be considered suspicious and the source should be flagged as a spam source. Therefore, even if a spam zombie becomes RFC compliant, the HELO parameter will still provide insight as to whether the sending server is a zombie machine or not.

To determine if a source used its fully qualified domain name as its HELO parameter, one can simply perform a reverse DNS query on the source's IP address and compare the response with the parameter. Reverse DNS is essentially the opposite of standard DNS where you turn an IP address into a hostname.

Standard DNS lookups uses DNS type 'A' records, which look like:

```
host.foo.com A 192.0.2.25
```

whereas reverse DNS lookups use DNS type 'PTR' records, which look like:

```
23.2.3.123.in-addr-arpa.
```

A reverse domain lookup for a particular IP address is performed by reversing the IP address and appending in-addr.arpa. For example, if we wish to find the reverse DNS entry for 123.23.21.2 then we simply request a DNS lookup on 2.21.23.123.in.addr.arpa. Such a query will result one of three different responses, which include a reverse mapping, `NXDOMAIN` or server fail. A reverse mapping is the hostname residing on that particular IP address whereas `NXDOMAIN` is the response given when no RDNS record exists. A server failure is presented when the DNS server that is queried cannot provide a

response. In such case, the DNS server cannot be determined whether a reverse mapping record exists or not.

Ensuring that a mail server has a reverse DNS record is standard industry practice and is recommended by some of the top anti-spam groups such as the Messaging Anti-Abuse Working Group (MAAWG) [59]. In fact, some of the largest service providers in the world such as America Online (AOL) have implemented filters that block e-mail from mail servers that have no reverse DNS. The main reason for this is that virtually all e-mail from such sources is spam.

5.4.3 Step 3: Sender Identification

Although the spam streams demonstrated inconsistent use of upper case (MAIL FROM) and lower case (mail from) MAIL FROM commands, we noticed that for the most part, spam sources consistently identified the sender to the receiving server in accordance with RFC 821 [53], RFC 822 [54] and RFC 2821 [55]. Although spam sources have shown inconsistent usage of case for the MAIL FROM command, such inconsistency is still in accordance with the above-mentioned major SMTP standards and as a result, may be performed by legitimate mail servers as well. As such, this discrepancy will not be further considered in this thesis.

5.4.4 Step 4: Receiver Identification

Similar to the sender identification analysis, we note that there was inconsistent use of upper case (RCPT TO) and lower case (rcpt to) RCPT TO commands but that the majority of spam sources had consistently identified recipients to the receiving server in

accordance with RFC 821, RFC822 and RFC 2821. Although spam sources have shown inconsistent usage of case for the RCPT TO command, such inconsistency is still in accordance with the above-mentioned major SMTP standards and as a result, may be performed by legitimate mail servers as well. As such, this discrepancy will not be further considered in this thesis.

5.4.5 Step 5: E-mail Content

The discrepancies in the e-mail content of legitimate and spam communications also helped in verifying that a particular packet stream was legitimate or spam. As discussed in Section 5.3, the e-mail content of spam streams was classified in one of three distinct groups. These groups are:

1. Scam spam [9]
2. Image spam [60], and
3. Virus spam [10].

A brief description of each classification group and analysis as to why we were able to classify thousands of spams into these categories will be provided in the following sub-sections.

5.4.5.1 Scam spam

This type of spam, also known as phishing, is defined as an attempt to fraudulently acquire sensitive information, such as personal identity data and financial account credentials using both social engineering and technical subterfuge [18]. Specifically, phishing scams are masqueraded as a trusted institution, giving the recipient the

impression that there is a real need for such information. Most of these scams contain a link that the user is urged to click. The link takes the victim to a fake website that is designed to look like the real thing. The only difference being that any personal information that the user provides is routed directly to the scammer who will then use it to steal the victim's identity, money, records or anything else they can get. According to the Anti-Phishing Working Group (APWG), the banking industry has lost over 200 million dollars in 2005 as a result of scam spams [18]. As a result, it is not surprising that spam zombies are being used to send scam spam in large volumes.

5.4.5.2 Image spam

This type of spam attempts to bypass content filtering by embedding words into images. Such spam would contain one or more images describing what the spammer is trying to sell and how the end user can purchase the product. Due to the current limitations of optical character recognition technology, anti-spam filters have a difficult time distinguishing between good and bad images [60]. As such, spammers are using zombies to distribute these types of spams because of the high probability of getting such spams through existing filters.

5.4.5.3 Virus spam

These spams contain malicious attachments or links where if the attachment is opened, malicious code will be installed on the user's machine. Likewise, if the user clicks on the link, then a web browser vulnerability may get exploited and, as a result, the user's machine could potentially become infected. In order to overcome existing spam filtering

techniques such as DNS blacklists, a fresh supply of compromised machines is needed. One means of doing this is by propagating an infected payload from one machine to another. In other words, a compromised machine may be instructed to distribute virus spam in the hopes of generating more spam zombies that can be used to send the two types of spam identified above.

Although existing filtering techniques can help distinguish discrepancies between legitimate and spam e-mail content, such filtering techniques are not enough because today's spam being delivered by spam zombies are specifically designed to overcome such solutions.

5.5 Summary

In this Chapter, we introduced our approach to detecting spam zombie behaviour. Specifically, we captured packets from one of our mail filtering servers and classified each packet stream into one of three categories: legitimate, spam and miscellaneous. We then examined the spam streams and compared the SMTP transactional flow of these streams with the five step model we introduced in the previous Chapter. Our analysis shows discrepancies in the Session Initiation, Sending Server Identification and E-mail Content steps of the model. In the next Chapter, we will introduce a new filtering technique that uses machine learning to exploit these findings.

Chapter 6: Design Decisions

6.1 Introduction

In Chapter 4 we modeled legitimate mail server behaviour according to the five step SMTP transactional model. In Chapter 5 we learned through experimentation that spam zombies exhibit behavioural characteristics that deviate from the five step model. Specifically, spam zombies were found to produce non-standard server identification values and generate e-mail content that could be classified into one of three distinct categories. We also noticed that some spam zombies deviated from the sequential five step model by issuing valid SMTP commands at unusual times. In this Chapter, we take a closer look at our findings from the previous Chapter and describe how these observations were used when we designed our new filtering technique.

6.2 Design Considerations

Our goal is to develop a new spam filtering technique that can be integrated into an existing anti-spam filtering solution to specifically detect spam zombie behaviour and block mail from such sources. The process of designing such a technique requires an understanding of legitimate mail server behaviour. This was presented in Chapter 4 as the five step SMTP transactional model. In Chapter 5, we observed spam zombies in a

live environment and described how their behaviour deviated from the five step SMTP model. From our experimentation, we learned that there were three key aspects of the SMTP transactional model where spam zombies clearly deviated from the five-step model. Specifically, spam zombies produced non-standard server identification values, their e-mail content was distinct from legitimate mail and in some instances, and SMTP transactional flow did not follow the sequential five-step model. In this Chapter, we take a closer look at these observations and describe our design decisions.

6.2.1 E-mail Content

In Chapter 5, our experimentation showed that e-mail content from spam zombies could be classified into one of three distinct groups: image spam, scam spams and virus spams [9][60][10]. Further to this classification, we developed a hypothesis to explain why these classes of spam were the preferred choice of spam zombies. Specifically, we hypothesized that these spams were designed to pass through existing content filtering techniques, produce new spam zombies and/or generate proven sources of revenue.

Although our classification provides useful information that can help detect spam zombies, we note that in Chapter 3, spammers have proven their ability to evolve spam content to overcome new filtering approaches. Consequently, we hypothesize that developing a spam filtering technique to detect a specific class of e-mail content would only be a temporary solution until spammers develop new content to bypass such a filtering mechanism.

Furthermore, with legitimate e-mail encompassing a wide range of writing styles, formatting, subject matter and languages, we conclude that finding a generalized

legitimate e-mail content pattern from which we can use to continually distinguish all legitimate e-mail content from spam content is not a feasible approach. Nonetheless, we decided to use e-mail content analysis as a separate filtering layer on a multi-layer filtering server, which also comprises our filtering technique, allowing us to isolate our filtering technique for comparison purposes yet provide the added benefit of additional filtering.

6.2.2 SMTP Transactional Flow

According to our five step SMTP transactional model described in Chapter 4, all SMTP transactions should follow the following five steps in sequential order:

1. Session Initiation
2. Server to Server Identification
3. Sender Identification
4. Receiver Identification
5. E-mail Content.

However, as mentioned in Chapter 5, some spam zombies were caught deviating from this five-step model by issuing legitimate commands at unusual times. For example, we observed that some spam zombies had issued the RST command after the HELO command, which exemplified behaviour that no legitimate mail server would ever need to perform. However, since this type of activity is not against the RFC 821 and 2821 standards, it is possible that a legitimate mail server may exhibit such behaviour. For this reason, we decided not to use this discrepancy as part of the design of our new filtering technique.

6.2.3 Server Identification module

According to the five step SMTP transactional model described in Chapter 4, a legitimate mail server must identify itself to the receiving mail server with its fully qualified domain name. If the server does not have a fully qualified domain name, then it may use an address literal. In Chapter 5, we discovered that spam zombies either lie about their identity or identify themselves using non-standard text that does not comply with our five-step model.

With legitimate server identification being either a fully qualified domain name or an address literal, it is possible to develop a generic pattern for such behaviour. Specifically, we propose a machine-learning algorithm that is capable of learning the server identification values of both legitimate mail servers and spam zombies. However, to ensure that a spammer cannot easily overcome our machine learning technique, we include additional checks in our approach. In particular, we prevent spam zombies from forging fully qualified domain names by performing a reverse DNS look-up on their IP address and verifying that their reverse DNS value matches the server identification value they provided. We note that reverse DNS entries are typically the fully qualified domain name of the IP address and are stored on a DNS supplied by either the user or their service provider. In other words, it is non-trivial for an attacker to compromise a user's machine and alter their reverse DNS. We also prevent spam zombies from forging and improperly using an address literal as its identification value by verifying that the sending source is RFC 2821 compliant. Specifically, we verify that the source does not have a

reverse DNS mapping and that the sending server IP address matches the IP address in the address literal.

6.3 Machine learning

In this section, we look at machine learning algorithms that are capable of classifying server identification values into two categories: legitimate mail and spam. Such algorithms will be trained with thousands of sets of identifier values where each value is pre-classified according to one of the two groups. Once training is complete, the algorithm should be able to accurately and efficiently detect whether a server identification value is from a spam source or not. Since training data are pre-labelled, we will need a learning algorithm where training is performed by providing it with an input value and a corresponding expected output. Such algorithms are known as supervised learning algorithms and will be described in greater detail in the following sub-sections.

6.3.1 Supervised Learning

In supervised learning, the learning algorithm is trained with classified data for the purposes of developing a classification function, which can be used for classifying unknown data. Specifically, training data are first manually classified into groups. For example, spam related learning algorithms are trained with two groups of classified data: legitimate and spam. Next, the classified data are applied to the input of the learning algorithm while its corresponding group label is used as the expected output. If the actual output of the learning algorithm does not match the expected output, the weights corresponding to each of the input values are adjusted using the algorithm described in

Section 6.3.1.1. The weights are continually adjusted until the actual and expected outputs are equal or a maximum number of iterations is exceeded. This process is referred to as supervised learning because the algorithm is learning a pre-determined output for a set of inputs.

Once learning is complete, the learned model is cross validated against classified test input. Specifically, each test input also has a corresponding expected output. Unlike the training input however, the test input does not adjust the weight values of the learning algorithm if the expected output does not match the actual output. As such, applying test input on the learned model and examining its output, helps determine how well the model learned the trained data and how well the system generalizes to new data.

One of the biggest challenges with supervised learning is gathering and pre-classifying training data. In particular, acquiring and classifying such a large set of accurate data is often a difficult and tedious task. It is also important to ensure that the training data are as close to the actual data for which the algorithm will be operating in. With this in mind, we have developed an environment, described in Chapter 7, that can provide us with large set of data as well as a means for classifying said data. In addition to accuracy, we must also ensure that our machine-learning algorithm can efficiently classify data without compromising e-mail delivery service. Specifically, during our experimentation, we measured the number of rejected incoming SMTP connections to our filtering server to determine the impact our filtering technique has to e-mail delivery service. Rejecting SMTP transactions for performance reasons was considered unacceptable. An incoming connection is rejected when our filtering server does not have

the processing capability to accommodate the incoming request. To meet our processing efficiency requirement whereby incoming SMTP connections are not rejected, we limit our analysis of supervised learning to one of the simplest of architectures, a single perceptron. In other words, we selected the single perceptron over more sophisticated algorithms such as multi-layered neural networks and Bayesian learning for performance reasons. Through experimentation described in Chapter 7, we were able to find learning parameters for the single perceptron that produced high blocking rates and that did not reject any incoming SMTP requests.

