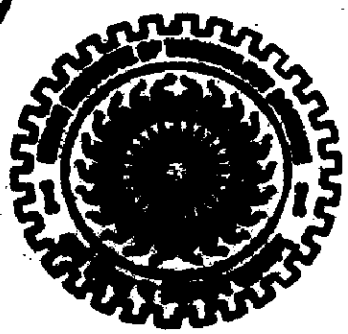
MINIMIZATION OF DHA (DIRECTORY HARVEST ATTACK) AND LOAD OF MAIL SERVER

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree
of
MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

By

SUMAN DAS



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667 (INDIA)

JUNE, 2008

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled "*Minimization of DHA(Directory Harvest Attack) and load of mail server*" towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee (India) is an authentic record of my own work carried out during the period from June 2007 to June 2008, under the guidance of **Dr. R. C. Joshi, Professor and Dr. Durga Toshniwal, Asst. Professor, Department of Electronics and Computer Engineering, IIT Roorkee.**

I have not submitted the matter embodied in this dissertation for the award of any other degree or diploma.

Date: 8-06-08
Place: Roorkee

Suman Das
(SUMAN DAS)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 9.6.08
Place: Roorkee

Durga Toshniwal. 9/6/08.
(Dr. DURGA TOSHNIWAL)
Asst. Professor

(Dr. R. C. JOSHI)
Professor
Department of Electronics and Computer Engineering
IIT Roorkee – 247 667

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my heartfelt gratitude to my guide and mentor **Dr. R. C. Joshi**, Professor and **Dr. Durga Toshniwal**, Asst. Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for their trust in my work, their esteemed guidance, regular source of encouragement and assistance throughout this dissertation work. I would state that the dissertation work would not have been in the present shape without their inspirational support and I consider myself fortunate to have done my dissertation under them.

I also extend my sincere thanks to **Dr. D. K. Mehra**, Professor, and Head of the Department of Electronics and Computer Engineering, and Mr. Raju, lab assistant in Information Security Lab for providing facilities for the work.

I also wish to thank Mr. Rajeev Singh, Ms Anjali Sardana and Mr. T P Sharma for their valuable suggestions and timely help.

Finally, I would like to say that I am indebted to my parents for everything that they have done for me. All of this would have been impossible without their constant support.

SUMAN DAS

Abstract

Directory Harvest Attackers (DHA) attack the mail server to get the valid email addresses and sell these addresses to the spammer(s). The attacker's not only collect the valid user addresses but also slow down the server. The attackers send many blank mails to the mail server to get the valid user address which exists in that domain. Spammers buy the user email addresses and send the SPAM to these addresses. There are many techniques to detect and filter the SPAM, but few techniques are there to reduce DHA. Some protection techniques of mail server are there against centralized DHA, but they fail to protect distributed DHA.

In this work "Minimization *of DHA (Directory Harvest Attack) and Load of Mail Server*", a distributed framework has been proposed, which minimizes the effect of DHA and distributes the load of SMTP server. Blocking criteria to protect distributed attack is totally novel in the proposed framework.

The framework consists of following module 1) front-end-filter, which comprises two databases 2) Reply generator and 3) Distributed servers, which also comprise two databases. The front-end-filter checks whether the source is in black list or not, if the source is not in the blacklist then sends ping to every SMTP server. It then forwards the mail to the SMTP server that responds first, thereby distributing the load. Each SMTP server has its own database. All the distributed SMTP servers store the email addresses and IP addresses to their own databases and in that corresponding entry store the number of mails coming from respective source. All the updates are shared between the distributed SMTP servers. If the count of number of mails is beyond the threshold, then corresponding source is blacklisted and this information is sent back to the front-end-filter. Front-end-filter checks the source address. If it is already black listed then send a packet to the reply generator along with source address. The reply generator generates 'invalid recipient' reply and send it back to the source. Use of Front-end filter minimizes the effect of DHA and load on SMTP server. The effectiveness of the approach is validated with simulation in NS-2 on a Linux platform.

CONTENTS

CANDIDATE'S DECLARATION i

ACKNOWLEDGEMENTS ii

ABSTRACT iii

TABLE OF CONTENTS iv

CHAPTER 1: INTRODUCTION AND STATEMENT OF THE PROBLEM 1

1.1 Introduction 1

1.2 Statement of the Problem 2

1.3 Organization of the Dissertation 2

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW 4

2.1 SPAM Detection 4

2.1.1 Distributed Method 5

2.1.2 Blacklist Method 8

2.1.3 Bayesian Approach 11

2.2 Load balancing Technique 12

2.3 Related Work and Research Gaps 14

2.3.1 Attack Method 14

2.3.2 Protection Method 14

2.3.3 Centralized RBL Method 16

2.3.4 Research Gaps 19

CHAPTER 3: REDUCING DIRECTORY HARVEST ATTACKS 21

3.1 Framework for reducing DHA 21

3.2 Response Method 22

3.3 Maintaining Database 24

3.4 Blocking Criteria 25

CHAPTER 4: MINIMIZING LOAD OF SMTP SERVER **31**
4.1 Distributed Method 31
4.2 Introducing front-end filter 33

CHAPTER 5: SYSTEM DESIGN AND IMPLEMENTATION **36**
5.1 System Design 36
5.1.1 System Components 36
5.1.2 Simulation Model 37
5.2 Implementation 38
5.2.1 Simulation Topology 38
5.2.2 Procedures 39
5.2.3 Simulation Parameters 40

CHAPTER 6: RESULTS AND DISCUSSION **41**
6.1 Effects of DHA 41
6.2 Effects of Load of SMTP Server 47

CHAPTER 7: CONCLUSIONS AND FUTURE WORK **50**
7.1 Conclusions 50
7.2 Suggestions for Future Work 51

REFERENCES **52**
APPENDIX: SOURCE CODE LISTING ..... **i**

CHAPTER 1
INTRODUCTION AND STATEMENT OF THE PROBLEM

1.1 Introduction

Now a day's spammers are increasing rapidly. They are doing their advertisement free of cost by sending SPAM. SPAM causes a big problem for users who use mail to communicate with each other. The spammers get the valid mail-id from Directory Harvest Attackers. The mail server should be protected from Directory Harvest Attack (DHA) to minimize SPAM. The attackers attack the mail server by sending blank message or simple with "hello" statement to randomly generated mail addresses. Attackers send the blank massage to a huge number of randomly generated mail addresses (in a particular domain) in a short time from normal user's machine by making this machine as a zombie[1] or by using open relay [2]. Attackers sell these valid email addresses to the spammers. SMTP server also becomes slow for processing the request from attacker.

Directory harvest attacks mainly use either of two methods for harvesting valid e-mail addresses. The first method uses a brute force approach to send a message to all possible alphanumeric combinations that could be used for the username part of an e-mail. These attacks are more effective for finding e-mail addresses of companies since they are likely to have a standard format for official e-mail aliases (i.e. jdoe@example.domain, johnd@example.domain, or johndoe@example.domain). The second and more selective method involves sending a message to the most likely usernames - for example, for all possible combinations of first initials followed by common surnames. In either case, the e-mail server generally returns a "Not found" reply message for all messages sent to a nonexistent address, but does not return a message for those sent to valid addresses. The DHA program creates a database of all the e-mail addresses at the server that were not returned during the attack.

The result of the DHA is not just more spam (as if that were not bad enough). An aggressive DHA can place such intense demands on a server that it mimics a denial-of-service attack and slows legitimate e-mail delivery.

The DHA approach explains how a new e-mail address can start receiving spam within days or hours after its creation. Various solutions have been developed toward repelling these attacks. Some of the most effective involve slowing down the rate at which the attack can take place, rather than attempting to filter out the entire attack. This can be done by limiting the number of e-mail messages per minute or per hour at which a server can receive messages, legitimate or otherwise. So-called spam filters, programmed to identify character and word combinations typical of spam, can also be effective, although they occasionally reject legitimate messages.

1.2 Statement of the Problem

The main objective of the thesis is to propose an efficient defense mechanisms against DHA also to minimize the load of server. This main problem is subdivided into two sub problems.

1. To analyze the various types of Directory Harvest Attacks (DHA) and to propose an effective and efficient defense mechanism against them.

2. Minimize the load of SMTP server against DHA

1.3 Organization of the Dissertation

This report comprises of seven chapters including this chapter that introduces the topic and statements of the problem. The rest of the dissertation report is organized as follows.

Chapter 2 gives different SPAM detection technique and overview of the Directory Harvest Attack (DHA). In addition, taxonomy of the existing defense mechanism is discussed in brief. It describes the method to minimize the load of SMTP server. This chapter discusses the research gaps in the phases of the defense against DHA attacks.

Chapter 3 gives an overview of the proposed framework. It gives a big picture of the solution to protect the mail server from DHA and how to block the source address and how to maintain the database.

Chapter 4 explains in detail how to minimize the load of SMTP server using distributed approach and using front-end-filter.

Chapter 5 describes the system design that includes the system components and the simulation model. The implementation details are also charted out in terms of the topology used for simulation purposes, procedures, and simulation parameters.

Chapter 6 discusses the simulation results and displays the effectiveness of the proposed mechanism for defense against DHA.

Chapter 7 concludes the work and gives the directions for future work.

CHAPTER 2
BACKGROUND AND LITERATURE REVIEW

Spam creates a lot of problem in our daily life. Spammers are sending lot of junk mails to the user for their advertisement. They take the email address from Directory Harvest Attacker. The Directory Harvest attackers send blank mail to the mail server and collect the valid user addresses. There are many techniques to protect mail server from SPAM. However, if DHA is minimized then SPAM will be minimized more. Directory Harvest Attackers increase the load of the mail server also. There are different techniques to minimize the load of server, which is described below. Different SPAM detection techniques are also discussed. Lastly, the existing method of Directory Harvest Attack and their counter measures are described.

2.1 SPAM detection

Unsolicited communications sent in bulk over an electronic media such as e-mail, mobile (SMS, MMS) and instant messaging services, usually with the objective of marketing products or services, called SPAM. These are not requested mail. This definition of SPAM is restricted to situations where the receiver is not especially selected to receive the email. This would exclude emails looking for employment or positions as research students for instance. This difficulty in definition demonstrates that the definition depends on the receiver and strengthens the case for personalized spam filtering.

First, spammers get email address through many means. Some companies sell their mailing lists to third parties, spammers included. Spammers also use "robots" to scour the Internet and harvest any email addresses that they find. If the email addresses are posted in some business organization then the spammers pick up the email addresses and send junk email. To get adequate spam protection and get rid of Spam, more than

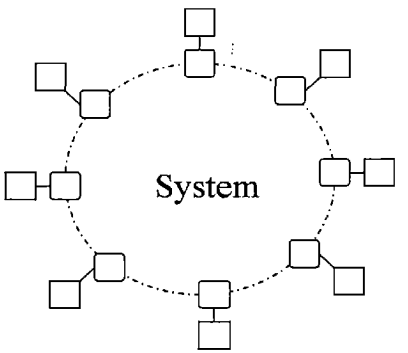
one email address should be used. This is an essential element of proper Spam control.

2.1.1 Distributed Method

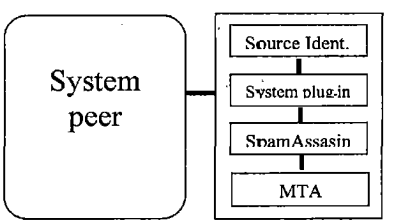
The assumption which has been taken to make the system is that a large class of spam-bots send large amounts of e-mail over short periods. Since it is believed that a large fraction of bots send very few spam each day, there must exist a significant fraction of bots that send hundreds of spam per day. Given that the average user only keeps their computer on for a few hours each day, such a bot must therefore send more spam per hour to meet its requirement [3]. A host that has recently sent large amounts of e-mails may be a spam-bot. Consequently, any e-mail coming from such hosts is potentially SPAM. If the source has a dynamically allocated IP address and the sender is not in the recipient's address book, then it is almost certain that the e-mail is spam. A spam-bot does not necessarily target multiple recipients within a single domain. It determines whether a bot is sending spam is inherently a collaborative effort. This distributed spam detection system identifies the source of each e-mail and then stores this information in a distributed database that is used and updated by all peers. This system depends on the collaboration of peers: as the number of peers increases, spam-bot identification improves because the sample size increases. This system classifies mails by several distinct parts: identifying the source of e-mails, keeping track of how many e-mails were recently sent by a source, and using this information for the purposes of classifying future emails.

When an e-mail arrives at mail transfer agent (MTA), the message is passed to a general spam classification framework, which classifies the mail, as spam or not, using different rule. The message is passed to a variety of plug-ins including a system plug-in, which is responsible for coordinating the source identification and source tracking tasks, and embedding the resulting classification in the e-mail.

The plug-in first passes the message envelope to the source identification server that is a locally running process. This server then determines whether the source of the e-mail has a static or dynamic IP address. If the source has a static IP address, existing blacklist mechanisms can be used. In that mechanism, if the source is listed in one of the blacklists, it can immediately be classified as spam. In this case, no further processing is necessary. Otherwise, if the source has a dynamic IP address, the address is returned to the plug-in for the next stage. The plug-in updates the distributed database used to track e-mail sources.



