Pre-MX Spam Filtering with Adaptive Greylisting Based on Retry Patterns

Bachelor Thesis

by

Peter Lieven

from

Düsseldorf

submitted to

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

August 2006

Advisors:
Dipl.-Inf. Björn Scheuermann
Dipl.-Inf. Michael Stini

Acknowledgments

First of all I would like to send out a big thank you to all the people who supported me during one of the worst periods of my life. Especially but not only to Matthias Dellweg, Marion Heitmann, Lena Jansen, Patrick Kambach, Richard Lohkamp, Anika Pahl and Dorothee Recke—without you I would not have managed to complete this work. Thanks for listening to my problems, your patience, your consolation and the right word at the right time!

Furthermore, I would like to thank my parents, Wilhelm and Renate Lieven, for their continuous support and their assistance.

Last but not least I would like to express my gratitude to the people from the Department of Computer Science at the University of Düsseldorf for their support not only during the writing of this thesis—particulary Björn Scheuermann and Michael Stini for their advice and understanding.

I am indebted to the staff of the companies MESH-Solutions GmbH, Kambach.Net and DLH.Net for allowing me to test my reference filter implementation on their productive mailservers, and their feedback during the evaluation period.

Contents

List of Figures

List of Tables

List of Abbreviations

CRS	Challenge- and Response System
CSR	Consecutive Short Retry
DDoS	Distributed Denial of Service
DoS	Denial of Service
DNS	Domain Name System
DNSBL	DNS-based Blackhole List
DNSWL	DNS-based Whitehole List
ERT	Expected Retry Time
IDS	Intrustion Dectection System
MR	Mail Relay
MTA	Mail Transfer Agent
MX	Mail Exchanger
PTR	Pointer Resource Record
RBL	Real-time Blackhole List
RMX	Reverse Mail Exchanger
SMTP	Simple Mail Transfer Protocol
SPF	Sender Policy Framework
SVM	Support Vector Machine

Chapter 1

Introduction

Communicating via e-mail is one of the biggest changes in communication behaviour within the last 20 years. It has become fundamental to the interaction between human beings in business, education and private life. It is easy, convenient, cheap and most people do not want to miss its comfort nowadays. However, the cheapness is one of the biggest disadvantages in paperless communication. Everybody can send one another e-mails—like it is with snail mail—but the fact that it costs almost nothing has led to a problem that many users know well: spamming [Wike, Tem]. They receive dozens of unwanted e-mails and they are unable to stop the flow. Many users receive four times more spam than legitimate e-mail [Mes06]. Spam can therefore be considered as one of the big diseases in the information technology of the 21st century.

1.1 History of SMTP

The original Simple Mail Transfer Protocol (SMTP) [Pos82], the standard for transporting e-mail over the Internet, dates back to 1982, a time when nobody could foresee in what environment the protocol would be used later. SMTP was designed to be able to communicate simply on a text based console, and originally it was intended to be spoken by humans in times when there were no real mail clients. In the beginning SMTP was used to communicate within a small user community and especially there was no thought ahead of someone sending another unwanted e-mails. Basically every SMTP server accepted mail from everyone and even relayed it to another server if the mail recipient

was not local. While the problem of an open mail relay [Wikc] is really easy to fix by user authentification or IP-based access lists, the problem of accepting local mail from everyone is the root of all spam. Easily said SMTP lacks a standard mechanism to verify whether the sending server and the sender address match. There are approaches like Sender Policy Framework (SPF) [WS06], Reverse MX (RMX) [Dan04] or Microsoft's SenderID [LW06] to fix this hole, but as long as there is no unified standard that everyone has to support, a Mail Exchanger (MX) still has to accept mail from sending domains not supporting this way of authentification. For example one could pretend to be evil@hell.com and there is no standard way to check if he really is.

1.2 Problem Statement

Although the design of SMTP is the origin of the whole spam case, it would be impossible to just replace the protocol at once to get rid of the spam problem. So the only options left are legal force and intercepting the mail delivery process at the receiver's side.

But even if today spamming is considered illegal in many countries [Alo04], spammers manage to hide their identity very clearly and remain undetected in most cases. Therefore, stopping spam by law alone is no effective option [Fed05]. This leads to the trouble that the only real mechanism to get rid of unwanted e-mails is to filter them technologically just before they enter a user's mailbox.