6.3.1.1 Single Perceptron – General Overview

The perceptron, as illustrated in Figure 6-1, consists of a neuron having a set number of inputs with corresponding weights, which are used to calculate the most likely classification of data.

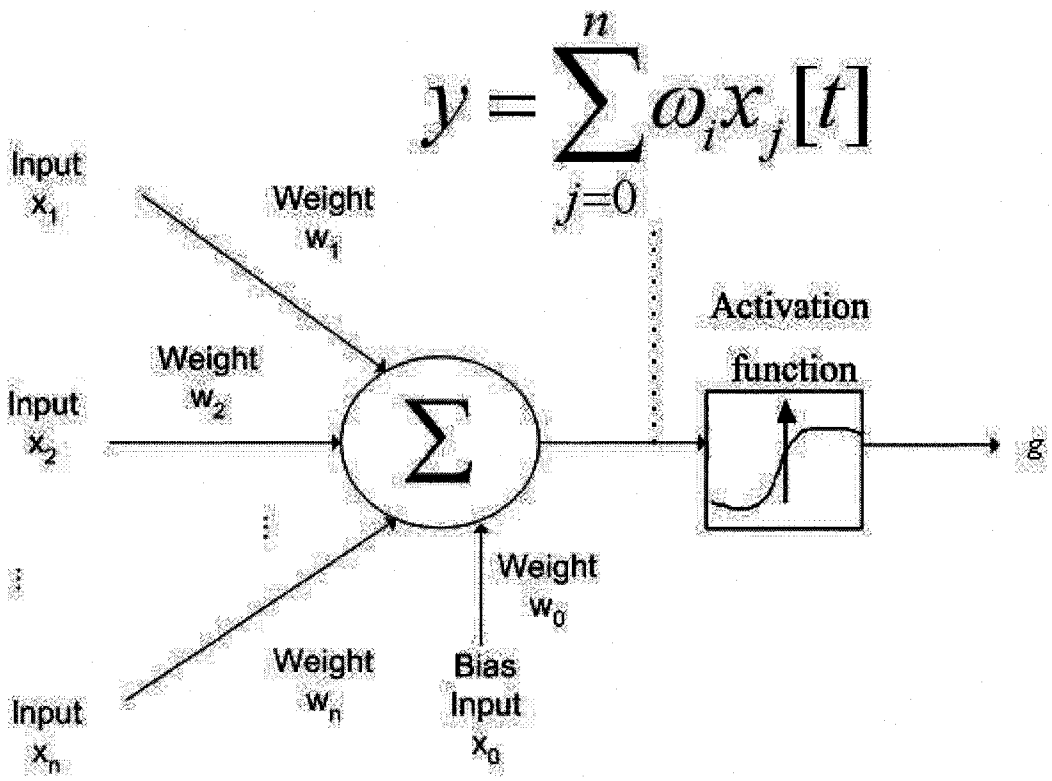

Figure 6-1: Single Perceptron Architecture

As shown in Figure 6-1, each input is multiplied by a corresponding weight and applied to the input of the neuron. The neuron then computes the sum of all its inputs and then applies the sum to an activation function to determine an output value. The activation function can be a linear or non-linear function. One problem with linear activation functions is that they can only solve problems where the solution is linearly separable. Non-linear activation functions face the same limitation where they represent only 'soft' linear separators [63]. For example, the XOR function is a function that is not linearly separable and as a result, cannot be solved by a perceptron with a linear or non-linear activation function. However, despite this shortcoming, perceptrons provide an

advantage where they can fit to any linearly separable set via a simple learning algorithm. In particular, by using the non-linear sigmoid function as our activation function, our learning can be formulated as an optimization search in weight space, also known as the gradient descent algorithm, which is described in Equation 6-2. As such, in our example, we will use a non-linear sigmoid function as our activation function and specifically describe what role it performs in training the perceptron.

Training

When using a non-linear activation function, the perceptron is able to employ a gradient method for training where the goal is to adjust the weights to reduce the error between the expected and actual outputs. Specifically, the true gradient is evaluated on a set of training example and the weights for each input are adjusted accordingly until a stopping condition is met.

In order to train the network, a corpus of pre-classified data is provided as input to the system. For each set of inputs, an output value will result. An error value is subsequently computed by using the sum of squares error equation, which is presented as equation 6-1 below [61]:

$$E(\omega) = \frac{1}{2} \sum_{i=1}^{n} \Sigma_e (y_e - g(x_i))^2$$

Equation 6-1: Mean Square Output Error

The gradient descent method is able to reduce the squared error output of Equation 6-1 [62] by calculating the partial derivative of the mean square output error equation with respect to each weight. We note that we are able to perform a partial derivative of

equation 6-1 because we are using a sigmoid function as our activation function where the sigmoid function is differentiable. A high level overview of the gradient descent algorithm follows.

For a single perceptron with weights *w*, perceptron output *y*, activation function *g* and activation function output *z*, we use the following algorithm by applying training example *t* to each input x_1, x_2, x_3, x_4 where $t \in Z^4$ [63].

Algorithm:

For each training example *t*, do:

$$y = \sum_{j=0}^{n} \omega_i x_j[t]$$

$$z = g(y)$$

$$\varepsilon = y[t] - z$$

$\omega_{j+1} = \omega_j + \alpha \varepsilon g'(y) x_j[t]$ [62] where (is the learning rate.

Iterate until stopping condition met.

Equation 6-2: Gradient Descent Algorithm

$$\text{where } g(y) = \frac{1}{(1+\ell^{-y})}$$

Equation 6-3: Sigmoid Activation Function

$$\text{where } g' = g(1-g)$$

Equation 6-4: Derivative of Sigmoid Activation Function

The gradient descent algorithm is described as Equation 6-2 and uses Equations 6-3 and 6-4. In Equation 6-2, we start by simply calculating the sum (i.e. *y*) of all the perceptron's inputs multiplied by their corresponding weight. In Equation 6-3, we take

the output y value from Equation 6-2 and apply it to the activation function g. Using the activation output value $g(y)$ we computed in Equation 6.3, we go back to Equation 6.2 and compare the computed output with the expected output. The difference between the expected output and the actual output is stored in the ε variable. Next, we calculate the weight adjustment value. This weight adjustment equation is the partial derivative of Equation 6-1. We note that this equation makes use of a learning rate, which dictates how quickly the network converges and as a result, we must use caution when selecting a learning rate. For example, if the learning rate is too large then the existing weights are overtaken by the delta values. However, if the learning rate is too small (close to 0) then the algorithm takes a long time to converge. As a final step to the gradient descent algorithm, we repeat the previous steps of the algorithm until a stopping criterion is met. Specifically, we stop once we reach a maximum number of iterations or our output matches our expected output.

Further motivation for using the single perceptron comes from [35] where the perceptron was used as a proven means for classifying information in the e-mail domain. We note that the reasons for using a perceptron in [35] align with our simple and efficient design objectives from the previous section and as a result, our filtering technique will be designed to use a single perceptron to distinguish spam zombies from legitimate mail servers. We note that the difference between [35] and our technique is that in [35], training is not performed continuously and in real time whereas it is in ours. Furthermore, [35] uses static rule sets as its inputs whereas we do not use such rule sets for our inputs.

6.4 Architecture

Our spam filtering technique is integrated with an existing corporate spam filtering solution. The integration of our new filtering with the existing filtering solution is illustrated in Figure 6-2.

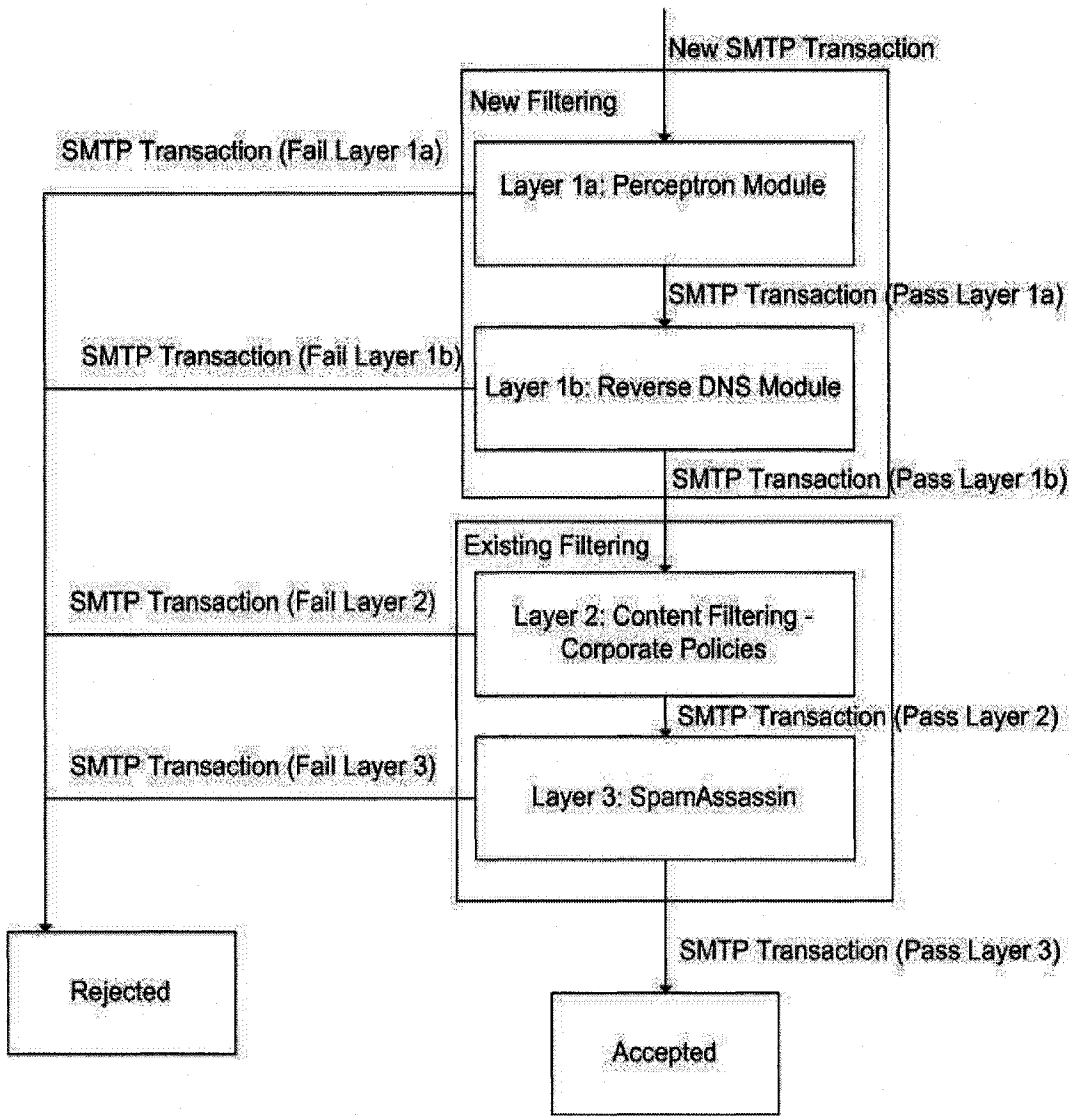

Figure 6-2: Integration of Filtering Techniques

In Figure 6-2, our filtering technique is the first of three filtering layers of the corporate filtering solution. Each of the three layers must be passed sequentially in order for the filtering server to relay the message to the intended recipient. The first layer, identified as layer 1 in Figure 6-2, is our perceptron module, which determines whether the sending server's identification value is considered legitimate or illegitimate. If the value is considered illegitimate, our technique rejects the SMTP transaction. If however, the value is considered legitimate then the SMTP transaction is then checked against our Reverse DNS module, identified as layer 1b in Figure 6-2. In the Reverse DNS module, we query the reverse DNS of the sending server to verify that the sending server's identification value is consistent with RFC 2821. If the value is inconsistent with RFC 2821 then the Reverse DNS module rejects the transaction. The main purpose behind the Reverse DNS module is to prevent spammers from simply forging server identification values to overcome layer 1a.

If however, the server identification value is considered legitimate by the perceptron and reverse DNS module then two events occur. First, we check to see if the sending server's IP address is listed in the Spamhaus list. If the sending server's IP address is listed then we set our perceptron's expected output to indicate SPAM (i.e. 1) and train our perceptron to learn that the server identification value is indicative of spam zombie behavior. We note that the Spamhaus list does not perform any filtering in our integrated server and is simply used as a means to train our perceptron of spam zombie server identification values.