(a)



(b)

Figure 2.1: System (a) and module (b)

A local server, which is part of the distributed database, is assumed to run on the same host as the MTA or on a nearby host, propagates the update to the distributed

database, stores chunks of the distributed database, and caches updates and queries for the local MTA.

The plug-in queries the distributed database, via the local server, about the number of e-mails that were recently sent, typically within an hour, from the same source as the current e-mail—this is the sender score. This query may occur as part of the database update. If the sender score is high, i.e., many e-mails were sent, then the score is appended to the e-mail's envelope and returned to the spam framework (which identify the mail as a spam or not) itself. If the score is low, this indicates that either the source has not sent many e-mails recently, or that the e-mail may be one of the first of many e-mails that were sent. To distinguish these two cases, the plug-in can store the message for a short period, and then perform the query again. The score from the second query is then appended to the e-mail's envelope and returned to the framework. In our system, the quarantine has minimal impact because it is only used for emails arriving from senders with dynamic IP addresses.The technical challenges of the system is

-to correctly identify the source of an e-mail,

-to quickly and efficiently update the distributed database,

-to ensure that the database is not susceptible to poisoning from malevolent peers or to denial-of-service attacks,

-to ensure that the system scales well, is easy to install and maintain, and does not require excessive resources.

Determining the true source of an e-mail is difficult because the sender or any malicious mail relay can add false 'received lines' to the e-mail's envelope. Each relay through which an e-mail travels must prepend a 'received line' to the e-mail's envelope, indicating the host from which the relay received the email. The 'received line' prepended by the first trusted relay that received the e-mail contains the IP address of the host from which it received the e-mail. This host is considered the source of the e-mail. However, determining the first trusted relay is challenging.

Spammers would try to develop countermeasures against this system. The three main approaches would be

- to fool the source identifier,
- to poison the distributed database via false updates and malevolent peers in the distributed database,
- to launch denial of-service attacks against the distributed database with the very same botnets. The source identifier must deal with the first countermeasure. The latter two countermeasures must be considered in next section.

2.1.2 **Black list Method**

SPAM can be blocked by address-based filtering, using which one can refuse to accept mail from hosts that are believed to send spam. Once the host is identified, the IP address of a that host is registered in centrally maintained databases. This database is made available via the Internet DNS. Mail recipients can know that host by making query this database using standard DNS lookups and deny any mails from that host [4]. DNSBLs maintain various lists of IP addresses based on some criteria — for example, each IP address may be an open relay (spammers started relaying mail through hosts that would accept responsibility for delivering anyone's mail—these hosts are called open relays.), a virus source, or an actual spam source caught by a spam trap [5].

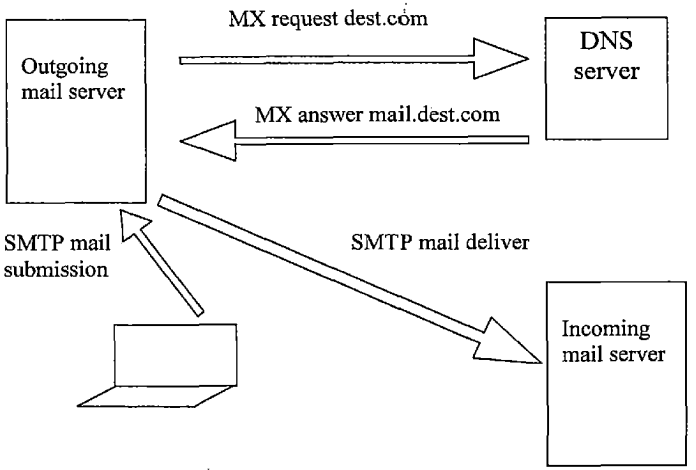


Figure 2.2: The Mail Delivery Process

How mails are transferred through internet to the destination are described as follows. After composing the mail, the sender hands off the mail to a local mail transfer agent (MTA) that delivers the mail to the final destination. This MTA is known as the injection point. If the sender is part of a large network, the local MTA may transfer the mail to additional hosts within the same administrative domain. Each host is said to relay the mail. One of the MTA's in the sender's organization will identify a host responsible for receiving mail for the recipient's domain, and relay the mail to that host. This host is known as the domain's mail exchanger, and is specified via a DNS MX record. At this point, the MTA on the recipient's mail exchanger may relay the message internally within the organization, ultimately arriving at an MTA that performs delivery, storing the message in a local mailbox for the recipient. MTAs relay mail between each other using the Simple Mail Transfer Protocol. We classify black lists based on two axes. First, we identify the

focus of the list second; we consider how addresses are added to the lists. Addresses can be added by downloading locally and using tools such as OpenBSD's spam and most people access these lists via the Domain Name System (DNS).

After getting the mail, MTAs check DNS black lists to determine whether the party relaying the mail is listed. Multiple lists may be checked. Then MTA can choose among the following

- to refuse the acceptence the mail
- end the transaction with an appropriate error code
- terminate the connection.

This activity happens before the mail is accepted locally.

Header Name	Description
To:	Address of intended receiver
Cc:	Address of other receivers
Bcc:	Address of other receivers
From:	Address of sender
Reply-To:	Address of recipient, usually the same as the senders address.
Received:	Line added by each transfer agent along the path that the email traveled.
Return-Path:	Can be used to trace sender.
Content-Length:	Length of email message

Figure 2.3: E-mail header

The blacklists either are static lists that are periodically updated by downloading new ones, or are stored in remote databases, which are themselves updated on a regular basis. A common technique, DNS Blacklists uses the DNS system to store

the blacklists and serve the queries. Blacklists are effective against hosts with static IP addresses but their response time is much too slow to counter botnets.

2.1.3 **Bayesian Approach**

Once the SMTP server has decided to accept a message, the sender transfers the entire set of message headers and the message body. Many filtering schemes work on the header and body (content filtering approach). It looks words or phrases which appear often in spam. Systems that require users to hand-build a rule set to detect junk assume that their users are perceptive enough to be able to construct robust rules. Early filters used fixed sets of strings, but spammers rapidly learned to avoid them by rewording their messages or using odd spellings. To overcome the problem, a junk mail filtering system should be able to automatically adapt to the changes in the characteristics of junk mail over time. Moreover, by having a system that can learn directly from data in a user's mail repository, such a junk filter can be personalized to the particular characteristics of a user's mail.

Let us see the classification scheme that provides a probability for its classification Decision. It helps to classify junk E-mail within a Decision Theoretic framework. To build probabilistic classifiers to detect junk E-mail, the formalism of Bayesian networks are employed. A Bayesian network is a directed, acyclic graph that compactly represents a probability distribution. In such a graph, each random variable Xi is denoted by a node [6]. A directed edge between two nodes indicates a probabilistic dependency from the variable denoted by the parent node to that of the child. To describe a probability distribution satisfying these assumptions, each node X_i in the network is associated with a conditional probability table, which specifies the distribution over X_i given any possible assignment of values to its parents. A Bayesian classifier is simply a Bayesian network applied to a classification task [6]. It contains a node C representing the class variable and a node X_i for each of the features. Given a specific instance x (an assignment of values x_1, x_2 ...xn to the

feature variables), the Bayesian network allows us to compute the probability P(C= c_k | X= x) for each possible class c_k. This is done via Bayes theorem, giving us

$$P(C= c_k \mid X= x) = P(X= x \mid C= c_k)\, P(C= c_k) / P(X= x) \quad [6] \ldots\ldots\ldots\ldots\ldots \text{(i)}$$

In the context of text classification, specifically junk E-mail filtering, it becomes necessary to represent mail messages as feature vectors to make such Bayesian classification methods directly applicable. Each individual message can be represented as a binary vector denoting which words are present and absent in the message. With this representation, it becomes straightforward to learn a probabilistic classifier to detect junk mail given a pre-classified set of training messages.

Content-based classification analyzes the contents and envelope of an e-mail using Bayesian networks. Such methods work well against known spam, i.e., spam that has been seen in the past and contains known strings or patterns. Some examples include using embedded pictures, clever use of different font sizes, and foreground and background colors. Content-based filters require never-ending tuning and adjustment in order to keep up with the spammers' latest tricks.

2.2 Load balancing Technique

Load Balancing is the dynamic sharing of load between servers, possibly on a per session basis. Usually using 'sticky' paths, where traffic will continue to flow via a particular route for the duration of a session. There are a number of ways to load balance traffic. The most common methods are [7]:

- Round-robin scheme
- Load balancing Switches and Routers

Round-robin scheme

Suppose, for a domain's MAIL servers to give a different IP address (actually, a different ordering of the set of possible IP addresses—A/B/C, B/C/A, C/A/B, A/B/C . . .) each time it is queried. Each of the IP addresses points to a logically identical server that is equally capable of handling the request. In addition, different clients are routed to different machines (with different IP addresses), this gives us a primitive form of load balancing.

The main advantage of round-robin scheme is that it requires no additional hardware. However, there are several disadvantages, which prevent many sites from using round-robin scheme for load balancing. The caching feature of mail server prevents complete load balancing because not every request that comes in will get its address directly from mail server.

It can be solved by disabling caching, but doing so means that every resolution will have to be resolved by servers, which is expensive and potentially slower for users. The mail server has no way of knowing if one or more of the servers in cluster is overloaded or out of service. Therefore, the round-robin scheme will send traffic to all servers in turn, even if some are overburdened or offline.

Load Balancing Switches

Load balancing switches are hardware Internet scalability solutions that distribute TCP requests across multiple servers. These switches sit between the connection to the Internet and the server cluster. All requests come to the switch using the same IP address, and then the switch forwards each request to a different server based on various algorithms implemented in the switch. Switches will frequently be able to ping the servers in the cluster to make sure they are still up, and to get an estimate of how busy they are so they can be relatively intelligent about load balancing.

Another common algorithm for load balancing is based on the content of the request. Perhaps the IP address of the requestor, or some other information in the request. Using the IP address alone does not work well since some ISPs, such as AOL, and companies use proxy servers that change the IP address of all of the requestors that go through the proxy to the same address. Using a load-balancing switch is much better and more scalable than using round-robin scheme, but switches can be quite expensive—and multiple switches are needed to avoid making the switch the single point of failure for entire server cluster.

2.3 Related Work and Research Gaps

2.3.1 Attack Method

Directory Harvest Attackers generate the email-id based on dictionary. They generate randomly a lot of email-id by different ways, which may or may not be valid. Some combination using user name or by combining of username with date-of-birth or with company name etc are being made. Attackers are also trying a different approach to get the valid user-id. They attack a particular domain mail server, which is responding well by sending blank message to many recipients, addresses of which are randomly generated [8]. After getting the valid email-id, they sell them to the spammer. Spammers then send a SPAM to these valid user-id and hence normal user get affected.

2.3.2 Protection Method

Many mechanisms are discussed to protect the server against DHA [9,10]. Attackers randomly generate an email-id and send them. A Captcha [9] is an automated Turing test which will differentiate a normal user and robot by asking a question, which can be answered only by human.

Introducing delay after unsuccessful trials is one of the solutions to protect DHA but it causes a DOS attack. Modifying the SMTP error message [10] is another technique to confuse the attacker. However, this creates problem for legitimate user. Another solution is to block the source sending a large volume of mails. This solution fails due to distributed attack.

Postini's white paper [11] has described the kind of attacks and way of stealing email-id from SMTP server by attackers. However, how to protect server against DHA is not described.

Other possible protection is to insert some delay into the authentication process after a number of unsuccessful trials. Although this can deny the attack from a single host, it cannot solve the problem with a distributed attack. If the whole system slows down then a DoS attack is possible.

Many commercial solutions also provide countermeasures against harvest attacks. The Kerio Mail Server [12] detects emails to unknown addresses and above a threshold; the server begins to filter out possible attackers. This method can be inefficient against a distributed attack.

Existing open source projects, like Project Honeypot [13] have not yet integrated protection against DHA. The ProjectHoneypot system tries to identify spammers by trap e-mail addresses. This can be extended by the identification of mass e-mail senders with many messages to unknown recipients. That was the initial idea described in this paper.

There are two types of protection

1. Host based protection: An autonomous system has its own protection method, without relying on other parties.

2. Network based protection: The system is cooperating with other parties to protect itself from the DHA. This method can be centralized: a server coordinates the protection.

The host-based filtering algorithm is not resistant against distributed attacks. Normally DHA attacks are highly distributed, the maximum of trials coming from an IP address can be as low as 1-2 [14]. Although the filtering algorithm can filter out attacking hosts, the attacker can utilize thousands of hosts, which are not yet known by the target. From a global aspect, the attacking computers should be detected all targeted SMTP server and should be filtered out by every individual filter routine to stop the attack.

Another important technique, which is published recently, is centralized RBL method [14]. It comes into Network based protection method. This technique is discussed below.