Most of today's so-called *spam filters* look at mail headers and bodies and classify them by pattern or probability analysis. This approach usually works very well for plain-text spam or mass-mailings, but has generally some big disadvantages.

Since content filters analyse the structure of a message, spam has to be already received by the Mail Exchanger (MX). Only after reception they are able to classify the mail by often very complex techniques. This late point of classification makes filtering spam very resource consuming.

Furthermore, these filters should work automatically and so they can only be as good as the kind of spam they were designed for. But spammers adapt to cheat the filters

and this leads to a never-ending competition between the cleverness of spammers and the ability of filters to detect them. The newest development is a heavy increase of so-called *image spam*—spam that only consists of images making the content very hard to analyse [Iro06].

However, not only the kind of spam, even the way the spam is sent, has changed during the last years. Nowadays a huge amount of spam is sent by so-called *Zombie* [Wikg] machines [Iro06]—compromised end-user systems—that today do not only act as distributed denial of service platforms, but also as spam agents.

Zombie machines often use very bad and non RFC-compliant Mail Transfer Agents (MTAs) for delivering their messages. This *ratware* [Wikd] behaves very resource unfriendly and is often unable to do proper message queueing and delivery timing.

The decreased detection rate caused by hard to analyse and rapidly changing message content and especially the fingerprints that zombies left in our mail server's logfiles led to the idea to analyse the behaviour of sending mail servers at connection level and trying to find characteristics to be able to filter them at this very early, resource-saving stage of the delivery process.

The vision is to build a filter that is completely independent of spam content and just relies on the fact that spammers usually use stolen resources and are forced to send a huge amount of spam in a small amount of time using carelessly implemented MTAs.

1.3 Contribution

In this thesis we analyse the reaction of sending mailservers on refused connections and their timing strategies on delivery reattempts. Derived from the observations and analysis of collected retry behaviour, we will introduce a filter idea that takes the timing between consecutive delivery retries of a sender into account. This filter idea is then implemented in a reference framework and put in production environment where its real life usability is tested and evaluated. Since all necessary information is collected at connection level the invented mechanism is able to join the mail delivery process before the mail is actually accepted by the MX. The introduced filter acts like a proxy forwarding only those con-

nections to the real MX that belong to senders that manage to behave resource-friendly while refusing their first connection attempts.

1.4 Structure

At first we give a short overview of other connection level spam filtering mechanisms and some basic requirements in Chapter 2. In Chapter 3 we describe how we initially gathered information about the retry behaviour, observations made and how these oberservations were incorporated in a filter model. Chapter 4 sketches our reference filter implementation. Then we give some performance analysis in Chapter 5 and follow with our conclusion and an outlook on further work in Chapter 6.

Chapter 2

Prerequisites

In this chapter we first give an overview of origin-based filter systems that can also be implemented at connection level. Then we discuss greylisting [Har03] and the disadvantages of its current implementations. At last we introduce the difference between Mail Exchanger and Mail Relay for a better understanding where a pre-MX filter has to be installed.

2.1 Origin-based Filter Techniques

Origin-based filter techniques are the lowest level spam filtering mechanisms currently available. They join the mail delivery process at a very early stage and are therefore very lightweight regarding resource usage. In this Section we give a short overview of most widespread mechanisms.

2.1.1 Black- and Whitelisting

The simplest way to filter mail is to have static filters based on the originating IP. For example, one could maintain a blacklist and add all IP addresses that have previously sent unwanted e-mail or which no communication is desired with. Then the mailserver would simply not accept mail from those hosts in the future. But normally it is not convenient to blacklist IPs manually and so this method is used only under very rare

circumstances. The general behaviour can be compared to a firewall allowing everyone not explicitly denied.

More widely used are whitelists among closed user groups where one exactly knows whom he wants to accept mail from. This is the opposite of blacklisting—with whitelisting everything not explicitely allowed is denied. Whitelists may also be used to exclude mail from certain IPs to be not classified as spam by higher level filters. Especially one would normally whitelist all private and local IP addresses.

2.1.2 DNS-based Blackhole Lists (DNSBL)

A more sophisticated way of blacklists are so-called DNS-based Blackhole Lists (DNSBLs) [Wika]—also known as Realtime Blackhole Lists (RBLs). These lists are generated dynamically and are based on the Domain Name System (DNS) [Pos84]. The DNS is supported everywhere, is hierarchical, allows caching with timeouts and is resource-friendly for a blacklist provider. It can be used instantly for blacklisting purposes without modifying the whole infrastructure. Therefore it is mostly preferred although other mechanisms like regularly fetching blacklists via HTTP or rsync would be possible. However, one has to trust the DNS which is not necessarily secure [Eis05].