In addition to the Spamhaus list lookup, our SMTP transaction is sequentially scanned against the second (i.e. content filtering) and third (i.e. spamassassin) layers of spam filtering, identified as layers 2 and 3 respectively in Figure 2. If the SMTP transaction is able to pass both the second and third layers and the sending server's IP address was not listed in the Spamhaus list, we then set the expected output of our perceptron to NON-SPAM (i.e. 0). Using said expected output, we then train our perceptron to learn that the server identification value that we received is not indicative of spam zombie behaviour. It should be noted that using the Spamhaus list to label the training data will have some errors in it. For example, new spam zombies that are not listed on the Spamhaus list and are able to get their spam through the additional layers of filtering of our filtering system (i.e. Layers 1b, 2, and 3), as shown in Figure 2, could be mislabelled by our training algorithm as NON-SPAM and used in the training set. The presumption is that these cases should be rare or outliers and the general training data labels will be correct. The single layer perceptron training should therefore not be terribly influenced by these errors.

In the following sub-sections, we will describe the architecture of layers 1a and 1b of Figure 6-2.

6.4.1 Layer 1a: Perceptron Module

In layer 1a, of Figure 6-2, a single perceptron is used to learn and predict spam zombie server identification values. A flow-chart for our perceptron module illustrating how the perceptron works from a high level perspective is presented in Figure 6-3 and described in detail below. We note that the perceptron module is broken up into three main areas.

The first area is called 'loading' and is responsible for creating a new perceptron object, loading its weight values and generating its inputs. The second area is called 'decision making' where based on the given inputs, the perceptron must decide whether to accept or reject the current SMTP transaction. The third area is called 'training', which is responsible for training the perceptron to learn good and bad server identification values. We note that the training module is purposely positioned after the decision module so that we can continuously re-train our perceptron in real-time based upon the outcome of the Spamhaus list [6] query and other filtering layers on the filtering server. More detail on each of the three sections is described in the following sub-sections.

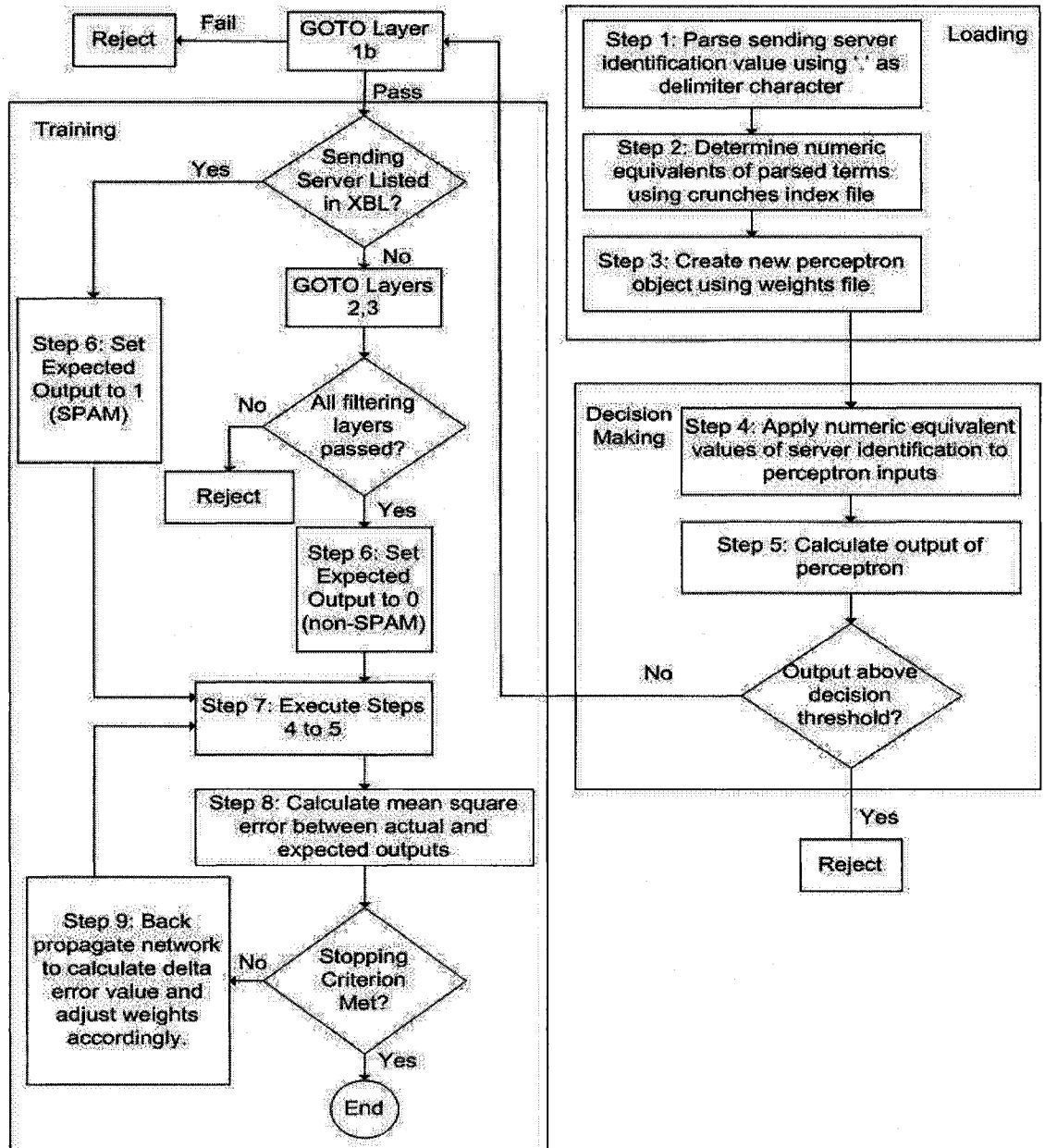

Figure 6-3: Information Flow of Perceptron Module

6.4.1.1 Loading

The purpose of the loading portion of the algorithm is to create a new perceptron object, initialize its weight values and convert the provided input into a corresponding numeric array reference that can be used by the learn and run functions. Specifically, in Step 1 of Figure 6-3, the algorithm is provided with the sending server's identification value as its input and parses the input into sub-domains, using the '.' as a delimiter. For example, a server identification value of `static.mail.example.com` would be parsed into four separate terms `static`, `mail`, `example` and `com`. In Step 2, each sub domain term is then stored in an index file called `crunches` on the filtering server and assigned a corresponding numeric value. For example, the server identification of `static.mail.example.com` could be indexed as 1, 2, 3 and 4.

Next, in Step 3, we create a new perceptron object and initialize it with pre-defined weights, learning rate and maximum iteration count. The pre-defined weights for each input of the perceptron is stored on a file called `weights` on the filtering server and are the most recent weights used by the last perceptron object. The learning rate was defined as the parameter α in Equation 6-2. The maximum iteration count variable is used to prevent our perceptron from spending a long period of time training a given set of input, which could affect performance of our filtering technique. We note that in order to perform on-line learning, we must create a new perceptron object for each SMTP transaction and preserve the weight values of the most recently learned training example. Once we have completed Step 3 of Figure 6-3, our perceptron is ready to either learn or make a decision. We describe both options in the subsequent sub-sections.

6.4.1.2 Decision Making

The decision making portion of our algorithm is called after a new perceptron is created. In Step 4 of Figure 6-3 we apply the numeric values of each sub domain from the server identification value to the inputs of the perceptron.

In Step 6, the algorithm calculates the net-sum for the perceptron and subsequently applies the net-sum value to a sigmoid activation function to calculate the output. This output value is then used to determine whether the server identification value is to be accepted. Specifically, if the output value is below the decision threshold of the sigmoid activation function (i.e. 0.5) then the SMTP transaction is able to get through our perceptron filtering and is subsequently verified against layer 1b, the Reverse DNS check layer. If however, the output value is above the decision threshold value, the SMTP transaction is blocked.

6.4.1.3 Training

The training portion is executed after the creation of a new perceptron and after the perceptron's decision module and reverse DNS module have decided that the current sending server provided a legitimate server identification value.

The training algorithm begins by determining whether the sending server's IP address is listed in the Spamhaus list. As previously mentioned, the Spamhaus list is designed to list the IP addresses that are sending spam through illegal third party exploits, in other words, spam zombies. In Step 6, if the sending server's IP address is on the Spamhaus list then we set the expected output of our perceptron to '1' to indicate that the sending server is a known malicious host. If however, the sending server is not listed in the

Spamhaus list then we proceed to the final 2 layers of filtering on the filtering server. If the current SMTP transaction passes the final two layers of filtering and the sending server's IP address was not listed in the Spamhaus list, then in Step 6, we set the expected output of our perceptron to '0' to indicate that the sending server is likely to be trusted.

To train the perceptron, we run through Steps 7 and 8 of Figure 6-3. Step 9 indicates that we repeat steps 7 to 8 until a stopping criterion is met. Once a stopping criterion is met, training is complete. Our stopping criterion occurs when the perceptron output equals the expected output or a maximum number of iterations has elapsed.

6.4.2 Layer 1b: Reverse DNS Check Module

The Reverse DNS module was referred to as layer 1b in Figure 6-1 and is the second filtering layer of our filtering server. A high level overview of the module is illustrated in the flowchart in Figure 6-4. A detailed description of each step follows.

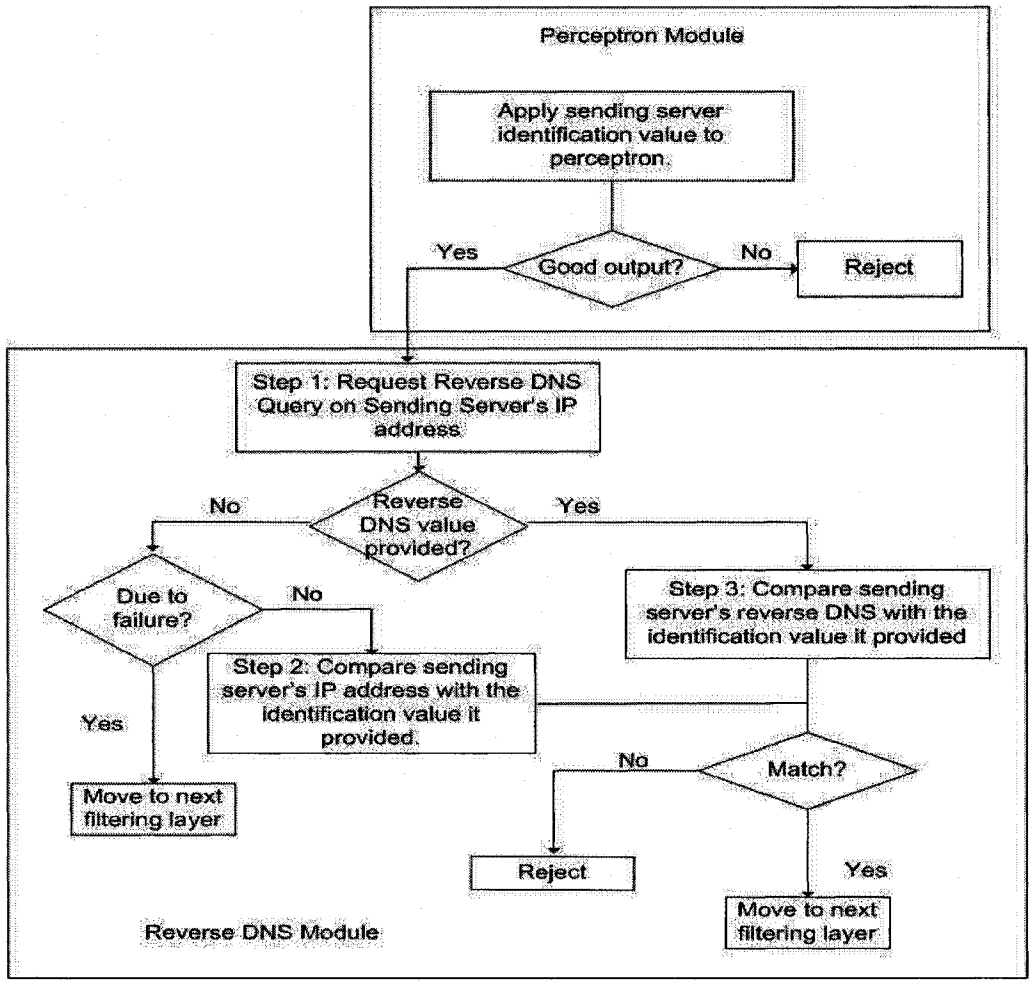

Figure 6-4: Information Flow of Reverse DNS Module

As illustrated in Figure 6-4, our filtering system will only execute our reverse DNS module if the SMTP transaction being analyzed has passed our perceptron module (i.e. layer 1a).