2.3.3 Centralized RBL method

A centralized protection method is implemented, where a real-time blacklisting server (DHA RBL server) gets information about every host that sends e-mails to unknown addresses. An e-mail SMTP server can query the DHA RBL server at the beginning of each incoming SMTP (e- mail) connection whether the DHA RBL server previously enlisted the sender as an attacker. If a sender is enlisted as a possible DHA attacker, the SMTP server rejects receiving e-mails from the sender. They proposed a method of centralized real black list server (RBL-server). Which store the information about source id and no. of emails coming from that user-id and marked which one is blacklisted. In this method, thay did not consider the load of server against DHA. This solution is also not sifficent for distributed attack.

Working Procedure of proposed model when attackers are not blacklisted is shown in the Figure 2.4.

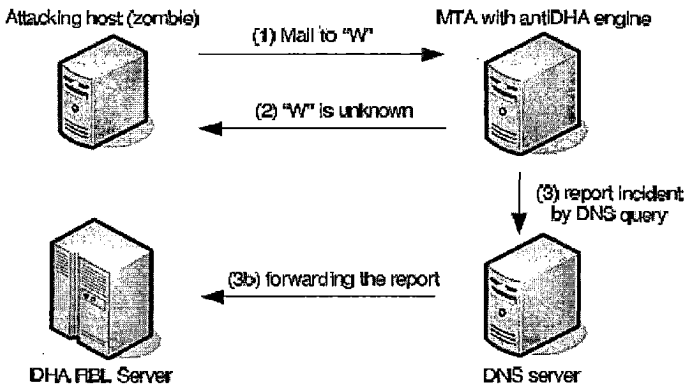


Figure 2.4: Reporting an incident to the anti DHA system [14]

Step 1. The attacker sends an e-mail to an internet mail server (MTA).

Step 2. The SMTP server answers with valid information: the user is unknown in the system.

Step 3a. The SMTP server sends an incident report to the server. This is done by a DNS query with a special format. The queried DNS name contains the information about the offending host.

Step 3b. The DNS server of the MTA forwards the query to another DNS server or directly to the DHA RBL server. The RBL server decodes the query and processes it

If an attacker sends an email to an unknown address on the attacked server, the attacked server will send an error report to the central DHA RBL server. The error report contains the offending IP address, the recipient address tried, and the time of the attack. The centralized server collects the reports from the protected servers. If the number of trials (reports) exceeds a limit, the server inserts the address of the attacker into the blacklist. The server also stores the timestamp when the last e-mail was observed from the given attacker with an unknown recipient. As usual, the RBL list can be queried by sending an address as a request to the server questioning if an address exists in the list. The server does not publish addresses on the list, instead it simply answers with yes or no.

Working procedure when enlisted attacker sends mail to the server is shown in the following Figure 2.5.

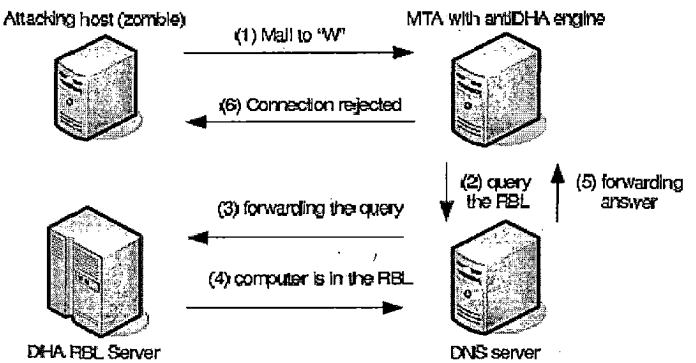


Figure 2.5: Filtering the attacker using the data from the DHA RBL [14]

Step 1. The attacker tries to send an e-mail to an internet mail server (MTA).

Step 2. The SMTP server sends a DNS query with the address of the client (the attacker) embedded in the query.

Step 3. The DNS server of the MTA forwards the query to the DNA RBL server.

Step 4. The RBL server answers the queries with a special form of IP address, meaning, "Yes, the computer is in the RBL list". The DNS server can cache the answer for a given address, the caching time (TTL- time to live) can be controlled by the RBL server.

Step 5. The DNS server sends back the answer to the SMTP server (protected host).

Step 6. The SMTP server denies the connection with the attacker. This can be done at TCP level, or the attacker can be denied with a proper SMTP error code and explanation.

2.3.4 Research Gaps

SPAM, sending by spammer, is increasing day by day. SMTP server has to respond with every request and has to process all the SPAM (i.e. sending to appropriate recipient). There are many techniques to filter the SPAM. Filtering the SPAM is not actually reduced the load of SMTP server. The reason is that SMTP server has to check to determine that, the mail is SPAM. Spammers send the SPAM to valid email recipient. They get the email address from the Directory Harvest Attacker. These attackers get the email address from the SMTP server. Therefore, it is better to protect DHA. Then automatically number of SPAM will be reduced. One paper [14] described, how efficiently DHA could be done. They also mentioned architecture to protect the SMTP server against DHA. However, this method is not suitable for distributed attack. Therefore, there is a gap to modify the architecture or implement another technique to prevent the attacker from getting the addresses. Even they did not describe how to minimize the load of SMTP server from processing the entire request coming from Directory Harvest Attacker. Directory Harvest Attackers send

blank message to multiple email addresses. These addresses may be valid or not. The SMTP server processes the request and sends the response as a positive or negative reply. Analyzing the response from the server, attacker determins the address is valid or not. Therefore, server has to processes the entire request. It is really a burden for server if lot of request comes in a short period. Then server cant processed the actual user's request. There should be some technique to overcome this problem. If central server is used then also there is problem, if it crashes. The approach, which is described to reduce the DHA, used central server. So server implementation is also a main concern to prevent this type of problem. Therefore, the area where research can be done is

- How to design the model so that attacker cannot get more valid user-id.
- How to design the model so that server can give good response to the normal user.
- How to design the model so that user request can be processed even server crashes.
- How to design the model so that server cannot accept more request from attacker.
- How efficiently server can block the attacker.
- How to maintain the black list for attacker.

CHAPTER 3
REDUCING DIRECTORY HARVEST ATTACKS

3.1 Framework for reducing DHA

Our model is based on the assumption that Directory Harvest Attackers send the blank mail to a large number of randomly generated recipients in a short time. Attackers use the normal user's machine by making them as zombie or they can use normal user's mail–id by hacking them. Mail server would be busy to process all the mail coming from attackers and due to this; normal user cannot get the good service from the server. SMTP Server gives the positive reply if the recipient's address is valid else give negative reply as invalid recipient. According to this reply, attackers store the email addresses to its own database for selling purpose. To minimize the effect of the attacker, a reply generator is introduced which will give the negative reply to the source after deciding the source as an attacker. There is a threshold value that will decide how many mails will be allowed to come from a specific source in a domain. The attacker can send a mail to a huge number of recipient's addresses, which are randomly generated using dictionary. Their main motto is to get a lot of valid email-id in a short time. Therefore, the source is blacklisted for a certain time. Else valid user cannot use their mail-id or their machine to send mail if the source is blacklisted forever. Front-end-filter is introduced to distinguish the mails and handles them in proper way without concerning mail server. Hence, reduce the workload of mail server. Again, to minimize the load we use a distributed mail server. The mail server that is less busy among the distributed servers will receive the mail. Front-end-filter will decide which one is less busy, by sending the ping request to all servers. Filter passes the mail to the server that responds first. All the mail servers are having their own database, which is used to decide blacklisted source. This information is passed to the front-end filter. Front-end filter distinguish the mail, which are from blacklisted source by comparing the source address with its own database.

The model is shown in the following Figure 3.1.

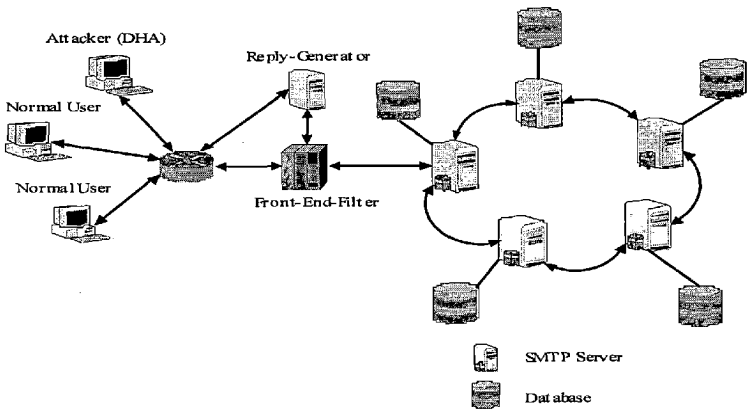


Figure 3.1: The structure of proposed model.

Now, the working principal of the framework in terms of response method, maintaining database, blocking criteria and distributed technique are discussed below.

3.2 Response Method

According to the response method of mail server Directory Harvest Attackers collect the valid user id. Attackers send the blank mail to the server. For invalid recipient, server sends back negative reply and for valid user sends back positive reply. Therefore, response method should be such a way so that attacker cannot get the valid user id. All the mails from user go to front-end-filter. This filter checks whether the source address of the mail is blacklisted or not. If mails are coming from blacklisted source, they are transferred to the reply generator through front-end-filter. Front-end filter sends not the whole mail instead; it sends the packet with source address only. Reply generator generates a negative reply to that source. Normally whenever mail

server gets the mail to unknown recipient, it gives a reply as invalid recipient address. Attackers observe the reply from the server. If the reply is invalid then they do not

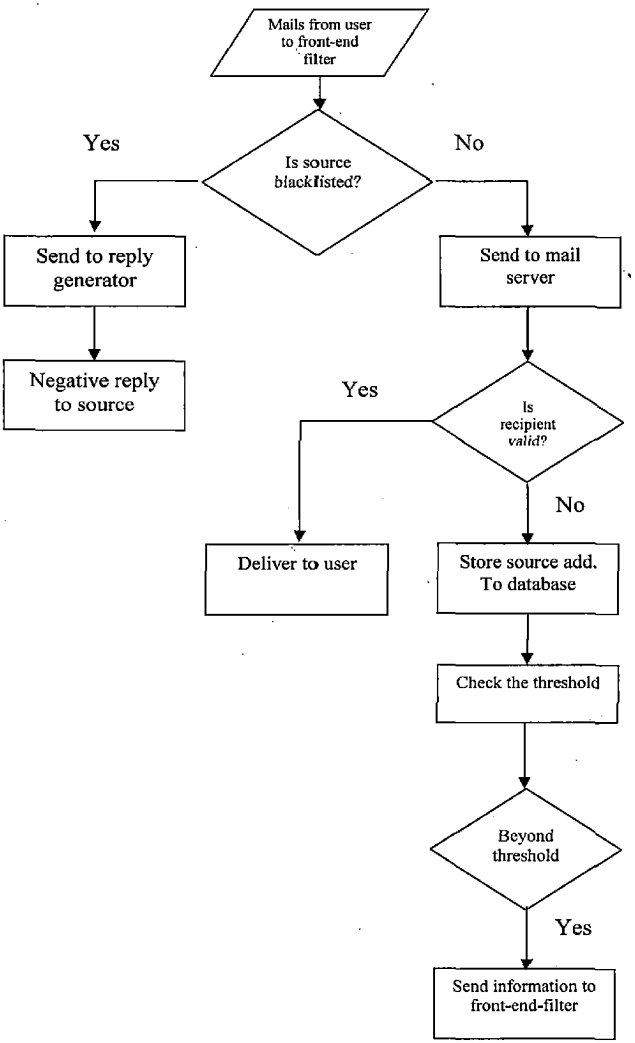


Fig 3.2: Flowchart of the working principle of the model

store this address to their database. Reply generator gives only reply as an 'invalid recipient address' to the source so that attacker will become confused. Attacker will think that the recipient's address is invalid, though there is chance to be valid. Therefore, once source becomes blacklisted, they get always this negative reply. Reply generator is used to help the server to be free from processing the mails coming from blacklisted source. Thus, reply generator reduces the load of SMTP server also from attacker.

3.3 Maintaining Database

Front-end filter and distributed SMTP server maintains the databases. SMTP server stores the source IP address and source email address and number of mails coming from that source. All the information is stored in its own database. Database contains two tables. One table contains the entry based on source IP address and other table contains entry based on source email address. Whenever server gets the mail from front-end-filter, it checks that this source address is already in its database entry or not. If this source address is already there then increment the count on corresponding source mail-id and source IP address. If the entry is not present then creates a new entry corresponding to that source. After updating their database, server shares this information with other servers in that domain. Each server communicates with others when they get any mail through update packet(s). This update packet(s) consists of source IP address and source mail address with a special flag. This flag is used to identify from where the packet came i.e. a packet may come from front-end filter in case of mail or it may come from server in case of update information. Therefore, flag distinguish the actual mail and a packet contains updating information. Servers increment the count entry corresponding to the source after getting packet from other server. Two threshold values are maintained, one is IP threshold and another one is user-is threshold. If the count is beyond that threshold value then this source is marked as blacklisted. Blacklisted information is passed to the front-end-filter, so that next time the server will not process any mail coming from the source.

The structure of the database maintained in the SMTP server and front-end-filter are shown in the Figure 3.3 and Figure 3.4 respectively.

Source IP address	Counter	timer

Source user address	Counter	timer

Figure 3.3 Database maintained by SMTP server

Source IP address	timer

Source user address	timer

Figure 3.4 Database Maintained by front-end filter

Mail server sends the update information to the other servers, which are distributed over there. It also sends the blacklist information to the front-end filter. A special packet carries out update information.