Our reverse DNS module begins with Step 1 in Figure 6-4 where the module performs a DNS query for the sending server's IP address. Specifically, the query will look for the reverse DNS record (PTR record) of the sending server. Details on how to

perform a reverse DNS query can be found in [67]. If the reverse DNS query did not respond with a value due to a query failure, then it is inconclusive as to whether the source is potentially legitimate or not and as such, the code exits our technique and proceeds to the next layer of filtering. If however, the sending server does not have a reverse DNS record and an NXDOMAIN error message was returned, then we know that no reverse DNS mapping exists for the sending server's IP address and code. At this point, our reverse DNS module then proceeds to Step 2. In Step 2, the reverse DNS module verifies that the sending server behaviour was consistent with our SMTP transactional model by specifying an address literal as its server identification value. Specifically, our code compares the sending server's IP address with the identification value the sending server specified. If there is no match then the SMTP transaction is rejected and if there is a match then the code exits our technique and proceeds to the next layer of filtering in the anti-spam server. If however, a reverse DNS mapping exists, then the code proceeds to Step 3, which will compare the reverse DNS value with the server's identification value to determine whether the two match. If there is no match then the SMTP transaction is rejected and if there is a match then the code exits our technique and proceeds to the next layer of filtering of our filtering server.

6.5 Implementation

In this section we will provide details to explain how the perceptron and reverse DNS modules were implemented. Specifically, for the perceptron module, we provide specific details to describe the perceptron architecture used and explain how each of the three main areas: load, train and decision-making were implemented. For the reverse DNS

module, we explain how the queries were performed and how decision-making was implemented.

6.5.1 Layer 1a: Perceptron Module

The architecture of the perceptron that we used is illustrated in Figure 6-5. Specifically, we used a single perceptron with four input values and a single output. Each input value is a numeric representation of a domain level term of the sending server's identification value and the output value is produced by a sigmoid activation function. We selected four inputs because we assumed that most legitimate fully qualified domain names have at most four domain level terms [64]. We note that a bias input, shown as input x_0 in Figure 6-1, was not included in our perceptron because with the exception of a rare circumstance where the first four terms of a server's identification value are the 0^{th} term of our index file, all four inputs will not be zero at the same time. More detail with regards to our index file will be provided in Section 6.5.1.1.

We also used a perceptron with a sigmoid function as our activation function for two reasons. First, the sigmoid function is non-linear and allows our perceptron to solve slightly more complex problems using a 'soft' linear boundary instead of a solid linear boundary in the solution space. The second reason why we chose the sigmoid function over other non-linear functions is that the sigmoid function is differentiable, allowing us to utilize the gradient descent training method described in Section 6.3.1.1. By using the sigmoid as the activation function, we were able to use the sigmoid decision boundary of 0.5 and ensure that our activation output was between the values of 0 and 1. In our

implementation, a value greater than or equal to 0.5 represented spam and a value below 0.5 represents legitimate mail.

The source code for our perceptron was written in the Perl programming language where the original source code was obtained from CPAN (`NET::AI::PERCEPTRON`) [65] but was heavily modified to suit our needs.

In the following sub-sections, we explain how each of the three areas described in the perceptron module architecture section 6.3.1.1 was implemented.

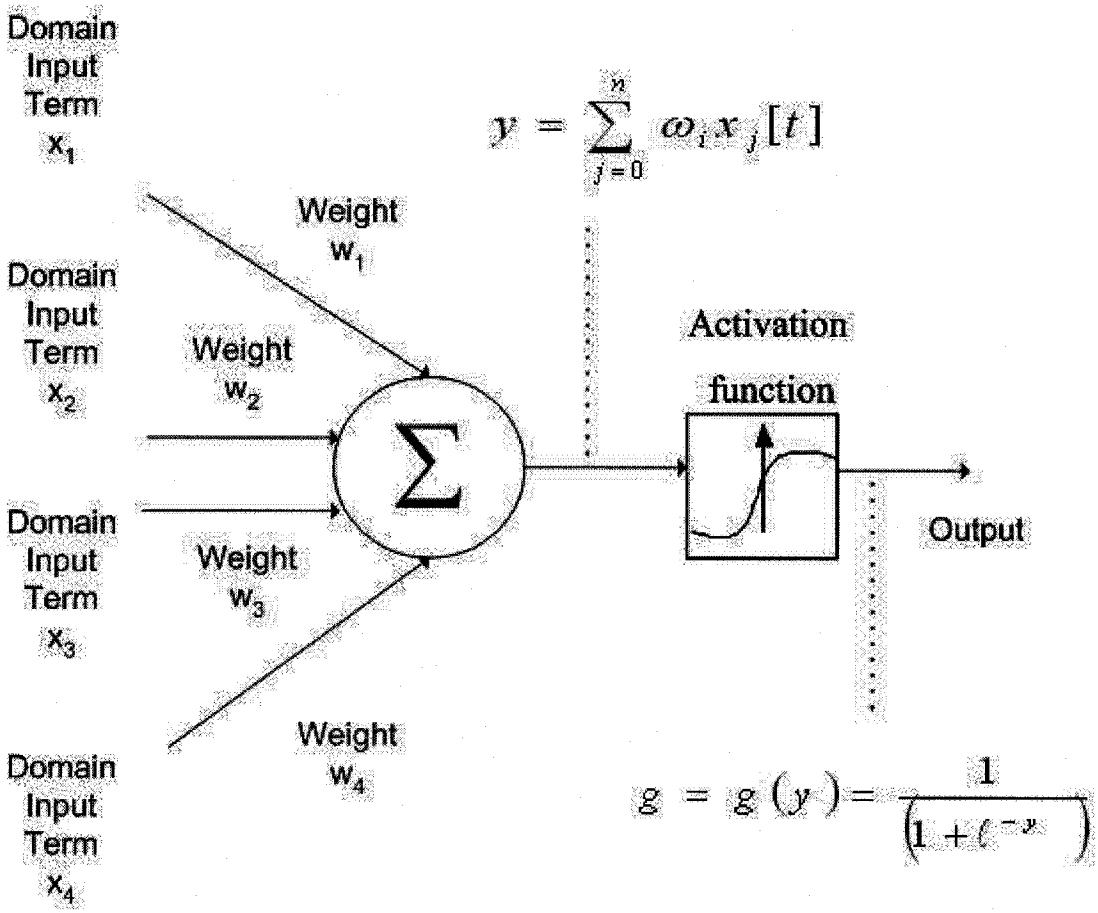

Figure 6-5: Implementation of Single Layer Perceptron

6.5.1.1 Loading

The loading portion of our algorithm illustrated in Figure 6-3, was implemented in two parts. The first part, which we will refer to as 'load', consists of loading pre-specified values into a newly created perceptron object. The second portion, which we will refer to as inputs, focuses on converting the server identification value into numeric inputs, which can be applied to the inputs of our perceptron.

Load

When the filtering server receives an SMTP connection from a sending server, the load function is initiated. This function creates a new instance of our perceptron object with default features. Specifically, it creates a new perceptron object with 4 inputs, a default learning rate and an empty weight vector. The default learning rate is determined via experimentation, which will be described in Section 7.3.

Next, the load function reads a text file on the filtering server called `weights` where said file stores the most recent weight values of our weight vector. A sample `weights` file is presented in Appendix B.

The load function parses each line of the weight file and only extracts the weight values, located on the right hand side of each line. These values are stored as the weight vector for the newly created perceptron object. At this point, we now have an up-to-date perceptron and are ready for further training or decision-making.

Inputs

As the filtering server and the sending server engage in an SMTP transaction, our technique waits for the sending server's identification value. Once this identification value is received, our code breaks up the first four left-most domain-level values of the server identification value. For each domain level value, our code scans through a file called `crunches` on the filtering server, which is an index of the all the values seen thus far. If the value already exists, then the code returns the associated index for that value. If however, the value does not exist then the code adds the value to the `crunches` file and creates a new index value for the domain-level value. The index returned from the `crunches` file is used to numerically represent the sending server's identification value and provided as input to our perceptron.

In the example below, the server identification value of `D99H3R51.hsd1.in.comcast.net` is broken up into four terms: `D99H3R51`, `hsd1`, `in` and `comcast`. Our code then looks up each of these terms in the `crunches` file and if there is a match then the index of the corresponding value is assigned to that term. For example, a scan through the `crunches` file, presented in Appendix B, determined that `comcast` was already assigned a value of 121 and as a result our code assigned said value to said term. Similarly, `hsd1` was already assigned a value 119 and `in` assigned a value 2. The term `D99H3R51` however was not found in the `crunches` file and as such was assigned the next available index value of 160.

```
o [2007.03.10 18:25:31] (111) Crunched term: D99H3R51 Crunched value:
160
o [2007.03.10 18:25:31] (111) Crunched term: hsd1 Crunched value: 119
```

o [2007.03.10 18:25:31] (111) Crunched term: in Crunched value: 2

o [2007.03.10 18:25:31] (111) Crunched term: comcast Crunched value: 121

As described in Section 4.3, spam filtering occurs in Step 5 of the five step transactional model and as a result, we were able to examine the body of the above-mentioned SMTP transaction. From our analysis, we were able to determine that the message was indeed spam and as such, expected that our perceptron would generate an output that would indicate as such. Specifically, we expected an output of 1 and cross validated our perceptron by verifying that the output logs produced a score of '1' for this particular SMTP transaction.

o [2007.03.10 18:25:31] (111) HELO of D99H3R51.hsd1.in.comcast.net. produced a score of 1

A similar cross-validation test was performed for the SMTP transaction with the server identification value of tag.bestbuy.com.

o [2007.03.10 18:26:45] (162) Crunched term: tag6 Crunched value: 222

o [2007.03.10 18:26:45] (162) Crunched term: bestbuy Crunched value: 223

o [2007.03.10 18:26:45] (162) Crunched term: com Crunched value: 5

o [2007.03.10 18:26:45] (162) Crunched term: Crunched value: 0

Again, by examining the body of the corresponding SMTP transaction, we were able to determine that the message was not spam and as such, expected that our perceptron would generate an output that would indicate as such. Specifically, we expected an output of '0' and cross validated our perceptron by verifying that the output logs produced a score of '0' for this particular SMTP transaction.

```
o [2007.03.10 18:26:45] (162) HELO of tag6.bestbuy.com produced a
score of 0
```

Decision Making

In the decision making stage, the load function is called to create a new perceptron object with the latest weight values. The server identification value is then parsed into four terms and scanned against an indexing file called `crunches` to determine a corresponding numeric value for each term. Each of these numeric values is then fed into our perceptron, which uses the sigmoid function as its activation function. The properties of the sigmoid function causes our output to be restricted between 0 and 1 where 0.5 is the decision boundary such that values above and below that threshold present a different decision. Specifically, if the perceptron determines that the identification value produces an output less than the decision boundary of 0.5, we then consider the value to be legitimate and the reverse DNS module is initiated and if the reverse DNS check passes then we prepare our perceptron for potential incremental training which will be described in Section 6.5.1.2.

In contrast, if the identification value produces an output greater or equal to 0.5 then the SMTP transaction is rejected. When an SMTP transaction is rejected by the perceptron, the rejection is labeled as `BAD HELO` so that when generating metrics in Chapter 7 to determine the effectiveness of our filtering technique, we simply parse the logs for such label.

6.5.1.2 Training

Stochastic Gradient

Our perceptron employs a stochastic gradient method for training, where the true gradient is evaluated on a single training example and the weights adjusted accordingly until a stopping condition is met. In other words, for each server identification training value, the perceptron continuously uses said value as its input until it either generates the desired output or reaches a pre-specified maximum number of iterations. Such pre-specified maximum number of iterations was determined by experimentation and described in greater detail in Section 7.3. At each iteration, an error and weight adjustment value are computed by comparing the actual output value with the expected output value using the formula described in equations 6-2 to 6-4.

Once the weight adjustment value is computed, the actual weight values are then adjusted. Upon updating the weights, the same training server identification value is used as input to the perceptron and the corresponding output is computed. If the actual output and the expected output do not equal then the weights are adjusted again. This process continuously iterates until the algorithm is able to generate an output equal to the expected output or the maximum number of iterations is reached. Since a new perceptron is created for each SMTP connection and stochastic gradient is used, we must preserve learned weight values after each training example. This is performed by the save function which will be described in the next sub-section.

In contrast, we note that standard gradient descent cycles through a set of training examples until a stopping condition is met, where each cycle through the set of training examples is called an epoch. We elected to use a stochastic gradient descent due to our large training corpus and because stochastic gradient descents are must faster than standard gradient descent [62].

In terms of training, if the sending server's IP address is listed in the Spamhaus list, the perceptron code from our filtering technique takes the server identification value and trains our perceptron to learn that that particular input corresponds to spam zombie behaviour. If however, the sending server IP address is not listed in the Spamhaus list and its SMTP connection is able to pass through all three layers of filtering then our training code takes the server identification value and trains our perceptron to learn that that particular input corresponds to a legitimate mail source. Consequently, by continuously training our perceptron in a real-time environment in the manner described above, we were able to generalize the Spamhaus list as illustrated in Section 7.7.1.