3.4 Blocking Criteria

For blocking purpose, source IP address and source mail-id are considered due to the reason that an attacker can use either same mail-id from different machines or can use different mail-id from same machine. A threshold is kept individually on both of this. The threshold can be decided depending upon individual policy of the domain users. The source with count above threshold is blacklisted and this information is conveyed to front-end-filter. Aging factor is also used, so that valid source for a certain time is not blocked for a long time. If no mails are coming from the blacklisted source for a

certain time then remove the entry corresponding to the source from the database. This will also keep a check on growing size of database.

Now let us see with example that how this model works.

Suppose, for simplicity in the mail server the existing user addresses are abc@domain.com, mno@domain.com and xyz@domain.com. The attacker sends a blank message from its own address. Lets the attackers email addresses are atk1@domain.com , atk2@domain.com. They use the different PC with IP address 192.168.1.2 and 192.168.1.3 and assume that the IP threshold is two and user-id threshold is one.

Now let's see what happens if the attacker starts to send the mail 1st time from atk1@domain.com address and from the PC whose IP address is 192.168.1.2 to unknown recipients address (cde@domain.com) i.e. which is not in the mail server's user list.

Front-end filter will pass this mail to the mail server since there is no entry in its own database corresponding to the source address of the mail. Mail server then checks the 'to' field of the mail which is cde@domain.com with its existing user list. This address is not listed in its existing user list. Mail server will now store the source address both (IP address and user-id) to its own database and set the count of the corresponding entry. Since the count is equal to the user-id threshold, mail server passes this information to the front-end filter. Front-end filter then store the user-id to its own database.

Therefore, after sending the mail from attacker the status of the database of the mail server and database status of the front-end filter are shown in the Figure 3.5 and Figure 3.6 respectively.

Source IP address	Counter	Timer
192.168.1.2	1	Starting time

Entry corresponding to source IP address

Source user address	Counter	timer
atk1@domain.com	1	Starting time

Entry corresponding to source user-id

Figure 3.5: Database status of SMTP server

Source user address	timer
atk1@domain.com	Starting time

Figure 3.6: Database status of front-end filter

Now, from the other PC (IP address 192.168.1.3) the attacker sends the mail from the same user-id (atk1@domain.com) to unknown recipients (aaa@domain.com). Front-end filter checks the source address with its own database. It will find that the source address is in the database, and passes this to the reply generator not to the mail server. Reply generator gives a negative reply to the source address. The attacker will block for the 15 minutes, if the blocking time is 15 minutes. Therefore, after blocking

attacker will not get the valid user id since reply generator will give negative reply always. The concept of blocking user address reduces the distributed attack. In the distributed attack the attacker, use the different IP address but same user-id.

The following Figure 3.7 shows the procedure when attacker is not blacklisted.

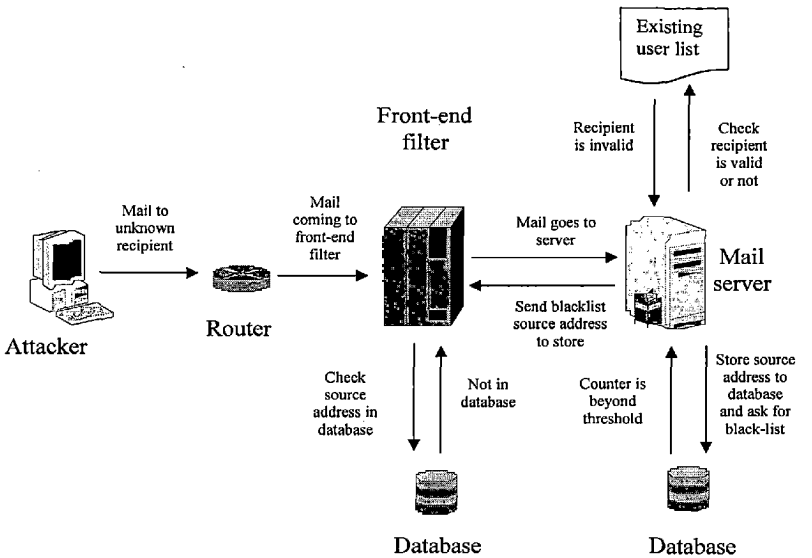


Figure 3.7: Procedure when attacker is not blacklisted

Now, let us see what happens, if the attacker sends the mail to the unknown recipient (kkk@domain.com) from IP address 192.168.1.2 and user-id atk2@domain.com.

The front-end filter will pass the mail to the server due to not found the source address of the mail in its own database. The server checks the 'to' field of the packet with its existing user list. The recipient address is invalid and server searches the source address in its own database. Now server increments the count corresponding to the source IP addresses and stores the source user-id in its source user-id database. Server also passes the IP address to the front-end filter since count is beyond the IP threshold.

So after this case the status of the mail server database and front-end filter are shown in the Figure 3.8 and 3.9 respectively.

Source IP address	Counter	Timer
192.168.1.2	2	Starting time

Source user address	Counter	timer
atk1@domain.com	1	Starting time
atk2@domain.com	1	Starting time

Figure 3.8: Database status of SMTP server

Source IP address	timer
192.168.1.2	Starting time

Figure 3.9: Database status of front-end filter

The following diagram (Figure 3.10) shows the procedure when attacker is blacklisted.

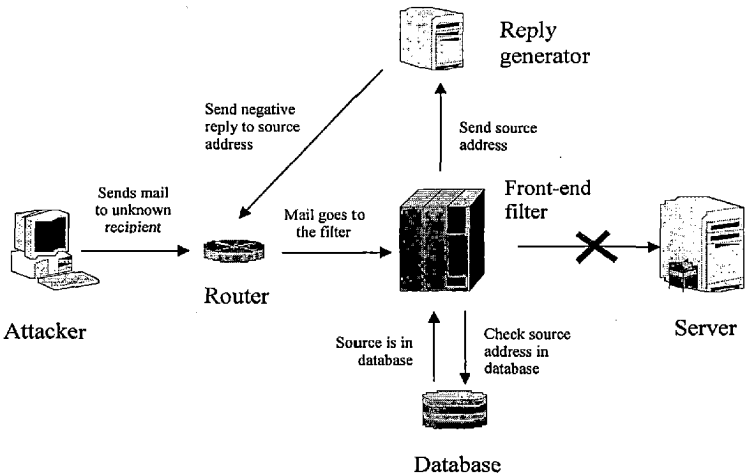


Figure 3.10: Procedure when attacker is blacklisted

This chapter describes the details of the proposed framework to reduce DHA. Distributed attack can also be minimized by blacklisting the user address. Normally in the distributed attack, system attacker use the different IP addresses i.e. different PC to send the mail using same user address. By blocking the user address, the attacker cannot send the mail to unknown recipient beyond a certain limit. Introducing front-end filter filters the mail coming from blacklisted source and transfers all the mails to reply generator, hence minimizing the load of SMTP server. This model is validated through network simulator (NS2). Next chapter describes the load minimization of SMTP server due to Directory Harvest Attack.

CHAPTER 4
MINIMIZING LOAD OF SMTP SERVER

Attackers send a lot of blank mail to the server to collect the valid user addresses. Due to this server takes a lot of time to process all the mails coming from attackers. Therefore, normal user will get slow response from mail server. So, it is necessary to reduce the load of mail server for giving better response to the user. Load can be distributed by using multiple servers. Blocking of all the mails coming from attackers reduces the load more. Front-end filter is used to block the attacker and distributed method is used to minimize the load. These two techniques, which are used to minimize the load of server, are discussed below.

4.1 Distributed Method

There are a number of SMTP servers, which are distributed along with their own database. These servers store the source IP address and source email address and number of mails coming from that source. All the information is stored in its own database. Database contains two tables. One table contains the entry based on source IP address and other table contains entry based on source email address. Whenever server gets the mail from front-end-filter, it checks that this source address is already in its database entry or not. If this source address is already there then increment the count of corresponding source user-id and source IP address. If the entry is not present then creates a new entry corresponding to that source. After updating their database, server shares this information with other servers in that domain. Each server communicates with others when they get any mail through update packet(s). This update packet(s) consists of source IP address and source mail address with a special flag. This flag is used to identify from where the packet came i.e. a packet may come from front-end-filter in case of mail or it can come from other server in case of update information. Therefore, flag distinguish the actual mail and a packet contains updating information. Servers increment the count entry corresponding to the source

after getting packet from other server. Two threshold values are maintained one is IP address and another one is for user-id. If the count is beyond that threshold value then this source is marked as blacklisted. Blacklisted information is passed to the front-end filter, so that next time the server will not process any mail coming from the source. There is some factor to decide the number of server. If number of server is increased then load will be minimized more whereas cost will be increased. (Cost includes server establishment cost and time requires updating or sharing their own database.) Distributed model is shown in the Figure 4.1.

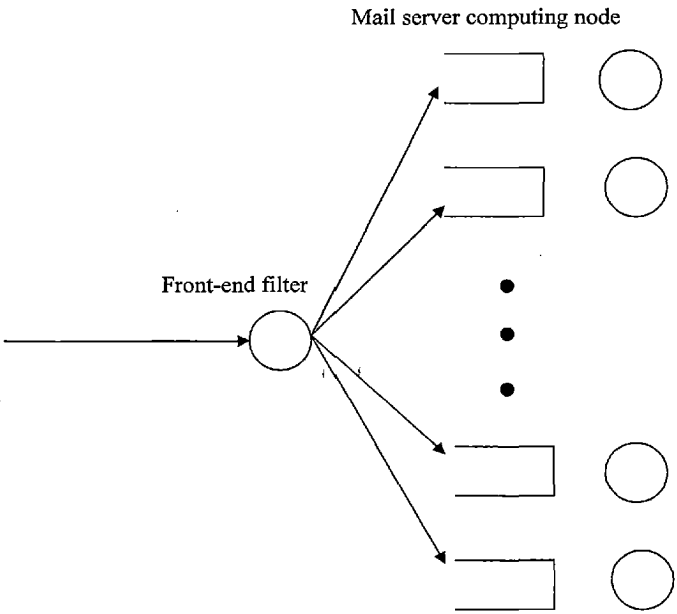


Figure 4.1: Structure of distributed mail server

Let T be the random variable denoting the client response time in the model, from the law of total probability we have,

E [T] $= \sum$E [T| request is served on node i]*P [request is served on node i]

$= \sum$E [Ti] P [request is served on node i]

Where E [Ti] is the expected response time of client requests served on computing node i.

4.2 Introducing front-end filter

Mails from attacker are not directly going to the SMTP server. Front-end-filter handles these mails and takes decision where to forward these mails. Front-end-filter can pass the mail either to the reply generator or to the distributed SMTP server, associated with filter. The filter maintains a list of source IP addresses and source email addresses, which are suspected as attacker. Based on this list filter forwards the mail. This suspicion is made by distributed SMTP server(s). Every mail, which is coming to the server, goes through front-end filter. After getting mail, filter checks the source address with its blacklist. If this is not in the blacklist then sends this mail to the server. There are more than one SMTP servers arranged in distributed fashion. Filter sends the mail to the server that is less busy than others. Filter 1st sends the ping request to every server. The server that is less busy gives response first. Front-end-filter then sends the mail to that first responded server. Thus, load is distributed among the distributed servers. After sending the mail, filter waits for the response. If server finds that, the source might be attacker then passes this information to the front-end filter. Filter then stores this source address in its blacklist. If next time any mail coming from this blacklisted source, filter forwards these mails to the reply generator. Hence, mail servers are free from processing the mails coming from suspected attacker. Front-end-filter maintains the blacklist for a certain time. If no mails are coming from a particular blacklisted source for last 30 min [16] then it removes this source address from blacklist. This information is passed to the distributed mail server. Then all the mail servers remove the whole entry corresponding to that source. If the blacklist is not modified after a certain time then a

normal user may be affected. The reason is that the attacker may use the normal user's address by hacking the email addresses or by making the machine as a zombie.

Here, front-end-filter is also working as a load distributor. Therefore, it not only blocks the attacker's mail but also distribute the load. Working principle of front-end filter is shown in the following diagram.