Save

Once training is complete, the save function is called to save the weights of our perceptron in the text file called `weights` as described in Section 6.5.1.1. We do this because each time our filtering server receives an incoming SMTP connection, our filtering technique is called and we must then create a new perceptron object. To ensure that we preserve the training we've done thus far, we save all trained weight values in a

separate file called `weights` where said weight values can be assigned to the corresponding weight array reference of the new perceptron object.

6.5.2 Layer 1b: Reverse DNS Check Module

The reverse DNS module is written in Perl and makes use of the `NET::DNS::Resolver` module [66]. This module first determines whether there is a reverse DNS entry for the sending server's IP address. If a reverse DNS entry exists, then the code compares the reverse DNS value with the value that the sending server identified itself as. If the two values match then we can proceed to potentially training the perceptron but if they do not match then the transaction is rejected because the transaction conflicts with our five-step model where the sending server failed to use its fully qualified domain name as its identification value. We label this rejection in our logs as `BAD RDNS`.

If a reverse DNS entry does not exist then our code receives the response `NXDOMAIN` and the server should have identified itself as an address literal. In such circumstances, the reverse DNS code will verify that an address literal was provided by the sender and that the IP address of the address literal matches the IP address of the sending server. If the two match then we can proceed to potentially training the perceptron but if they do not, then the transaction is rejected because the transaction conflicts with our five-step model where the sending server failed to use its IP address as an address literal as its identification value. We label this rejection in our logs as `BAD NXDOMAIN`.

Finally, if our reverse DNS module cannot determine whether the sending server's IP address has a reverse DNS then we proceed to training the perceptron since a reverse

DNS failure is inconclusive as to whether one exists or not. One reason why our module might not be able to determine a sending server's reverse DNS is that the sending server's DNS is down or there is a problem with the network [67].

6.6 Summary

In this Chapter, we presented a high level overview of the design decisions of our new filtering technique along with its implementation details. Our filtering technique is based upon our findings from Chapter 5 where server identification values from legitimate mail servers can be generalized into either a fully qualified domain name or an address literal. Using this observation, we developed a filtering technique that uses a perceptron learning algorithm to learn both legitimate server identification values and spam zombie identification values. To prevent forging as a means to overcome such filtering, we have included a reverse DNS function that ensures that the sending server is not lying about its identification. In the next Chapter, we will present results of our new filtering technique when used in a large corporate environment.

Chapter 7: Training and Experimentation

7.1 Introduction

In this Chapter, we examine the effectiveness of our filtering technique by comparing it against some of the filtering techniques described in Chapter 3. We start by describing the live corporate mail-filtering environment that we used for our testing as well as the configurations and processes that we implemented to allow us to gather useful metrics. We then illustrate how our filtering technique was trained to learn spam zombie behaviour and describe how it was comparatively tested against other techniques. We conclude by presenting and analyzing the results from our tests where the results include the number of SMTP transactions correctly and incorrectly blocked by each technique as well as generic characteristics of spam zombies that can assist us in our future work. We use these observations in the next Chapter where we discuss future work and present an overall conclusion to our thesis.

7.2 Test Environment

The test environment used to test our filtering technique is illustrated in Figure 7-1 where two live production inbound mail filtering server (i.e. Server 1 and Server 2), located in the de-militarized zone (DMZ) of a corporate network, were responsible for filtering mail

for a large corporate domain comprising over 35,000 distinct e-mail accounts. Also located in the corporate DMZ was a DNS-based blacklist that the inbound mail servers could use for performing DNS blacklist queries. The reason for positioning our filtering server in the DMZ is to minimize the number of routers and firewalls that a sending server would need to go through in order to engage in an SMTP conversation with the filtering server to prevent delivery performance issues. Specifically, each screening router and firewall adds additional delay to the processing of an SMTP transaction and by minimizing the number of said devices along the path from sending server to filtering server, we reduce the odds of a third party bottleneck affecting mail delivery service.

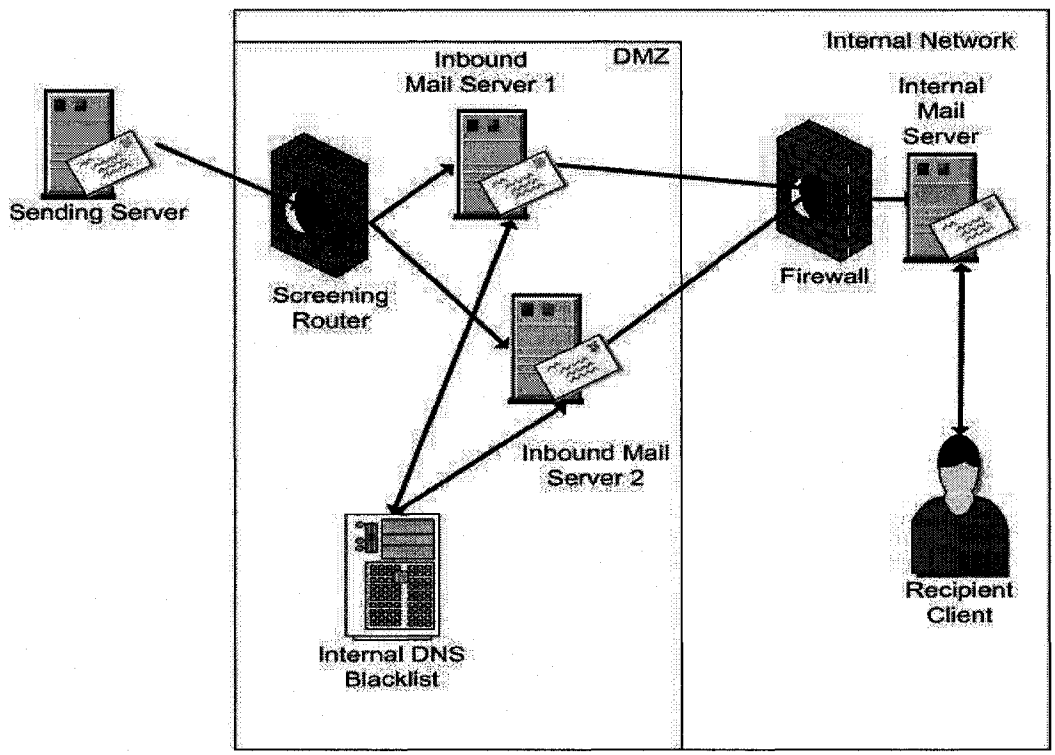

Figure 7-1: E-mail Test Environment

According to our test configuration illustrated in the figure above, a sending server would connect to one of our inbound servers via a screening router located at the perimeter of our DMZ. The purpose of the screening router is to provide a port blocking means to filter inbound traffic where in this case, the screening router is configured to allow SMTP traffic. As a result, the sending server is able to perform an SMTP transaction with one of the corporate inbound mail filtering server. During the transaction, the mail filtering server may decide to either discard the message or allow it into the internal network. Specifically, our filtering server was designed to initiate and conduct filtering just before the very end of the SMTP transaction or in other words, after Step 5 of the five-step transaction model as described in Section 4.3. If the message is discarded then a `550` reject message is sent from the receiving server to the sending server to inform the sending server of the failure. If the message was allowed, the inbound mail filtering server then initiates a new SMTP transaction with the internal mail server, which then receives and stores the message on disk allowing the recipient's client to retrieve the message.

Both Server 1 and Server 2 used three layers of filtering as shown in Figure 7-2. We note that Server 2 was the original filtering server that we traditionally use to filter mail for our corporate domain while Server 1 was a customized version of Server 2. Specifically, our technique was implemented as Layer 1 on Server 1. In comparison, the Spamhaus list was used as Layer 1 on Server 2. This is illustrated in Figure 7-2, which shows the three layers of filtering for the first and second inbound filtering servers.

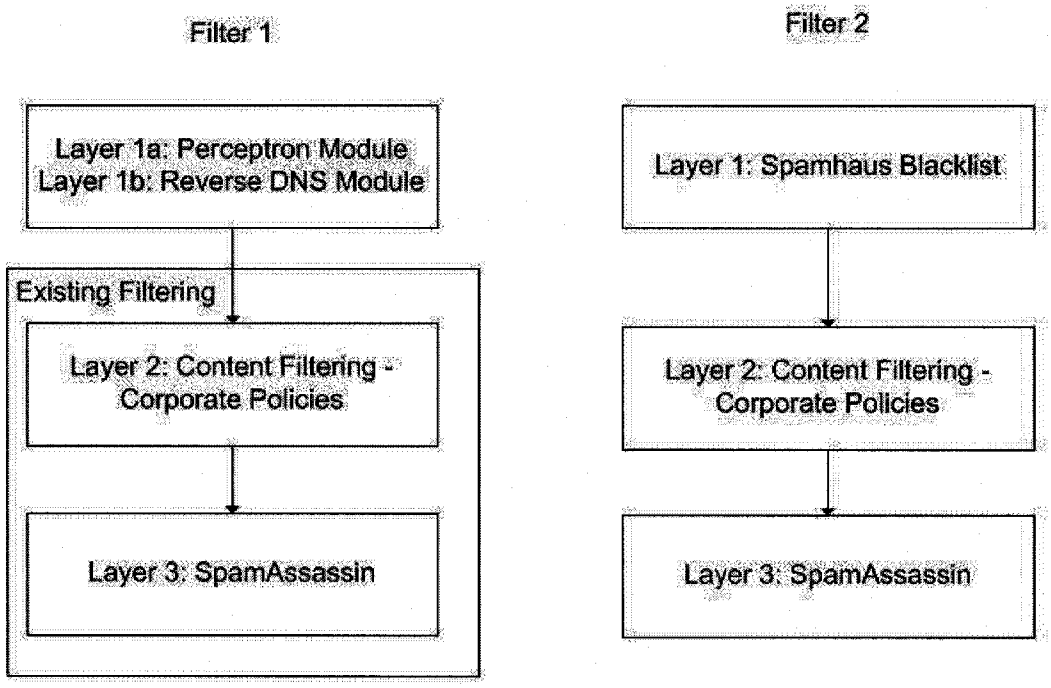

Figure 7-2: Filtering Layers of Filtering Servers

The second layer of Servers 1 and 2 used in our test environment comprised of a simple content-based filter designed to enforce corporate policies. This second layer was added for corporate security purposes. The third filtering layer was a default SpamAssassin 3 (Apache) [34] engine with a threshold value set at 6.2, which was determined through additional experimentation. Since the focus of this thesis is on our filtering technique we do not provide additional details about how we determined the 6.2 threshold level for our filter.

Both filtering servers were implemented such that each SMTP transaction would be filtered against each filtering layer in sequential order. Specifically, each SMTP transaction would be filtered against the first layer first and if that layer did not flag the

transaction then the transaction would be examined against the second layer and so on. Correspondingly, if the first layer filtering flagged an SMTP transaction as spam then the transaction would be rejected and the subsequent layers of filtering would not get applied to that particular transaction. As such, the second layer of filtering on the first server did not impact the filtering effectiveness of our filtering technique.

7.3 Determining Parameter Values

We conducted experiments in our test environment to determine the preferred learning rate and maximum iteration number that we should use for our perceptron. We conducted our experiment in the above mentioned environment on a single filtering server using a combination of three distinct learning rates and three distinct maximum iteration counts. In particular, we were interested in seeing the effect of having one large, medium and low learning rate and maximum iteration count value.

From Equation 6-2, we can see that a larger learning rate will create larger adjustment values, which will drastically alter the output after each iteration but will also make it more difficult to narrow in on the weight values that will result in an output equal to the expected output. We can also see that a smaller learning rate results in smaller incremental changes to the weight values but can help us narrow in on the values that will result in the output equaling the expected output. Furthermore, due to the complexity of our problem of distinguishing legitimate from illegitimate server identification values and our large continuous sample space, we hypothesized that more iterations would be needed to find the desired weight values.

We tested each combination for ten minutes where prior to each experiment, we cleaned the `weights` and `crunches` files to ensure that each experiment was not influenced by previous experiments. Our experiment was limited to ten-minute intervals because we were testing our technique in a live corporate environment and had to ensure that we did not degrade service to our employees. We elected to use the production servers of our corporate environment instead of a test server because the volume of SMTP transactions that we receive on the production servers far surpasses any volume that we could have achieved using our test servers. In other words, the volume on our production servers accurately represented the type of volume we would eventually want to expose our filtering technique to. Our experimentation is shown on Table 7-1.