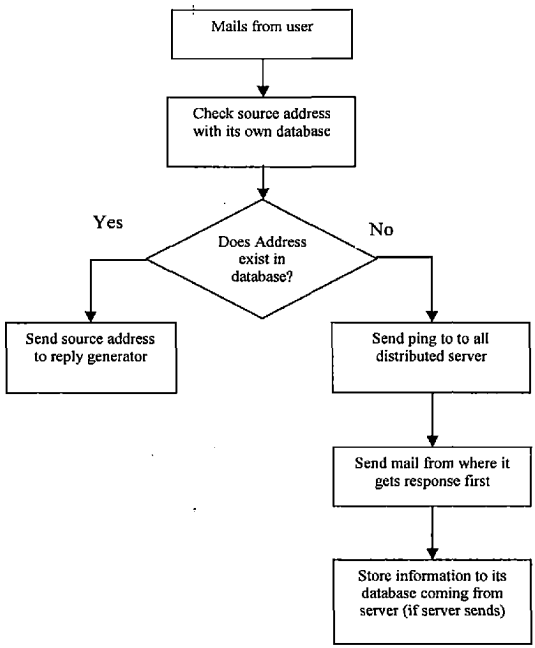


Figure 4.2 Flowchart of working principle of front-end filter

Microsoft exchange server can be used as a mail server. According to the survey [18] report, it is better than other server. This server is tested in two server configurations: a departmental server and a corporate server under various loads (medium and high). This server can be installed in distributed machine. This server also communicates with database. The response time of the server for different domain is different. For a departmental server configuration, average response would be 2 sec and for corporate server configuration, average response time would be 1.1 sec. So according to the organization number of server will be decided. The decision depends upon cost of the server and average response time of the server. The cost of Microsoft exchange server and hardware cost is the total cost to setup a server machine. If the number of distributed server increases then total cost will be increase whereas the overall response time of the server will be minimized. The graph shown in the results section (chapter 6), describes the effect of cost and response time by increasing number of server.

In this chapter, we have seen that how distributed technique is used to distribute the load of SMTP server and how front-end filter filters the mail coming from attackers. Another advantage of the distributed technique is that, if any server crashes then user would not be affected since other mail server will handle the request.

CHAPTER 5
SYSTEM DESIGN AND IMPLEMENTATION

5.1System Design

To investigate the effectiveness of the proposed framework in defending against Directory Harvest Attack and minimize the load of SMTP server, the simulation on a simplified topology has been carried out on Network Simulator (ns-2) [15]. A large number of scenarios are explored.

5.1.1 System Components

The system consists of the following components:

Clients: Clients are considered as a Directory Harvest Attackers. The attacker sends a blank mail to the user address (randomly selected from the file) from its own email address. The file contains valid and invalid recipients' address.

Server: The service provided by the server is to deliver the mail to the valid recipient. There are number of servers, which are distributed over there. If the recipient is invalid then reply, "recipient address is invalid" to the source address and store the information about source to its own database. Attacker sends a lot of blank mail to the server to get the valid user-id.

Front-end filter: This is the main gateway to the domain. Any mails coming to the server will come to that filter first. This filter filters the mails, which are coming from blacklisted source.

Reply generator: It only gives the negative reply to the source. It gets the source address from front-end filter.

Agents: Filter agent and SMTP agents are created in order to provide for the functionality of the proposed framework. They are deployed at front-end filter and SMTP server accordingly. They are discussed in detail next.

5.1.2 Simulation Model

Clients: Mail agents are deployed into client machine i.e. attacker machine. This agent maintains two files. When attacker sends the mail, agent fills 'to' and 'from' field of the mail by randomly choosing the data from two file. Among these files, one contains source address and another contains recipient address. In the recipients' address file, some are valid and some are invalid.

Server: The server is modeled by a simple destination of UDP packet. Here SMTP agents are deployed.

Filter agent: Filter agents are deployed in the front-end filter. Front-end filter maintains two databases to store the blacklisted source. Filter agent checks the source address with its own database. If the address is in the database then it sends the address to the reply generator, else sends this mail to SMTP server. After sending, filter agent stores the information returning from mail server (if any). Before sending, filter agent sends a ping request to all the servers. The server, which responds first, will receive the mail.

SMTP agent: This agent is deployed into mail server. After getting mail, it checks the source address with its existing user address. If the address is valid then do nothing else store the source address with count. If the count is greater than threshold then pass this information (containing source address) to the front-end filer.

5.2Implementation

5.2.1 Simulation Topology

Figure 5.1 illustrates the simulated network topology. The topology, which is introduced here, is different from the existing one. Here mail servers are distributed. Front end filter and reply generator are also new in the proposed framework.

The simulation is carried out in Network Simulator ns-2 [48] in which the functionality of filter agent, mail agent and SMTP agent are coded. Attacker sends mails from different IP address using different user-id to the recipient address, which may be valid or invalid. The implementation model is shown in the following Figure 5.1.

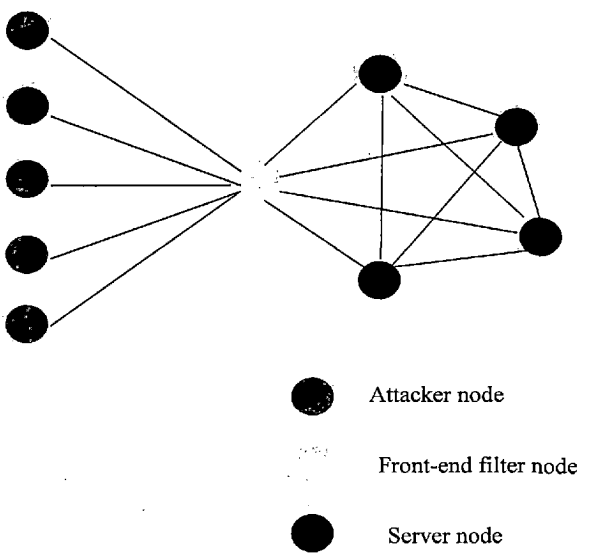


Figure 5.1 Simulation topology

5.2.2 Procedures

The procedures used to implement the various features are described next.

The database maintained by the front-end filter and SMTP server is implemented as a link list. Whenever new data arrives in the list, is added into the front. Attackers send the mail using UDP connection.

There are two input files. One is 'to.txt' file, which contains the recipients' address to which attacker will send the mail. This file contains valid and invalid both address. Here invalid addresses are those, which are not in the existing user list. Another file is 'from.txt' file. This file contains source user address. Mail agent takes the random address from these files and put into the 'to' and 'from' field of the packet (mail). The IP address will be node address from where mail is sent.

Front-end filter extracts the source address from the coming packet and matches with the database. If the source address is already in the database then do nothing else sends this packet to the server. SMTP agent then checks the 'to' field of the packet with the existing user address list. If it is not in the list, then stores the source address to its own database and increments the count field corresponding to that source address. If the count is beyond the threshold value then passes the source address to the front-end filter. Then filter adds the source address to its database. Front-end filter checks periodically if there is any address from where no mails are coming for last 30 min. If this type of address is there then delete this address from the database.

Simulation Parameters

The main purpose for simulation is to study the effect of DHA, load of mail server, cost and benefit of the proposed framework. The effect of the DHA is examined. The cost is incurred by the overhead of the server setup cost and mail server software cost. The benefit is measured in terms of number of mails going to the server and amount of load reduced in the mail server due to front-end filter and distributed server in the framework. IP threshold and user address threshold are varied to analyze the result. Other parameters, which are used, are listed below.

Parameter	Value	Description
Simulator	ns-2	Simulation tool
Number of attacker	5,15	Network nodes
Number of mail from one node	20,15	Mails coming from one IP address
Total recipient address	10,15	Destination email address

Table 5.1: Simulation Parameters.

CHAPTER 6
RESULTS AND DISCUSSION

Results of the simulated model (implemented in NS2) are analyzed and discussed by varying the parameter used in the model. Some parameters are taken as a fixed and some parameters are varied to analyze the effect of the Directory Harvest Attack and load of mail server. Two things are analyzed and discussed the results below

i) Effect of DHA
ii) Effect of Load of SMTP server

Effect of DHA

In the proposed model DHA can be minimized more by blocking user-id. If only IP address is blocked then DHA will be minimized but blocking user-id will minimize the effect of DHA more. Reply generator generates only negative reply. So that after blocking no attacker will get valid user-id, hence minimizing the attack. For simulation purpose existing user-ids are there which is in mail-server. There is two list of user-id of which each denote the 'from' field and to field of the mail from sender. List of recipient address contains 10 user ids where 6 are valid user id and others are invalid user-id. In another case recipient address contains 15 user ids where 6 are valid user id and others are invalid user-id. Sender sends the mail by randomly choosing the 'to' and from filed. The result is observed by varying the number of attackers' node, IP threshold and user-id threshold.

In the first graph we have shown the filtering by taking total number of attacker's node as five and from one node, 20 mails are sent to the mail server i.e. total 100 mails are coming to the server from attacker. Attackers use different user id, which is chosen randomly from input file. The input file contains 6 valid user-ids and 4 invalid user-ids. The source address is also filled randomly from an input file. Server maintains a file containing existing user address. Two parameters are varied to analyze the result, one is IP threshold and another one is user-id threshold.

User-id threshold denotes how many mails are allowed to unknown recipient from sender email-id and IP threshold denotes how many mails are allowed to unknown recipient from an IP address. IP threshold is represented along with x-axis. It varies from 1 to 20. Y axis represents no. of mails going to the mail server after filtering. Here total 100 mails are sent from attacker, 20 mails from each node. In the centralized RBL method, only IP address is blocked. In our model, user-id is also blocked. In the centralized RBL method, IP threshold value is taken as 10. The following graph (fig 4) shows how much filtering is there by varying IP threshold and user-id threshold.

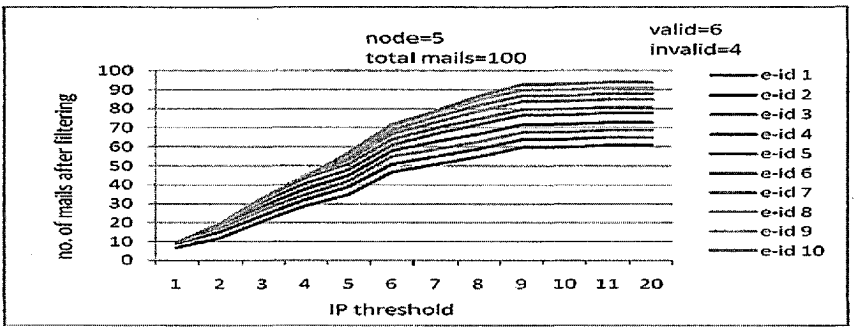


Figure 6.1: Filtering graph where attacker node is 5 and total recipient address is 10

	mails after filtering									
ThrshldIP	e-id=1	e-id=2	e-id=3	e-id=4	e-id=5	e-id=6	e-id=7	e-i=8	e-id=9	e-id=10
1	7	9	9	9	9	9	9	9	9	9
2	12	15	17	19	19	19	19	19	19	19
3	21	24	27	29	31	33	33	33	33	33
4	29	32	35	38	41	44	45	45	45	45
5	35	39	42	45	48	51	53	54	56	57
6	47	51	55	58	61	64	67	69	71	72
7	51	55	59	63	67	70	73	76	78	79
8	55	59	63	67	72	75	79	82	85	87
9	60	64	68	72	77	80	84	87	90	93
10	60	64	68	72	77	80	84	87	90	93
11	61	65	69	73	78	81	85	88	91	94
20	61	65	69	73	78	81	85	88	91	94

If we decrease the IP threshold, then filtering is more. The reason is that, if we take IP threshold as 10, then after attempting 10 wrong invalid addresses from one PC, the rest of all mails will be filtered. If this value is five, then after attempting five wrong invalid addresses from one PC, the rest of all mails will be filtered. So more mails will be sent if IP threshold value is larger. We can see also the effect of DHA due to user-id threshold in the graph (Figure 6.1). If the value of user–id threshold is less, more mails will be filtered. Only IP threshold is not sufficient to reduce DHA effect. In distributed attack, the attackers send 1-2 mails from one IP address and use many IP address. Hence, user-id threshold is more important to protect distributed attack in some extent. Now, let us see the effect of DHA by taking attackers node as 15 and from each node 15 mails are sent i.e. total 225 mails are sent to the server. Here IP threshold and user-id threshold are also varied to observe the effect of DHA. The following graph (Figure 6.2) shows the filtering.

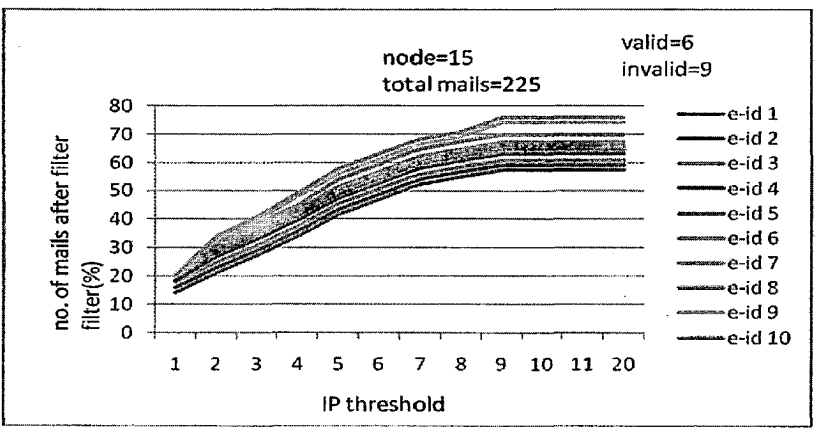


Figure 6.2: Filtering graph where attacker node is 15 and total recipient address id is 15.

	Mails after filtering									
thrshldIP	e-id 1	e-id 2	e-id 3	e-id 4	e-id 5	e-id 6	e-id 7	e-id 8	e-id 9	e-id 10
1	32	36	40	42	45	45	45	45	45	45
2	48	52	56	61	63	66	69	72	74	77
3	61	65	69	74	77	80	83	86	89	92
4	76	80	84	89	92	95	98	101	108	111
5	94	98	102	107	110	113	116	122	126	130
6	106	110	114	119	122	125	128	134	138	142
7	118	122	126	131	134	137	140	146	150	154
8	124	128	132	137	140	143	146	152	156	160
9	129	133	137	142	145	148	151	157	167	171
10	129	133	137	142	145	148	151	157	167	171
11	129	133	137	142	145	148	151	157	167	171
20	129	133	137	142	145	148	151	157	167	171

We also observed the filtering in the following graph (Figure 6.3) by taking attackers node as 5, number of mails from one node as 20 i.e. total 100 mails are sent to the server, number of valid user-id as 6 and invalid user-id as 9.

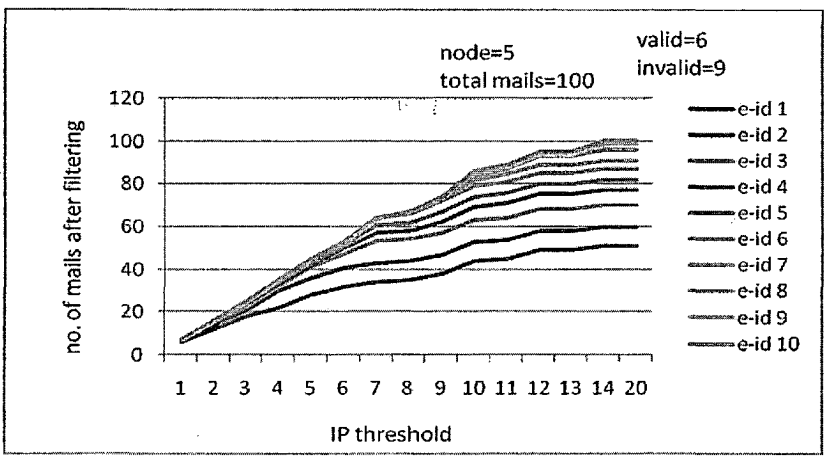


Figure 6.3: Filtering graph where attacker node is 5 and total recipient address id is 15.

	mails after filtering									
thrshldIP	e-id 1	e-id 2	e-id 3	e-id 4	e-id 5	e-id 6	e-id 7	e-id 8	e-id 9	e-id 10
1	6	7	7	7	7	7	7	7	7	7
2	12	14	16	16	16	16	16	16	16	16
3	18	21	23	24	25	25	25	25	25	25
4	22	30	32	33	34	35	35	35	35	35
5	28	36	41	42	43	44	45	45	45	45
6	32	41	47	50	51	52	53	53	53	53
7	34	43	53	57	61	64	64	64	64	64
8	35	44	54	58	62	66	67	67	67	67
9	38	47	57	62	67	72	73	74	74	74
10	44	53	63	69	74	79	82	84	85	86
11	45	54	64	71	76	81	85	87	88	89
12	49	58	68	75	80	85	89	93	94	95
13	49	58	68	75	80	85	89	93	94	95
14	51	60	70	77	82	87	91	96	99	100
20	51	60	70	77	82	87	91	96	99	100

In all the cases, the graph is almost same. No matter how many attackers are there. Filtering depends upon IP threshold and user-id threshold both. If attackers are more then filtering will be there but more mails will be coming.

Value of user-id threshold depends upon IP threshold and normal user's service time. Let us assume IP threshold is 6. In this case, user-id threshold should not be more than 6. If from one IP address, 6 mails are sent from one user-id to unknown recipient then after that all mail will be blocked from that source. Since IP threshold is 6. The user-id threshold should not be so less. If the value of user-id threshold is so less, then normal user may be affected. Suppose blocking time is 30 min, which is normally used in other mail server [16]. In this case, if user-id threshold is one, then after sending one mail to unknown recipient, user will be blocked for 30 min. That means in 30 min. they can send one mail (if this mail goes to unknown recipient address). Now let us see the number of attacker's mail coming to the server. If the user-id threshold is one then 47 attackers mail will be sent. Therefore, the user-id threshold depends upon IP threshold, number of attacker's mail and waiting time of a normal user. The following graph (Figure 6.4) shows the effect of waiting time and number of attacker's mail by varying user-id threshold. If the user-id threshold is greater,

waiting time will be reduced and number of mails from attacker will be increased. In X-axis, one unit is 10 sec for waiting time.

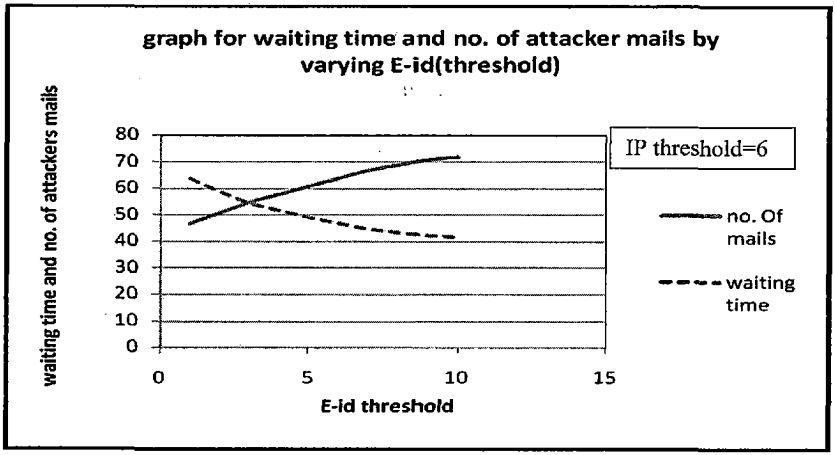


Figure 6.4 Effect of waiting time and no. of attacker's mail by varying the user-id

	e-id=1	e-id=2	e-id=3	e-id=4	e-id=5	e-id=6	e-id=7	e-i=8	e-id=9	e-id=10
attacker mails after filter	47	51	55	58	61	64	67	69	71	72
waiting time(ms)	63.82	58.82	54.54	51.72	49.18	46.87	44.77	43.47	42.25	41.66

Say, IP threshold is 6. Now in the graph it can be seen that if user-id threshold is increased then waiting time will be minimized but attacking mail will be increased. Say blocking time is 30 min [16]. Then for user-id threshold=1 no of mails will be 47 hence in 30 sec 47 mails can go. Therefore, for one mail (30000/47=63.82*10) msec is needed for user-id threshold=1 and for user-id threshold= 2 (30000/51=58.82*10) msec is needed. So, if no of user-id threshold is increased, time is decreased to send one mail and no of attackers mail is increased. For user-id threshold=1, no. Of attacker mail is 47 and for user-id threshold=2, it is 51. So convergence factor should be 2to 4 which is shown in the graph to decide user-id threshold for IP threshold=6.

6.2 Effects of Load of SMTP Server

Load of SMTP server for front-end-filter

If the attacker's mails reduce, load of the server will be minimized. Front-end filter blocks the mails coming from attacker and thus it reduces the load of SMTP server. In the following graph it can be seen that if IP threshold value is decreased then no. of mails from attacker to server is minimized, hence load of server is also minimized. For a particular value of IP threshold, if user-id threshold is varied then load is also varied. If the user-id threshold is decreased then no. of mails from attacker will be minimized hence load will also be minimized. The reason is that from blocking source no mails will go to the server. All the mails from that source will go to the reply generator. In the following graph (Figure 6.5) x-axis denotes the IP threshold, and y axis denotes number of mails which are going to the mail server i.e load of server. IP threshold is varied from 1 to 12 and beyond that all are same. User-id is varied from 1 to 10. It can be seen that if IP threshold value is increased then no. of mails, which are going to mail server are increased hence load is increased. For a particular IP threshold, if user-id is increased then also load is increased so user-id threshold should be less than IP threshold and not so less. If user-id threshold is so less then waiting time of normal user will be increased that is seen before.

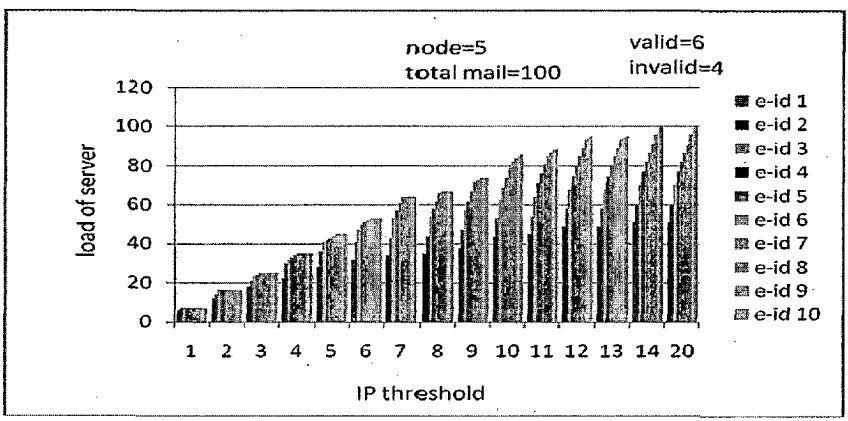


Figure 6.5: Graph for Load of SMTP server for front end filter

The result has been taken by changing the number of attackers' node and number of recipient address. The following graph shows the load of server when attacker node is 15 and number of recipient address is 15. Load is shown here as percentage.

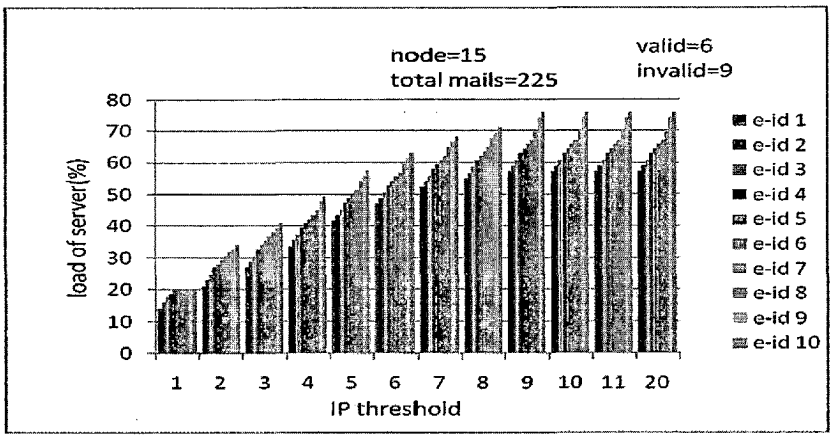


Figure 6.6: Graph for Load of server for front end filter where attacker node is 15

Both graphs are mostly same. In the second graph (Figure 6.6) number of attackers' mail are slightly greater than 1st graph (Figure 6.5) for a particular IP and user-id threshold due to more attacker nodes. But load is minimized for frnt-end filter which are not in the centralized RBL method.

Load of SMTP server for distributed method

If there are no. of server then load will be distributed, hence load will be minimized for each server. No of server depends upon some fact. If the no of server is increased then cost of server will be maximized. This cost includes server establishment cost, delay to send request to other servers. Other side if no of server is increased then response time of the server will be minimized.

If there is n number of server then response time will be

$E[T] = E(T \mid \text{request is served on node i}) * P[\text{request is served on node i}]$

$= \sum_{\text{For all i}} E[Ti] * P[\text{request is served on node i}]$

Where, P[request is served on node i]= probability to send mail to i^{th} server.

E [Ti] = expected response time of client requests served on computing node i.

T is the random variable denoting the client request response time and E[t] is the total probability.

In the following graph 6.6 X-axis denotes no. of server and Y axis denotes the cost and load. Say, for simplicity cost of each server is constant k=$2000(setup cost) +$800(s/w cost) [17] and response time for each server is 2 sec [18]. Probability of choosing one server among n server is 1/n. In the following graph, no. of server is taken from 1 to 10. In Y-axis for cost, per unit is $1000 and for response time, one unit is 1 sec. From this graph, the optimum value of the no. of server is 4 to 6.

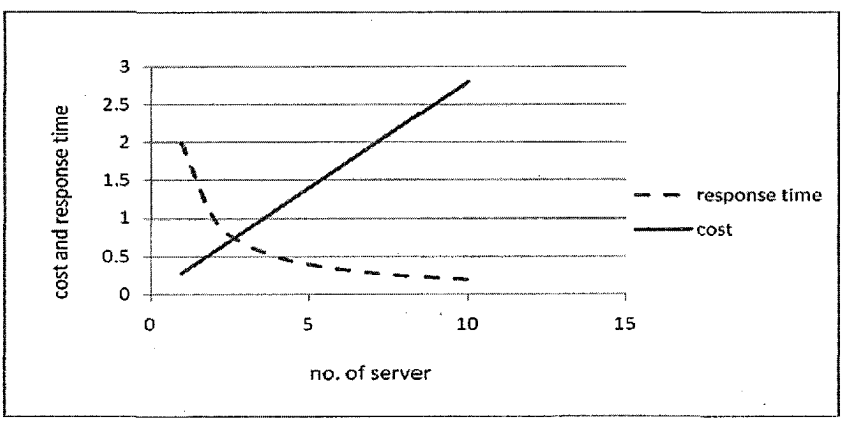


Fig 6.6: Graph for cost and response time varying the number of server

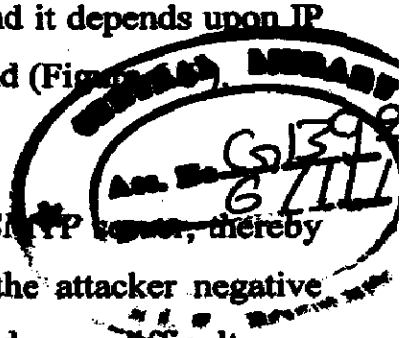
No. Of server	1	2	3	4	5	6	7	8	9	10
response time	2	1	0.66667	0.5	0.4	0.33333	0.28571	0.25	0.22222	0.2
cost	0.28	0.56	0.84	1.12	1.4	1.68	1.96	2.24	2.52	2.8

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1 Conclusions

The framework proposed as part of the dissertation provides an end-to-end solution for defense against Directory Harvest Attack (DHA). Currently, no framework exists that would minimize both DHA and load of server like the proposed framework. The effectiveness of the framework has been illustrated by appropriate simulation test bed.