Table 7-1: Experiment – Determining Perceptron Parameters

Perceptron learning rate	Perceptron max iterations	layer 1a: BAD HELO blockings	layer 1b: BAD RDNS blockings	layer 1b: BAD NXDOMAIN blockings	layer 2: content filtering blockings	layer 3: Spam-Assassin blockings	E-mails blocked
0.8	1,000	66	120	60	10	3	259
0.5	1,000	18	126	37	5	1	188
0.2	1,000	10	172	45	5	1	239
0.8	10,000	131	2	3	3	2	147
0.5	10,000	106	94	32	8	1	238
0.2	10,000	41	126	48	9	2	223
0.8	100,000	80	0	0	38	1	95
0.5	100,000	89	0	0	2	1	92
0.2	100,000	71	0	0	9	0	72

Perceptron learning rate	Perceptron max iterations	E-mails passed	%E-mails blocked by first layer	% E-mails blocked by all three layers	SMTP requests rejected
0.8	1,000	123	95.57%	74.59%	0
0.5	1,000	102	97.29%	71.67%	0
0.2	1,000	111	97.75%	73.70%	0
0.8	10,000	84	97.10%	78.68%	0
0.5	10,000	102	96.62%	79.06%	0
0.2	10,000	133	95.68%	72.29%	0
0.8	100,000	36	83.41%	86.42%	134
0.5	100,000	1	98.84%	99.43%	77
0.2	100,000	5	92.19%	96.24%	94

We note from Table 7-1 that a larger number of iterations resulted in better overall blocking rates. Specifically, a maximum iteration count of 100,000 resulted in an average overall blocking rate of 94.03%, a maximum iteration count of 10,000 resulted in an average overall blocking rate of 76.68% while a maximum iteration count of 1,000 using the same learning rate resulted in an average overall blocking rate of 73.32%.

However, we note that using a high number of maximum iteration counts requires further processing for each SMTP transaction, which means less processing for additional transactions. From our results, we can see that a 100,000 maximum iteration count resulted in an average of 101.7 incoming connections being rejected whereas the 1,000 and 10,000 maximum iteration counts resulted in no incoming connections being rejected.

With respect to the learning rate, we note that a higher learning rate combined with a smaller maximum iteration count resulted in our perceptron module blocking more

connections. Specifically, a learning rate of 0.8 with a maximum iteration count of 1,000 produced 48 more perceptron-based blocks than a learning rate of 0.5 with the same maximum iteration count. Similarity, a learning rate of 0.8 with a maximum iteration count of 10,000 produced 25 more perceptron-based blocks than a learning rate of 0.5 with the same maximum iteration count. We also note that a learning rate of 0.8 with a maximum iteration count of 10,000 was the combination that produced the highest blocking rate of 97.10% by our filtering technique without introducing any failures. We note that in a live production environment, failures are unacceptable and to ensure that all mail is delivered in a timely matter, we elected to use an iteration count that would block the most mail but not reject incoming transactions. In other words, we elected to use a perceptron with a learning rate of 0.8 with a maximum iteration count of 10,000.

7.4 Experimentation Setup

In our testing, we measured the effectiveness of our technique by developing a means to track and record the total number of SMTP transactions blocked at each filtering level , total number of SMTP transactions blocked, total number of SMTP transactions allowed, total number of SMTP transactions mistakenly allowed (i.e. false negatives), and total number of SMTP transactions mistakenly blocked (i.e. false positives) for each filtering server. We describe how we were able to provide such means in the sections below.

7.4.1 Mail filtering server logs

In the previous section, we mentioned the need to track and record various events on each mail-filtering server at each filtering level. To meet this need, we configured the two

mail-filtering servers to generate logs for each event in an SMTP transaction. Specifically, a detailed log event was created for each of the five steps in the five-step SMTP transaction model. For example, the logs of all SMTP rejections and acceptances include detailed information such as:

1. Disposition: Identifies whether the transaction was accepted or blocked.
2. Server ID: Specifies the sending server's identification value.
3. Reason: Identifies the layer that blocked the SMTP transaction.
4. TCP/IP: Lists the IP address of the sending server.
5. Subject: Subject value used in the e-mail content portion of the transaction.
6. From: 'From' value used in the e-mail content portion of the transaction.
7. To: 'To' value used in the e-mail content portion of the transaction.
8. Date: Time stamp, in GMT, that identifies when the transaction was performed.
9. SessionID: Unique identifier to identify this particular transaction.

The format of such a log event is presented below:

```
Disposition:BLOCK
Server ID:  M018.iofferstar.com
Reason:     Layer 1 DNSBL: 66.154.113.12
TCP/IP:     66.154.113.12
Subject:    Save over $100 on Mineral Makeup! 9 piece kit only $49.95!
From:       "Simply Cosmetics"<THree@iofferstar.com>
To:         "random@example.com"<random@example.com>
Date:       2/5/2007 3:21:59 AM
SessionID: 1170673855.04009696
```

We note that a log-parsing tool was created to parse the mail filtering server logs to provide us with the metrics we present in the results section below. In the next two sub-sections, we describe how these logs were used to identify and record false positives and false negatives.

7.4.2 False Positive Process

To determine the number of false positives that result from each layer of filtering, we implemented the in-line rejection means, described in Chapter 4, on both filtering servers to detect and track false positives. Specifically, when our filtering server flags a particular SMTP transaction as spam, the sending server should receive a notification informing them that the transaction was refused along with instructions on how they can report it to the sending domain administrators as a false positive. In our test environment, the instructions included a unique identifier identifying the blocked SMTP transaction along with an unfiltered e-mail address that the report should be sent to. When a user receives such a notification, they then simply forward it to the unfiltered address, which bypasses all the filtering layers on both filtering servers and arrives in the mailbox of the sending domain's mail administrator.

When an administrator receives such a false positive report, he or she takes the unique identifier and looks it up in the mail filtering logs, as described in the previous sub-section, to determine which filtering layer the SMTP transaction was blocked by. The administrator then records the false positive report as well as the filtering layer that it was blocked by and subsequently takes the necessary steps to fix the problem. Fixing includes either re-configuring the filtering layer that caused the block or maintaining

existing configurations. We maintain existing configurations in situations such as when the sender violates corporate policies. Since false positive values are dependent on the sender reporting the error to us, it should be noted that the false positive values produced in our experimentation may not reflect the true number of false positives. For example, some senders may not bother to report the false positive error.

7.4.3 False Negative Process

The false negative collection came from end users who reported spams to us using a software plug-in on the Microsoft Outlook e-mail client. Specifically, whenever a user received spam, he or she would simply highlight the message and click the SPAM button on their Microsoft Outlook client and the entire spam with full headers would then be sent to an automated account, which would extract the headers and find the source IP address that sent the spam to us and the time that the spam was sent. We would then correlate this information with the source IP address and time stamp in the mail filtering logs as described in sub-section 7.2.1 to determine the mail-filtering server that allowed the unsolicited e-mail to get through. We record the total number of spams reported to us by the false negative process to determine how much got through each server. Since false negative values are dependent on the recipient reporting spam to us, it should be noted that the false negative values produced in our experimentation may not reflect the true number of false negatives. Specifically, some recipients may not bother to report the spam to us and simply delete it. In other cases, the recipient may have reported a false negative to us but did not include the header information, which is needed in order to determine which of the two filtering servers let the particular spam through.

7.5 Training

Our filtering technique was trained throughout the entire experimentation process in the above-mentioned environment. We elected to deploy a continuously learning model because we anticipate spammers to counter this filtering technique by modifying the behaviour of their zombies. By continuously learning the server identification values generated by known spam sources, we hypothesize that our filtering technique will be able to withstand most counter attacks.

7.6 Experimentation

In our experimentation, we tested both filtering systems in real-time for a 2 day period using a perceptron with a learning rate of 0.8 and a maximum iteration count of 10,000 where both values were determined via experimentation defined above. We selected 2 days for our experimentation because we hypothesized that our technique would produce better results on the second day than the first day. Specifically, the second day of testing would have the added benefit of a full day of training that the first day did not have.

For each of these days, we present results for the two filtering systems. Specifically, we detail the total number of SMTP transactions, the number of SMTP transactions blocked by each of the filtering levels, the number of SMTP transactions passed, the total number of SMTP transactions blocked, the percentage of SMTP transactions blocked by layer 1a, the percentage of SMTP transactions blocked by layer 1 and the number of false positives and false negatives. We provide results for each filtering server in Tables 7-2 and 7-3 below.

Table 7-2: Results - Day 2 Statistics

Day 2	Total Mail	Layer 1a: BAD HELO	Layer 1b: BAD RDNS	Layer 1b: BAD NXDOMAIN	Layer 1: DNSBL	Layer 2: Content	Layer 3: SpamAssassin
Server 1	40046	29631	1360	736	NA	1721	546
Server 2	67852	NA	NA	NA	22526	2693	1624

Day 2	PASS	BLOCK	% Blocked by Layer 1a	% Blocked by Layer 1	False Positives	False Negatives
Server 1	5712	34334	92.4%	92.4%	0	0
Server 2	42633	25219	NA	89.3%	9	0

Table 7-3: Results - Day 1 Statistics

Day1	Total Mail	Layer 1a: BAD HELO	Layer 1b: BAD RDNS	Layer 1b: BAD NXDOMAIN	Layer 1: DNSBL	Layer 2: Content	Layer 3: SpamAssassin
Server 1	48866	22088	5869	2046	NA	1844	498
Server 2	35433	NA	NA	NA	20610	1755	1070

Day1	PASS	BLOCK	% Blocked by Layer 1a	% Blocked by Layer 1	False Positives	False Negatives
Server 1	16521	32345	68.3%	92.8%	0	0
Server 2	13068	22365	NA	92.2%	0	0

7.7 Analysis

The results from Tables 7-2 and 7-3 validate our hypothesis stated in Section 7.6 in that the second day resulted in a slightly higher blocking rate by our perceptron, identified as layer 1a. The key columns in Tables 7-2 and 7-3 are the `%Blocked by Layer 1a` and

%Blocked by Layer 1. The former column is calculated using Equation 7-1 and only applies to the first server since the second sever does not use our filtering technique.

$$\%\,Blockbylay\ er1a = \frac{Layer\,1a\ :BAD\,HELO}{BLOCK}$$

Equation 7-1: Blocked by Layer 1a

Specifically, the %Blocked by Layer 1a value is calculated by dividing the Layer 1a BAD HELO column value by the total BLOCK column value.

The latter column, %Blocked by Layer 1, is calculated by dividing all blocks performed by layer 1 by the total number of blocks. Specifically, for server 1, we used Equation 7-2 by dividing the sum of layer 1a BAD HELO, Layer 1b: BAD RDNS and Layer 1b: BAD NSDOMAIN by the column BLOCK.

$$\%\,Blockbylay\ er1 = \frac{Layer\,1a + Layer\,1b : BAD\,RDNS + Layer\,1b : BAD\,NXSDOMAIN}{BLOCK}$$

Equation 7-2: Blocked by Layer 1 (Server 1)

Whereas with server 2, we used Equation 7-3 by dividing the Layer 1: DNSBL column value by the total BLOCKED column.

$$\%\,Blockbylay\ er1 = \frac{Layer\,1\ :DNSBL}{BLOCK}$$

Equation 7-3: Blocked by Layer 1 (Server 2)

We note the Layer 1:DNSBL column does not apply to server 1 because it does not use the DNSBL as its first layer of filtering.

Despite the limitations of the perceptron, as mentioned in Section 6.3.1.1, our perceptron-based filtering technique performed very well. Specifically, as shown in

tables 7-2 and 7-3, our perceptron blocked 68.3% of all blocked mail on the first day and blocked 92.4% of all blocked mail on the second day. We also note that on the second day, the entire first layer of filtering on the first server produced better overall blocking results than the first layer of filtering on the second server. Specifically, our filtering technique combined for a total blocking percentage of 92.4% whereas the DNSBL on the second server accounted for 89.3% of all blocking. We note that the large input domain combined with the stopping criteria made both seperability and performance a non-issue. Furthermore, we hypothesize that our technique was linearly separate because the combination of domain terms used by spam zombies is clearly distinguishable from the combination of domain terms used by illegitimate sources. Support for our hypothesis is found in our analysis in Section 5.4.2 along with our results from Section 7.6. We note as future work in Section 8.3, that we wish to map our results on a four dimensional plane to prove our theory correct.

We note that there was a large discrepancy in the number of e-mails that were passed through the first and second servers on the first day. This simply indicates that more legitimate mail was being sent to the second server than the first server on that day. To ensure that the PASS values did not affect our analysis, we did not include these values in any of our calculations.