Attackers send many mails to the SMTP server. The server processes all these mails and wastes its lot of time in processing and replying though attacker might be blacklisted. The normal users' requests are affected due to this. In our model, the mails, from blacklisted attackers are sent to the reply generator. The front-end-filter sends ping to all the SMTP servers to check which one is free. The relatively free server will respond first and hence reduces load from the busy servers. In the previous paper [14] the load of server is 100% because all the mails are processed by mail server. In our model load is minimized due to front-end filter and it depends upon IP and user-id threshold. This is shown in the graph (Figure 6.5) and (Fig



The reply generator will then process such requests instead of SMTP server, thereby saving the time of SMTP server. This reply generator gives the attacker negative reply, hence make the attacker's task of selecting the valid mail-id more difficult.

The blocking criterion depends upon the domain administrator. The domain administrator decides the value of IP-threshold i.e. how many mails to unknown recipient address are allowed from one PC. The threshold of user address should be less than threshold of IP address. Because from one machine no user cannot send the number of mails to invalid recipient which is more than the value of IP-threshold. The

value of user-id threshold will not be so less. If it is then the user will be blocked for a long time due to single mistake. Therefore, front-end filter minimizes the server load by filtering the attacker's mail and distributing the load.

7.2 Suggestions for Future Work

Policies for the threshold value can be decided after analyzing the real case. Some heuristics may be used to do that. Data sharing between distributed SMTP servers and its maintenance can be made more efficient and secure way. The actual number of SMTP server, which is sufficient to distribute the load, may be further analyzed.

Reducing the effect of DHA can be more optimized by analyzing the existing user's friend list. This observation will protect the server from false positive. Attackers can send mail to the valid user address but may not be in recipient's friend list. Therefore, some threshold value can be set to decide how many mails of this type i.e. mail to valid user but not his friend will be allowed.

REFERENCES

[1] *Spam Assassin. (http://spamassassin.apache.org)*

[2] *Vicomsoft connect and protect,spam. (http:// www.vicomsoft.com)*

[3] A.Brodsky,D.Brodsky. A distributed content independent method for spam detection. *In First Workshop on HOT topics in Understanding Botnets* (2007).

[4] J. Jung, and E. Sit. An empirical study of spam traffic and the use of dns black lists. In *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement (2004).*

[5] A.Iverson. Dnsbl resource. http:// www.dnsbl.com /, 2007.

[6] M. Sahami, S. Dumais., HECKERMAN, D., AND HORVITZ, E. A bayesian approach to filtering junk e-mail. *In AAAI-98 Workshop on Learning for Text Categorization (1998).*

[7] Load Balancing VisNetic MailServer (www.deerfield.com/support/VisNetic-MailServer)

[8] *Postini Enterprise Spam Filtering.* The Silent Killer: How Spammers are Stealing Your Email Directory. http://www.postini.com/whitepapers/,2006.

[9] L. V. Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security," *in Proceedings of Eurocrypt*, pp. 294-311, 2003.

[10] J. Klensin, Simple Mail Transfer Protocol, RFC 2821, Apr. 2001.

[11] *Postini Enterprise Spam Filtering.* The Silent Killer: How Spammers are Stealing Your Email Directory. http://www.postini.com/whitepapers/,2006

[12] *Kerio MailServer* - State-of-the-art Secure Email Server. (http://www.kerio.com/kms/home.html)

[13] Project Honeypot - Distributed System for Identifying Spammers. (http://www.projecthoneypot.org)

[14] B. Bencsath, I. Vajda, "Efficient directory harvest attacks". *In Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems, pp. 62- 68, IEEE Computer Society, 2005.*

[15] *NS-2 Network Simulator*, available at, http://www.isi.edu/nsnam/ns/

[16] *Blocking time*: http://www.pcmag.com/article2/0,1759,1543581,00.asp

[17] *Cost of server*: http://www.microsoft.com/exchange/howtobuy/default.mspx#EIC

[18] *Response time of the server:* http://www.microsoft.com/presspass/press/1997/ jun97/msexc5pr.mspx

Publication

1. Suman Das, Rajeev Singh and Dr. Durga Toshniwal. "Distributed Method to Minimize the Effect of Directory Harvest Attack and Load of SMTP Server". *In 1st International Conference on Advances in Computing*, Chikhli, India, 21-22 February 2008.

2. Suman Das, Rajeev Singh , Dr. Ramesh C. Joshi and Dr. Durga Toshniwal. "Minimize the Effect of Distributed Directory Harvest Attack and Load of mail Server". *In 3rd IEEE International Conference on Industrial and Information Systems (ICIIS 2008)*, IIT Kharagpur, India. (paper submitted).

#mail-agent.h

```
#ifndef _MAIL_AGENT_H
#define _MAIL_AGENT_H

#include <stdio.h>

#include "agent.h"
#include "packet.h"

#include "dmail.h"

class MAILAgent : public Agent {
    public:
        MAILAgent();
        int command(int, const char * const *);
        void recv(Packet *, Handler *);
    protected:

        Agent *upper_agent_;
        void send_up(Packet *, Handler *);
        void send_down(Packet *, Handler *);
};
#endif
```

#mail-agent.cc

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <fstream.h>
#include "ip.h"
#include "tcp.h"
#include "mail-agent.h"
#include "dmail.h"
static class MAILHeaderClass : public PacketHeaderClass {
    public:
        MAILHeaderClass() : PacketHeaderClass("PacketHeader/MAIL",
sizeof(mail))
        {    bind_offset(&mail::offset_);  }
} class_mailhdr;

static class MAILAgentClass : public TclClass {
    public:
        MAILAgentClass() : TclClass("Agent/MAIL") {}
        TclObject* create(int, const char * const *) {
            return (new MAILAgent());
        }
} class_mailagent;

MAILAgent::MAILAgent() :Agent(PT_DMAIL)
{
    //bind("from_smtp_", &from_smtp_);
    //bind("bl_list_", &bl_list_);

}
```

```
int MAILAgent::command( int argc, const char*const* argv)
{
      static int y=0;
      if (argc == 2)
      { if (strcmp(argv[1], "send") == 0){
      y++;
      int j,k;
      j=(rand() % 10)+1;
      k=j;
      ifstream filefrom ("frommail.txt", ios::in);
      ifstream fileto ("tomail.txt", ios::in);
      char from[10],to[10];
      while(k){
                  filefrom.getline (from,10);
                  fileto.getline (to,10);
                  k--;
            }
      filefrom.close();
      fileto.close();

      Packet* p = allocpkt();
      mail *amail=mail::access(p);
      hdr_ip* hip = hdr_ip::access(p);
      ofstream fileop1("mailagent.txt", ios::out | ios::ate |
ios::app);
      fileop1<<from<<"\t"<<to<<y<<"\t"<<hip->saddr()<<"\n";
      fileop1.close();

      strcpy(amail->to, to);

      strcpy(amail->from, from);
      amail->from_smtp=0;
      amail->ipbllist_=0;
      amail->bllist_=0;
      send(p,0);

      return (TCL_OK);
    }
  }
    return (Agent::command(argc, argv));

}
```

#Filtering-agent.h

```
#ifndef _FILTERING_AGENT_H
#define _FILTERING_AGENT_H

#include <stdio.h>

#include "agent.h"
#include "packet.h"

#include "dmail.h"
```

```
struct DATABASE {
      char from[20];
      int count;
      struct DATABASE *next;
};
struct IPDATABASE {
      int from;
      int count;
      struct IPDATABASE *next;
};

class FILTERINGAgent : public Agent {
    public:
        FILTERINGAgent();
        int command(int, const char * const *);
        void recv(Packet *, Handler *);
       void show();
          DATABASE *db;
          IPDATABASE *ipdb;

  protected:
        int offset_,bl_list_,nodeid_;
    private:

        void delete_dbase(Packet *);

        void delete_IPdbase(Packet *);
        void recv_mail(Packet *);

      IPDATABASE* lookup(Packet *,IPDATABASE *);
void insert_dbase(Packet *);
void insert_IPdbase(Packet *);
void recv_update(Packet *);
DATABASE* lookup(Packet *,DATABASE *);
bool lookupdb(Packet *);
bool lookupipdb(Packet *);

};

#endif /* _FILTERING_AGENT_H */
```

#Filtering-agent.cc

```
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <string.h>
#include "ip.h"
#include "tcp.h"
#include "filtering-agent.h"
#define THRESHOLDIP 9
#define THRESHOLD 10
int mail::offset_=-1;
static int countmail=0,m;
static class MAILHeaderClass : public PacketHeaderClass {
    public:
```

```
        MAILHeaderClass() : PacketHeaderClass("PacketHeader/MAIL",
sizeof(mail))
        {
            bind_offset(&mail::offset_);
        }
} class_mailhdr;
static class FILTERINGAgentClass : public TclClass {
    public:
        FILTERINGAgentClass() : TclClass("Agent/FILTER") {}
        TclObject* create(int, const char * const *) {
            return (new FILTERINGAgent());
        }
} class_filteringagent;

FILTERINGAgent::FILTERINGAgent() :Agent(PT_DMAIL)
{
   bind( "bl_list_", &bl_list_ );
   bind( "nodeid_", &nodeid_ );
      //test=new DATABASE;
      db = new DATABASE;
      db->next=NULL;
      db->count=0;
      ipdb = new IPDATABASE;
      ipdb->next=NULL;
      ipdb->count=0;
}
void FILTERINGAgent::show()
{     DATABASE *tmp=db;
      while(tmp!=NULL){
            tmp=tmp->next;
            //printf("%s",tmp->from);

      }
}
int FILTERINGAgent::command(int argc, const char*const* argv)
{
    Tcl &tcl=Tcl::instance();
    return Agent::command(argc, argv);
}
void FILTERINGAgent::recv(Packet *p, Handler *)
{

mail *amail=mail::access(p);
hdr_ip* hip = hdr_ip::access(p);
//printf("fromsmtp= %d\t%d",amail->from_smtp,hip->saddr());
//printf("dest. addr = %d",hip->daddr());
//printf("source. saddr = %d",hip->saddr());
   if(amail==NULL)
    {
        printf("PANIC: mail is NULL\n");
        abort();
    }
    if(amail->from_smtp==1)

      {     //printf("receive update");
            recv_update(p);
      }
```

```
    else
      recv_mail(p);
}

void FILTERINGAgent::recv_mail(Packet *p)
{
      int flag=0;
      m++;
      mail *amail=mail::access(p);
      hdr_ip* hip = hdr_ip::access(p);
      ofstream fileop("filter.txt", ios::out | ios::ate |ios::app);
      IPDATABASE *tmpipdb ;

      if(lookupdb(p))
      {
            tmpipdb=lookup(p,ipdb);
            flag=1;
            if(tmpipdb->next!=NULL)
            {
                  (tmpipdb->next)->count++;

            }
            else
                  insert_IPdbase(p);
      }
      if(lookupipdb(p))
            flag=1;
      if(!flag)
      {     //printf("filter to mail: %s \n",amail->from);
            countmail++;
            printf("count : %d\n",countmail);
             fileop<<countmail;
      fileop<<amail->from<<"\t"<<amail->to<<" --- "<<hip-
>saddr()<<"\n";
            Tcl& tcl = Tcl::instance();
            tcl.evalc("getnodeid");
            hip->daddr()=nodeid_;
            send(p,0);
      }
      fileop.close();
      //show();
}
IPDATABASE* FILTERINGAgent::lookup(Packet *p,IPDATABASE *ipdb)
{
      hdr_ip* hip = hdr_ip::access(p);
      IPDATABASE *tmp1=ipdb;
      while(tmp1->next)
      {
            if(tmp1->next->from==hip->saddr())//same
                  return tmp1;
            else
                  tmp1=tmp1->next;
      }
      return tmp1;
}
```

```
void FILTERINGAgent::insert_IPdbase(Packet *p)
{
	IPDATABASE *tmp1 = new IPDATABASE;
	hdr_ip* hip = hdr_ip::access(p);
	tmp1->count=1;
	tmp1->from=hip->saddr();
	tmp1->next=ipdb->next;
	ipdb->next=tmp1;
}

 void FILTERINGAgent::insert_dbase(Packet *p)
{
	DATABASE *tmp = new DATABASE;
	mail *amail=mail::access(p);
	tmp->count=1;
	tmp->next=db->next;
	strcpy(tmp->from,amail->from);
	db->next=tmp;
}


void FILTERINGAgent::recv_update(Packet *p)
{
	mail *amail=mail::access(p);
	DATABASE *tmpdb ;
	IPDATABASE *tmpipdb ;
	//if(amail->bllist_==1)
		tmpdb=lookup(p,db);
		if(tmpdb->next!=NULL)
			(tmpdb->next)->count++;
		else
			insert_dbase(p);

		tmpipdb=lookup(p,ipdb);
		if(tmpipdb->next!=NULL)
			(tmpipdb->next)->count++;
		else
			insert_IPdbase(p);
}

DATABASE* FILTERINGAgent::lookup(Packet *p,DATABASE *db)
{
	mail *amail=mail::access(p);
	DATABASE *tmp=db;
	while(tmp->next)
	{
		if(!strcmp((tmp->next)->from,amail->from))//same
			return tmp;
		else tmp=tmp->next;
	}
	return tmp;
}

bool FILTERINGAgent::lookupdb(Packet *p)
{
	mail *amail=mail::access(p);
	DATABASE *tmp=db;
```

```
        while(tmp->next)
        {       tmp=tmp->next;
                if(!strcmp(tmp->from,amail->from))//same
                {       if(tmp->count>=THRESHOLD)
                                return TRUE;
                        else
                                return FALSE;
                }
        }
        return FALSE;
}
bool FILTERINGAgent::lookupipdb(Packet *p)
{
        IPDATABASE *tmp=ipdb;
        hdr_ip* hip = hdr_ip::access(p);
        while(tmp->next)
        {       tmp=tmp->next;
                if(tmp->from==hip->saddr())//same
                {       if(tmp->count>=THRESHOLDIP)
                                return TRUE;
                        else
                                return FALSE;
                }
        }
        return FALSE;
}
```

#smtp-agent.h

```
#ifndef _SMTP_AGENT_H
#define _SMTP_AGENT_H

#include <stdio.h>

#include "agent.h"
#define THRESHOLD 3
#include "dmail.h"

struct DATABASE {
        //uint8_t count;
        int count;
        char from[20];
        //int bl_list;
        DATABASE *next;
};
struct IPDATABASE {
        int from;
        int count;
        //int ip_bl_list;
        IPDATABASE *next;
};
struct MAILDATABASE {
        char to[20];
        MAILDATABASE *next;
};

class SMTPAgent : public Agent {
```

```
    public:
        SMTPAgent();
        int command(int, const char * const *);
        void recv(Packet *, Handler *);
        DATABASE *db;
        IPDATABASE *ipdb;
        MAILDATABASE *mdb;
    //protected:
        int bl_list_;

    private:
        void insert_dbase(Packet *);
        void delete_dbase(Packet *);
        void insert_IPdbase(Packet *);
        void delete_IPdbase(Packet *);
        DATABASE *lookup(Packet *,DATABASE * );
        IPDATABASE *lookup(Packet *,IPDATABASE *);
        bool lookup(Packet *);
        void recv_filter(Packet *);
        void recv_smtp(Packet *);
      void show();
};

#endif /* _SMTP_AGENT_H */
```

#smtp-agent.cc

```
#include <stdio.h>
#include <string.h>
#include <fstream.h>
#include "ip.h"
#include "tcp.h"
#include "smtp-agent.h"
static int k,n;
static class MAILHeaderClass : public PacketHeaderClass {
    public:
        MAILHeaderClass() : PacketHeaderClass("PacketHeader/MAIL",
sizeof(mail))
        {    bind_offset(&mail::offset_);  }
} class_mailhdr;

static class SMTPAgentClass : public TclClass {
    public:
        SMTPAgentClass() : TclClass("Agent/SMTP") {}
        TclObject* create(int, const char * const *) {
            return (new SMTPAgent());
        }
} class_smtpagent;

SMTPAgent::SMTPAgent(): Agent(PT_DMAIL)
{

    //bind(" bl_list_ ", &bl_list_ );
    db= new DATABASE;
```

```
    ipdb= new IPDATABASE;
    mdb = new MAILDATABASE;
    db->next=NULL;
    mdb->next=NULL;
    ipdb->next=NULL;
    db->count=0;
    ipdb->count=0;

}

int SMTPAgent::command(int argc, const char*const* argv)
{
    Tcl &tcl=Tcl::instance();
    return Agent::command(argc, argv);
}

void SMTPAgent::recv(Packet *p, Handler *)
{     n++;
    mail *amail=mail::access(p);
      hdr_ip* hip = hdr_ip::access(p);
      ofstream fileop("smtp1.txt", ios::out | ios::ate | ios::app);
      fileop<<amail->to<<"  "<<amail->from<<n<<" --- "<<hip-
>saddr()<<"\n";
      fileop.close();
    if(amail==NULL)
    {
       printf("PANIC: mail is NULL\n");
       abort();
    }
    if(amail->from_smtp==1)

      recv_smtp(p);

    else
      {recv_filter(p);
            }
}
void SMTPAgent::recv_filter(Packet *p)
{     static int a;
      mail *amail=mail::access(p);
      DATABASE *tmpdb ;
      IPDATABASE *tmpipdb ;
      bl_list_=0;
      hdr_ip* hip = hdr_ip::access(p);
                        //hip->daddr()=hip->saddr();
                        //printf("smtp dest = %d",hip->daddr());
                        //printf("smtp source = %d",hip->saddr());
      if(!lookup(p))
      {     a++;
            ofstream fileop("smtp.txt", ios::out | ios::ate |
ios::app);
            fileop<<amail->from<<"    "<<amail->to<<" --- "<<hip-
>saddr()<<"\n";
            fileop.close();

                  amail->from_smtp=1;
                  hdr_ip* hip = hdr_ip::access(p);
```

```
                hip->daddr()=1;
                send(p,0);
        }

        //show();
}
void SMTPAgent::show()
{       IPDATABASE *tmp = ipdb;
        ofstream fileop("smtplist.txt", ios::out| ios::ate | ios::app);
        while(tmp->next){
                tmp=tmp->next;

                fileop<<tmp->from<<" "<<tmp->count<<"\t";

                //printf("%s  ",tmp->from);
                //printf("%d \n",tmp->count);
        }
        fileop<<"\n";
        fileop.close();
}
void SMTPAgent::insert_dbase(Packet *p)
{
        DATABASE *tmp = new DATABASE;
        mail *amail=mail::access(p);
        strcpy(tmp->from,amail->from);
        tmp->count=1;
        tmp->next=db->next;
        db->next=tmp;

}
void SMTPAgent::insert_IPdbase(Packet *p)
{
        IPDATABASE *tmp1 = new IPDATABASE;
        hdr_ip* hip = hdr_ip::access(p);
        tmp1->count=1;
        tmp1->from=hip->saddr();
        tmp1->next=ipdb->next;
        ipdb->next=tmp1;
}
DATABASE* SMTPAgent::lookup(Packet *p,DATABASE *db)
{
        mail *amail=mail::access(p);
        DATABASE *tmp=db;
        while(tmp->next)
        {
                if(!strcmp((tmp->next)->from,amail->from))//same
                        return tmp;
                else tmp=tmp->next;
        }
        return tmp;
}
IPDATABASE* SMTPAgent::lookup(Packet *p,IPDATABASE *ipdb)
{
        hdr_ip* hip = hdr_ip::access(p);
        IPDATABASE *tmp1=ipdb;
        while(tmp1->next)
        {
```

```
            if(tmp1->next->from==hip->saddr())//same
                  return tmp1;
            else
                  tmp1=tmp1->next;
      }
      return tmp1;
}

void SMTPAgent::recv_smtp(Packet *p)
{
//    mail *mail=mail::access(p);
      DATABASE *tmpdb;
      IPDATABASE *tmpipdb;
      tmpdb=lookup(p,db);
      if(tmpdb->next!=NULL)
            tmpdb->count++;
      else
            insert_dbase(p);
      tmpipdb=lookup(p,ipdb);
      if(tmpipdb->next!=NULL)
            tmpipdb->count++;
      else
            insert_IPdbase(p);

}
bool SMTPAgent::lookup(Packet *p)
{
      mail *amail=mail::access(p);
      char b[10];
      int k=6;
      ifstream fileop ("user.txt", ios::in);
//    fileop.getline(b,10);
      while(k)
      {     fileop.getline(b,10);
            k--;
            if(!strcmp(b,amail->to))//same
                  return TRUE;
      }
      return FALSE;

}
```

#dmail.tcl

```
#Create a simulator object
set ns [new Simulator]
$ns color 1 Red
$ns color 2 Blue

#Open the nam trace file
set nf [open ne.nam w]
$ns namtrace-all $nf



#Define a 'finish' procedure
proc finish {} {
```

```
        global ns nf
        $ns flush-trace
      #Close the trace file
        close $nf
      #Execute nam on the trace file
        exec nam ne.nam &
        exit 0
}

#create 7 nodes
for {set i 0} {$i<4} {incr i} {
      set m($i) [$ns node]
}
#normal nodes
for {set i 100} {$i<105} {incr i} {
      set m($i) [$ns node]
}
#attacker nodes
for {set i 200} {$i<203} {incr i} {
      set m($i) [$ns node]
}

#Create links between the nodes
$ns duplex-link $m(200) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(201) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(202) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(100) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(101) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(102) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(103) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(104) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(0) $m(1) 5Mb 10ms DropTail

#$ns duplex-link $m(1) $m(2) 5Mb 10ms DropTail
#$ns duplex-link $m(2) $m(0) 5Mb 10ms DropTail
$ns duplex-link $m(1) $m(3) 5Mb 10ms DropTail
#$ns duplex-link $m(1) $m(4) 5Mb 10ms DropTail
#$ns duplex-link $m(1) $m(5) 5Mb 10ms DropTail
#$ns duplex-link $m(1) $m(6) 5Mb 10ms DropTail
#$ns duplex-link $m(3) $m(4) 5Mb 10ms DropTail
#$ns duplex-link $m(3) $m(5) 5Mb 10ms DropTail
#$ns duplex-link $m(3) $m(6) 5Mb 10ms DropTail
#$ns duplex-link $m(4) $m(5) 5Mb 10ms DropTail
#$ns duplex-link $m(4) $m(6) 5Mb 10ms DropTail
#$ns duplex-link $m(5) $m(6) 5Mb 10ms DropTail

#Class Application/TVAUser -superclass Application

set rng [ new RNG ]
      set filter [ new Agent/FILTER ]
            $ns attach-agent $m(1) $filter
            #set tcp [ new Agent/TCP ]
            #$ns attach-agent $m(2) $tcp

            set smtp3 [ new Agent/SMTP ]
            $ns attach-agent $m(3) $smtp3
            #set smtp4 [ new Agent/SMTP ]
```

```
            #$ns attach-agent $m(4) $smtp4
            #set smtp5 [ new Agent/SMTP ]
            #$ns attach-agent $m(5) $smtp5
            #set smtp6 [ new Agent/SMTP ]
            #$ns attach-agent $m(6) $smtp6
for { set j 1} { $j <= 20 } { incr j } {
   for { set i 100 } { $i < 105 } { incr i } {
            #global rng ns
            #$ns instvar rng
            set magent [new Agent/MAIL]
            $ns attach-agent $m($i) $magent
            #set app1 [new Application/Traffic/CBR]
            #$app1 set packetSize_ 500
            #$app1 set interval_ 0.005
            #$app1 attach-agent $magent
            $ns connect $magent $filter
            $ns connect $filter $smtp3
            #$ns connect $magent $smtp3
            #$ns at [ expr 0.1 + [ $rng uniform 0 1 ] ] "$app1 start"


            #set a($i) [ expr $i-100 + [ $rng uniform 0 1 ] ]
            puts $i
            #$ns at 1.0 "$magent send"
            $ns at [ expr 2.0 + [ $rng uniform 0 1 ] ] "$magent send"
            }
      }
proc getnodeid {} {
global ns m filter
$filter set nodeid_ [$m(3) node-addr]
}
#for { set i 100 } { $i <= 102 } { incr i }
#$ns at $a($i) "$magent send"
  #for { set i 200 } { $i <= 202 } { incr i } {
#                 set magent [ new Agent/MAIL]
#            $ns attach-agent $m($i) $magent
            #set app1 [new Application/Traffic/CBR]
            #$app1 set packetSize_ 500
            #$app1 set interval_ 0.005
            #$app1 attach-agent $magent
#           $ns connect $magent $filter
            #$ns connect $filter $smtp3
#$ns connect $magent $smtp3
            #$ns at [ expr 0.1 + [ $rng uniform 0 1 ] ] "$app1 start"
#           $ns at 3.0 "$magent send"
            #$ns at [ expr 0.2 + [ $rng uniform 0 1 ] ] "$magent send"
#   }

    $ns run
#}

#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"


#Run the simulation
#flow-create
```