7.7.1 Generalization of Spamhaus list

As was mentioned in section 6.5.1.2, our perceptron was trained using the Spamhaus list [6] that was specifically designed to list spam zombies. Specifically, if an incoming mail server was listed on the Spamhaus list, our perceptron would learn the identification of

said server. In theory, our perceptron should be able to detect spam zombies that are listed in the blacklist as well as zombies that are not already listed on the blacklist. In Section 3.4.2, we discussed how the Spamhaus blacklist suffers from a fairly substantial lag time with respect to listing newly created spam zombies. Therefore, introducing a new means by which one can reduce or potentially eliminate such lag time would help overcome one of the main deficiencies behind the usage of DNS blacklists. In order to determine whether our perceptron can detect spam zombies that the Spamhaus list cannot, we picked five consecutive SMTP connection blocks performed by our perceptron and for each block, we immediately queried the Spamhaus list to determine if said servers were listed or not. We note that since the transaction was rejected after Step 5 of our five-step SMTP model, we were also able to review the contents of these five transactions and thereby confirm that each of them were unsolicited mailings of one the three types described in Section 4.3. In the data shown in Table 7-4, we present the five blocks as they appeared on the mail filtering logs of server 1 and for each block, we also indicate whether the IP address is listed in the Spamhaus list.

Table 7-4: Results - Generalization Validation

Test No.	TimeStamp	Sending Server IP address	Spamhaus Listed	SPAM
1	2007.03.12 17:01:38	122.168.69.80	No	Yes
2	2007.03.12 17:01:41	217.227.127.253	No	Yes
3	2007.03.12 17:01:42	24.207.15.100	Yes	Yes
4	2007.03.12 17:01:44	85.207.18.230	Yes	Yes
5	2007.03.12 17:01:46	68.217.127.63	No	Yes

Test No.	Server Identification (HELO) Value
1	`revueaveri.kylin-kited5bunns.great-diabetic-foot.net`
2	`pD9E37FFD.dip.t-dialin.net`

3	h24-207-15-100.cst.dccnet.com
4	quhuldb
5	adsl-217-127-63.asm.bellsouth.net

Test No.	Reverse DNS Value
1	dsl-MP-dynamic-080.69.168.122.airtelbroadband.in
2	pD9E37FFD.dip.t-dialin.net
3	h24-207-15-100.cst.dccnet.com
4	230-18.207.85.bluetone.cz
5	adsl-217-127-63.asm.bellsouth.net

Table 7-4 illustrates that although our technique relies on the Spamhaus list to learn bad mail server behaviour, that our technique is able to generalize said list and detect 3 servers that were not already listed in the Spamhaus list. We note that each Spamhaus list query was performed immediately after our perceptron blocked the SMTP connection to ensure accuracy of our data. Specifically, a lag between the block and Spamhaus list query could result in erroneous results since the Spamhaus list is likely to be regularly updated with new IP addresses of newly detected spam zombies [44].

We also note that since our filtering technique used the Spamhaus list to determine the expected outputs of our perceptron training, we expect that our filtering technique's ability to generalize the Spamhaus list will improve with each passing day. As such, in the Spamhaus list lookup Step in Figure 6-3, we added a logging mechanism to indicate when a Spamhaus list lookup has been performed and after each day of filtering, we parsed the logs to determine how many times the Spamhaus list needed to be queried. As illustrated in Figure 6-3, the Spamhaus list is not queried if our filtering layer blocked the incoming SMTP transaction.

We expect that in the beginning, the number of Spamhaus list queries would be high because our perceptron is just starting to learn. However, with each passing day, the number of queries should reduce as it continues to learn spam zombie behaviour and become better at detecting such sources. This ability to learn on-line and in real-time provides an additional advantage in that our filtering technique is capable of adapting to changes in spam zombie behaviour.

The number of Spamhaus list hits from our two days of testing is shown in Table 7-5 below.

Table 7-5: Results - Perceptron and Spamhaus List Comparison

Experimentation Day	Number of Spamhaus list hits
1	5263
2	2775

As expected, the first day resulted in a higher number of Spamhaus list hits compared to the second day, where on the second day, the number of Spamhaus list hits dropped by 2488. We note that on second day, our first server generated a smaller number of Spamhaus queries despite an increase in total volume of mail in comparison to the first day. We hypothesize that with each passing day, we would continue to observe a decrease in Spamhaus list hits.

We also did not receive a single false positive or false negative report for any SMTP transactions handled by the first filtering server. We did however receive nine false positive reports for the second filtering server on the first day, which was likely due to the higher volume of mail that it handled that day. As we mentioned in Section 7.4.2 and

7.4.3, we cannot accurately measure false positive and false negative values using our preferred method of Section 4.4.1 and as a result, we do not place much emphasis on our false positive or negative reports.

Finally, we note that spammers cannot easily overcome our technique. First, our filtering technique performs continuous learning against the Spamhaus list such that our technique will be able to adjust itself to detect changes as spammers adjust their zombies. This will hold true as long as the Spamhaus list is continuously updated with newly detected spam zombie sources, which is currently the case [44]. Second, our technique also prevents spammers from making simple adjustments such as forging legitimate host names like `mail.google.com` as their server identification value. Specifically, the reverse DNS module of our filtering technique will ensure that the sending server abides by Step 2 of our five step SMTP transactional model. In other words, if a spammer wanted to use a server identification value of `mail.google.com`, they would likely be able to pass through our perceptron but would not get past our reverse DNS module unless they are able to change the reverse DNS entry of the compromised machine to `mail.google.com`. Since the reverse DNS of most potential zombie machines (i.e. home machines) are provided by the service provider of the machine's IP address [67], an attacker cannot easily change the reverse DNS entry of the server to overcome our technique. In the case where no reverse DNS entry exists, the attacker will need to check that the machine has no reverse DNS and change their server identification values to an address literal containing the IP address of the compromised machine. This change is feasible but requires a non-trivial amount of effort on the spammer's part to find

machines without reverse DSN entries and create malicious software that will identify themselves as an address literal of the machine's IP address. One large benefit that our technique provides over existing technique is that our technique is capable of blocking mail from newly formed spam zombies whereas other techniques such as DNS blacklists will only block mail from a zombie machine only if there was the machine had been caught sending spam.

7.8 Summary

In this Chapter, we described the experiments that were used to determine preferred learning rate and maximum iteration count parameters for our perceptron. We used these parameters when comparing the effectiveness of two three-layer filtering servers where the first layer of the first server comprised of our filtering technique while the first layer of the second filtering server comprised of a DNS-blacklist. Our results showed that the blocking rate of our filtering technique improves significantly after a full day of training. Furthermore, in comparison to a second filtering server using the Spamhaus blocking list as its first filtering layer, we note that our perceptron produced higher blocking rates than the Spamhaus list. Additionally, we also note that our entire filtering technique (i.e. layers 1a and 1b) produced slightly better blocking rates in comparison to the Spamhaus list.

We note that our technique is designed such that a spammer cannot easily overcome our technique. First, our technique performs continuous learning against a third party spam zombie blacklist that is regularly updated. Second, an attacker would need to change the reverse DNS entry of the compromised machine, typically stored on a

separate server, in order to bypass our technique. Alternatively, they would need to know that the compromised machine has no reverse DNS value and thus change its identification value to an address literal of its IP address.

Chapter 8: Conclusion

8.1 Conclusions

In this thesis, we presented a new technique for filtering spam that cannot be easily overcome by spammers. Our technique consisted of two main parts. The first part comprised a single perceptron that was designed to learn and distinguish legitimate and illegitimate sending server parameter values. The second part was used to ensure that the sending server parameter value was consistent with Step 2 of our five step SMTP transactional model described in Chapter 4. In order to determine the effectiveness of our filtering technique, we integrated it as a first layer of an existing multi-layer corporate spam filtering server and compared the integrated server with a non-integrated spam filtering server. Each filter consisted of three filtering layers where each layer was performed sequentially. Specifically, the first layer had precedence over the second layer, which had precedence over the third layer.

In the integrated server, the first layer of filtering was our filtering technique followed by corporate content filtering rules as a second layer and a generic version of SpamAssassin (Apache) as the third layer. In the non-integrated server, the first layer of filtering was the Spamhaus blacklist. The second and third layers of the non-integrated and integrated servers were the same.

Once our filtering technique was integrated with the existing filtering solution, we ran a set of nine tests to determine a preferred learning rate and stopping condition parameter values for our perceptron. In Chapter 7, we presented the results from this experiment and discovered that a learning rate of 0.8 and a maximum iteration count of 10 000 provided the highest blocking percentage for our technique without resulting in any major performance problem which would prevent incoming connections from being rejected.

Next, we compared the filtering effectiveness of the two filtering servers over a two day period. In particular, we were interested in the percentage of SMTP connections blocked by layers 1a (i.e. perceptron) and 1 (i.e. perceptron and reverse DNS) of our first server and layer 1 (i.e. Spamhaus list) of our seconrd server. The first observation we noted was that our hypothesis was correct where the perceptron performed significantly better on the second day than on the first day since on the second day, the perceptron had a full day of training whereas it had no training on the first day. The second observation was that our filtering layer (i.e. layer 1 of the first server) accounted for a higher percentage of total blocked SMTP transactions in comparison to the Spamhaus list. We also showed how our perceptron was able to generalize the Spamhaus list by selecting five consecutive block reports in the logs of our first server at a random time. In particular, we noticed that our perceptron was not only able to detect sending servers that were listed in the Spamhaus list but could also detect spam zombies that had yet to be listed on the Spamhaus list.

8.2 Future Research

As future research, we would like to further investigate the following four items:

- We would like to make use of the transaction flow and e-mail content deviations described in Chapter 6 and determine whether adding these features to our technique would result in better zombie detection.
- We would be interested in mapping our results in a four-dimensional plane to confirm the combination of domain terms used by spam zombies are clearly distinguishable from the combination of domain terms used by legitimate mail sources and are hence linearly separable as our results indicate.
- We would also like to observe our filtering technique over a longer period of time. In Chapter 7, we presented results of our mail filtering technique in a two day span where in the second day, less Spamhaus list queries were needed to train the perceptron. We would be interested in determining the number of days that would be required such that our perceptron would need less than 10 Spamhaus list queries. We suggest 10 because we account for the situation where spammers are likely to eventually make substantial improvements to their spam zombies.
- We would be interested in determining how well our technique would work if the index value associated with each domain term represented the frequency of said term used in all server identification values inspected by the perceptron.
- We would also like to conduct further performance tests on our filtering technique to determine the preferred activation function, learning rate and maximum iteration count required for our technique to achieve higher blocking statistics.

Specifically, we would be interested in performing the same experimentation as described in Section 7.3 but over a longer period of time.

References

[1] MessageLabs, "Threat Statistics," December 2006, http://www.messagelabs.com/Threat_Watch/Threat_Statistics/.

[2] M. Shiels, "Why one spam could cost $50," BBC News, April 2002. http://news.bbc.co.uk/1/hi/sci/tech/1917458.stm.

[3] Coalition Against Unsolicited Bulk Email, Australia, "Microeconomics of Spam and Direct Marketing," December 2006, http://www.caube.org.au/microec.htm.

[4] R. Hunt and J. Carpinter, "Current and New Developments in Spam Filtering," in Proceedings of 14th IEEE International Conference on Networks, vol. 2, pp. 1-6, 2006.

[5] Ironport, "Spammers Continue Innovation: IronPort Study Shows Image-based Spam, Hit & Run, and Increased Volumes Latest Threat to Your Inbox," 2006, http://www.ironport.com/company/ironport_pr_2006-06-28.html.

[6] Spamhaus. "The Spamhaus Project - XBL," 2006, http://www.Spamhaus.org/xbl/.

[7] S. Hershkop, and S. J. Stolfo, "Identifying spam without peeking at the contents," ACM Crossroads, vol. 11, pp. 3, 2005.

[8] G. Manaco, E. Masciari, M. Ruffolo, and A. Tagarelli, "Towards an adaptive mail classifier," Institute of Italian National Research Council, Rende, Italy, Tech. Rep. 2002, http://www-dii.ing.unisi.it/aiia2002/paper/APAUT/manco-aiia02.pdf.

[9] L. Cranor, and B. LaMacchia, "Spam!" *Communications of the ACM*,

vol. 41 no. 8 pp. 74–83, 1998.

[10] Mertz, D, "Spam filtering techniques, Six approaches to eliminating unwanted e-mail,". Technical report, Gnosis Software, Inc., Tech. Rep., 2002, http://www-128.ibm.com/developerworks/linux/library/l-spamf.html.

[11] M. Mangalindan, "For bulk e-mailer, pestering millions offers path to profit," *Wall Street Journal*, 2002.

[12] B. Sullivan, "Who profits from spam? Surprise," MSNBC, 2003 http://www.msnbc.msn.com/id/3078642/.

[13] F. Provost, and T. Fawcett, Robust classification for imprecise environments. *Machine Learning Journal*, vol 42, pp. 203–231, 2001.

[14] J. St. Sauver, "Spam Zombies and Inbound Flows to Compromised Customer Systems,". *MAAWG General Meeting*, 2005, http://www.uoregon.edu/~joe/zombies.pdf.

[15] R. Conner, "Rick's spam digest," July 2006, http://www.rickconner.net/spamweb/pop-find-mail-host.html.

[16] J. Levine, M. Levine-Young, and R. Everett-Church, "Fighting Spam for Dummies," For Dummies, January 2004.

[17] D. Schlesinger, "Human Perception, Computer Vision, and CAPTCHA," July 2004, http://www.imakenews.com/eletra/mod_print_view.cfm?this_id=279051&u=refinery&issue_id=000056359&show=F,T,T,T,F,Article,F,F,F,F,T,T,F,F,T,T.

[18] Anti Phishing Working Group, "Anti-Phishing Working Group," 2007, http://www.antiphishing.org/.

[19] Malware – Wikipedia, the free encyclopedia, "Malware," 2007, http://en.wikipedia.org/wiki/Malware.

[20] Secure Computing – Anti Spam and Email Security Solutions For Enterprises, "CipherTrust Confirms 21-Percent Jump In New Zombie Machines And 20-Percent Overall Increase In Spam," June2006, http://www.ciphertrust.com/company/press_and_events/article.php?id=0000892.

[21] Honeynet Project and Research Alliance, "Know your enemy: Tracking Botnets," March 2005, http://www.honeynet.org/papers/bots/.

[22] A. Weiss, "Ending spam's free ride," netWorker, vol. 7 pp. 18–24, 2003.

[23] U.S. Senate and House of Representatives, Controlling the assault of non-solicited pornography and marketing act of 2003. S. 877.

[24] D. Fallows, "Can-spam a year later", Pew Data Memo, April 2005, http://www.pewinternet.org/pdfs/PIP_Spam_Ap05.pdf.

[25] Y. Lee. "The can-spam act: a silver bullet solution?," COMMUNICATIONS OF THE ACM, vol. 48, No. 6, pp. 131–132, June 2005.

[26] Barracuda Networks, "CAN-SPAM Act Continues to Come Up Short in Efforts to Curb Unsolicited Email," December 2006.

[27] D. Boneh. "The Difficulties of Tracing Spam Email," *Report to Congress on A CAN-SPAM Informant Reward System*, 2004, http://www.ftc.gov/reports/rewardsys/expertrpt_boneh.pdf.

[28] S. Gorling, "The myth of user education," in *Proceedings of the 16th Virus Bulletin International Conference*, 2006, pp. 11-13.

[29] J. Klensin. "RFC2821 - Simple Mail Transfer Protocol," IETF (Standards Track) Request for Comments, April 2001.

[30] Messaging Anti-Abuse Working Group, "Managing Port 25 for Residential or Dynamic IP Space Benefits of Adoption and Risks of Inaction," 2005, http://www.maawg.org/port25/MAAWG_Port25rec0511.pdf.

[31] D. Whyte, E. Kranakis, and P. C. van Oorschot, "DNS-based Detection of Scanning Worms in an Enterprise Network," in *Proceedings of the 12th Network and Distributed System Security Symposium*, 2005, pp. 181-195.

[32] S. Ahmed and F. Mithun, "Word stemming to enhance spam filtering," in *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004, http://www.ceas.cc/papers-2004/167.pdf.

[33] E. Crawford, J. Kay, and E. McCreath, "Automatic induction of rules for e-mail classification," in *Sixth Australian Document Computing Symposium, Coffs Harbour*, Australia, 2001.

[34] SpamAssassin. "The Apache SpamAssassin Project," 2006, http://spamassassin.apache.org/.

[35] H. Stern. "Fast SpamAssassin Score Learning Tool,", January 2004, http://search.cpan.org/src/PARKER/Mail-SpamAssassin-3.0.3/masses/README.perceptron.

[36] T. Oda and T. White, "Developing an Immunity to Spam," in *Genetic and Evolutionary Computation Conference*, 2003, pp. 231-242.

[37] A. Gray and M. Haahr, "Personalised, collaborative spam filtering," in

Proceedings of the First Conference on Email and Anti-Spam (CEAS), 2004, http://www.ceas.cc/papers-2004/132.pdf.

[38] V. Prakash, "Vipul's razor,"2005, http://razor.sourceforge.net/.

[39] Rhyolite Software, "Distributed Checksum Clearinghouse", 2001, http://www.rhyolite.com/anti-spam/dcc/.

[40] Composite Blocking List, "The CBL". 2004, http://cbl.abuseat.org/.

[41] NJABL – Not Just Another Blocking List, 2006, http://www.njabl.org/.

[42] A. Ramachandran D. Dagon, and N. Feamster, "Can DNS-Based Blacklists Keep Up with Bots?," in *Proceedings of the Third Conference on Email and Anti-Spam CEAS,* 2006, http://www.ceas.cc/2006/14.pdf.

[43] S. Haykin. 2nd Ed., *Neural Networks. A Comprehensive Foundation.* New Jersey: Prentice-Hall Inc., 1999.

[44] Spamhaus. "The Spamhaus Project," 2006, http://www.Spamhaus.org/.

[45] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 291-302, 2006.

[46] J. Jung and E. Sit, "An empirical study of spam traffic and the use of dns black lists," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004, pp. 370–375.

[47] M. Wong and W. Schlitt, "RFC4408 - Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1," IETF (Standards Track) Request for Comments, April 2006.

[48] J. Chan, "SURBL – Spam URI Realtime Blocklists," 2007, http://www.surbl.org/.

[49] M. Delany, "Domain-based Email Authentication Using Public-Key Advertised in the DNS (DomainKeys)," Internet Draft, 2006, http://www.ietf.org/internet-drafts/draft-delany-domainkeys-base-06.txt.

[50] S. Hershkop, "Behavior-based Email Analysis with Application to Spam Detection," Ph.D. dissertation, Columbia University, New York, NY, U.S.A, 2006 http://www.cs.columbia.edu/~sh553/publications/final-thesis.pdf.

[51] D. D. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," in *Proceedings of 10th European Conference on Machine Learning*, 1998, pp. 4-15.

[52] K. Schneider, "Learning to filter junk e-mail from positive and unlabeled examples," in *Proceedings of the 1st International Joint Conference on Natural Language Processing*, 2004, pp. 602-607.

[53] J. B. Postel. "RFC 821 - Simple mail transfer protocol," IETF (Standards Track) Request for Comments, August 1982.

[54] D. H. Crocker. "RFC822 - Standard for the format of arpa Internet text messages," IETF (Standards Track) Request for Comments, August 1982.

[55] J. Klensin. "RFC2821 - Simple mail transfer protocol," IETF (Standards Track) Request for Comments, April 2001.

[56] E. Levy, "The Making of a Spam Zombie Army: Dissecting the Sobig Worms," *IEEE Security and Privacy*, vol. 1, pp. 58-59, July 2003.

[57] Ethereal: A Network Protocol Analyzer, "Ethereal," 2007, http://www.ethereal.com/.

[58] Y. Guo, Y. Zhang, J. Liu, and C. Wang, "Research on the Comprehensive Anti-Spam Filter," Industrial Informatics, pp. 1069-1074, 2006.

[59] Messaging and Anti-Abuse Working Group, "Messaging and Anti-Abuse Working Group," 2007, http://www.maawg.org/home.

[60] H. Aradhye, G. Myers, and J. Herson, "Image analysis for efficient categorization of image-based spam e-mail," in *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, 2005, pp. 914-918.

[61] R. Duda and P. Hart, *Pattern classification and scene analysis*. New York: John Wiley and Sons, 1973.

[62] S. Russell, and P. Norvig, 2nd Ed., "Artificial Intelligence: A Modern Approach," New Jersey: Prentice Hall, pp. 738-743, 2003.

[63] Z. Uykan, and H. N. Koivo, "Unsupervised learning of sigmoid perceptron," in *Proceedings of 2000 International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, pp. 3486-3489, 2000.

[64] Domain name – Wikipedia, the free encyclopedia, "Domain Name," 2007, http://en.wikipedia.org/wiki/Domain_name#Other-level_domains.

[65] S. Purkis, "AI-Perceptron-1.0," 2003, http://search.cpan.org/dist/AI-Perceptron/. http://www.barracudanetworks.com/ns/news_and_events/index.php?nid=124.

[66] O. Kolkman, "Net::DNS – Perl interface to the DNS resolver – search.cpan.org," http://search.cpan.org/~olaf/Net-DNS-0.59/lib/Net/DNS.pm.

[67] Reverse DNS lookup - Wikipedia, the free encyclopedia, "Reverse DNS lookup," 2007, http://en.wikipedia.org/wiki/Reverse_DNS_lookup.

Appendix A

Nigerian 419 E-mail Scam Spam Example

Lagos, Nigeria.
Attention: The President/CEO

Dear Sir,

Confidential Business Proposal

Having consulted with my colleagues and based on the information gathered from the Nigerian Chambers Of Commerce And Industry, I have the privilege to request your assistance to transfer the sum of $47,500,000.00 (forty seven million, five hundred thousand United States dollars) into your accounts. The above sum resulted from an over-invoiced contract, executed, commissioned and paid for about five years (5) ago by a foreign contractor. This action was however intentional and since then the fund has been in a suspense account at The Central Bank Of Nigeria Apex Bank.

We are now ready to transfer the fund overseas and that is where you come in. It is important to inform you that as civil servants, we are forbidden to operate a foreign account; that is why we require your assistance. The total sum will be shared as follows: 70% for us, 25% for you and 5% for local and international expenses incidental to the transfer.

The transfer is risk free on both sides. I am an accountant with the Nigerian National Petroleum Corporation (NNPC). If you find this proposal acceptable, we shall require the following documents:

(a) your banker's name, telephone, account and fax numbers.

(b) your private telephone and fax numbers – for confidentiality and easy communication.

(c) your letter-headed paper stamped and signed.

Alternatively we will furnish you with the text of what to type into your letter-headed paper, along with a breakdown explaining,

comprehensively what we require of you. The business will take us thirty (30) working days to accomplish.

Please reply urgently.

Best regards

Howgul Abul Arhu

Appendix B

Sample Weights File

```
weight0=5.71730566686504
weight1=-1.08493498508378
weight2=1.31181955122055
weight3=-0.11022471848593
```

Sample Crunches File

```
0=agenda
1=si
2=in
3=247
4=36
5=46
6=58
7=45
8=164
9=83
10=clearthought
11=ca
12=diddlina
13=tmfweb
14=nl
15=203
16=117
17=231
18=159
19=ANantes-156-1-94-85
20=w90-12
21=abo
22=wanadoo
23=188
24=253
25=BULLA
26=76-39-136-85
27=user
28=auna
29=net
```

```
30=servidorpro4
31=81-8-153-147
32=pp
33=it4u
34=mail
35=bigdeal
36=com
37=alotfy-mail
38=host34-48-static
39=56-217-b
40=business
41=telecomitalia
42=cmc205
43=neoplus
44=adsl
45=tpnet
46=p508C30F8
47=dip0
48=t-ipconnect
49=89-138-151-214
50=bb
51=netvision
52=yiryg
53=DSL217-132-166-119
54=82-45-76-14
55=cable
56=ubr03
57=basi
58=243
59=107
60=242
61=laurynas-8b2ba4
62=elekta
63=lt
64=fl-showcase
65=pD951664B
66=222
67=248
68=148
69=97
70=69
71=comppost
72=80
73=178
74=167
75=186
76=offb-590ebdb4
77=pool
78=einsundeins
79=sdgate2
80=igk
81=scri
82=hr
```

```
83=bcwebheads
84=my
85=dyn-83-157-57-59
86=ppp
87=tiscali
88=fr
89=smtp
90=ccpteam
91=eu85-87-54-229
92=clientes
93=euskaltel
94=USER-0EC9C181BC
95=nc-71-54-56-176
96=dhcp
97=embarqhsd
98=aiy212
99=internetdsl
100=cable-99-51
101=zeelandnet
102=pool-72-85-169-77
103=bstnma
104=east
105=verizon
106=segment-124-7
107=sify
108=marie-7661eb2b1
109=202
110=37
111=184
112=red-217-217-186
113=lbttech
114=91
115=201
116=applehyip
117=localhost
118=dynamic8121395123
119=hsd1
120=STEALTH-4729532
121=comcast
122=goldeagledesigns
123=freee
124=ru
125=host38-124
126=pool8252
127=interbusiness
128=DM
129=ppg-ppgpower
130=yesppg
131=cn
132=astroprise
133=5ac133e5
134=sky
135=85-220-90-149
```

```
136=simnet
137=hunnycomputer
138=162
139=155
140=87-126-100-157
141=btc-net
142=bg
143=221
144=115
145=92
146=194
147=168
148=mx240mi
149=trafficbucks
150=d83-181-125-78
151=cust
152=tele2
153=pD9EE0D80
154=t-dialin
155=0x50c62bf3
156=arcnxx12
157=adsl-dhcp
158=nortelprobetester
159=geomail
```