If a DNSBL is configured on an MX on each incoming connection, the sender IP address (a.b.c.d) is reversed and a DNS look-up under a well known domain (e.g. d.c.b.a.mydnsbl.org) is made. If the IP address is being considered evil by the DNSBL provider, the look-up yields a predefined status code. Then the sender is locked out and no mail will be accepted.

DNSBLs are widespread nowadays and if a spammer IP address is preknown, they are very efficient. The problem here is that spammers mostly have a large network of compromised hosts they use for sending mail. As seen in our logfiles, after a new spam rush has been sent, it takes some time (usually 2–4 hours) after the first delivery attempts before the spamming IPs are entered into the DNSBLs—depending on how many spam reports the blacklist provider receives for an IP address. Generally, DNSBLs force spammers to send their mails in a very short timeframe before they are listed and locked out.

2.1.3 Challenge and Response Systems

It is very painful to maintain whitelists and blacklists by hand. A very common way of filling whitelists automatically is the use of Challenge and Response Systems (CRSs) like introduced in TMDA [TMD]. The idea is simple, but very effective. Every still unknown combination of sender address and IP is considered harmful. The corresponding mail is spooled and the sender gets a challenge sent back via e-mail. Usually the challenge is to click a link or to enter a code. Once the system gets a valid response the address/IP-combination is whitelisted, the corresponding mail is released and all future mail will be directly delivered.

This way almost no spam makes it through such a system. But as we already know the sender address of an e-mail may be false or even non-existant so there must be additional filters in front of the CRS to keep the local server's mail queue clean from challenge mails that cannot be delivered correctly. Additionally there is a risk that the e-mail sender does not understand the challenge or simply ignores the mail, so it is almost indispensable to have regular looks at the queue of the CRS and release some mails by hand.

2.1.4 Firewall-Based Filters

Spammers often tend to so-called hammering [Wikb]. Therefore, firewalls often filter new incoming connections and only allow a reconnection from the same IP after a reasonable timeframe. Normally it is a good idea to also limit the overall number of incoming new connections to an MX to protect the system from a denial of service (DoS) attack. There are also approaches to dynamically blacklist hosts hammering the local system in the firewall for a certain amount of time or throttling the allowed maximum bandwidth for resource wasting IPs [Ban05]. But since we want to analyse the retry behaviour and are especially interested in extremely bad behaviour like hammering we have to disable most of the firewall filters and keep only those that would make our system vulnerable to DoS if absent.

2.1.5 Malicious Activity

A very interesting approach is detecting spammers by malicious acitivities like it is done in intrusion detection systems (IDSs) [CHMS06]. This approach relies on the fact that many of the sending hosts are compromised end-user systems (zombies) that try to infect the MX itself before starting to send spam to it. If recording their activity like connecting to port 0 or scanning netbios ports we have a strong hint that they might not be legitimate servers trying to send mail, but have a bad intention. Often they do not even perform a real MX look-up when sending spam, but just connect to mail.mydomain.com when trying to send spam to someone@mydomain.com. Giving the real MX a name distinct from mail.mydomain.com and mapping mail.mydomain.com to a different IP, provides an additional hint that the sender might be bad.

2.2 False Positives and False Negatives

When talking about spam filters and their effectiveness it is important to know how to measure their efficiency. A good indicator is the amount of *false positives*—legitimate mails that are misclassified as spam by a spam filter and consequently blocked out. Analogical *false negatives* are those spam mails not detected by the filter and let through. A spam filter providing less than one percent of false positives is generally considered exceptionally good. Of course, false positives and negatives influence each other—a filter generating less false positives is likely to generate more false negatives and vice versa.

2.3 Greylisting

One of the latest inventions in early stage spam filtering is greylisting [Har03]. Greylisting uses the ability of the SMTP protocol to deny an incoming connection with a temporary error. According to the SMTP RFC the sending server should spool that mail and try again after a reasonable amount of time. Normal greylisting as used today uses a triple of sender address, sender IP and destination address. This triple is stored in a database at

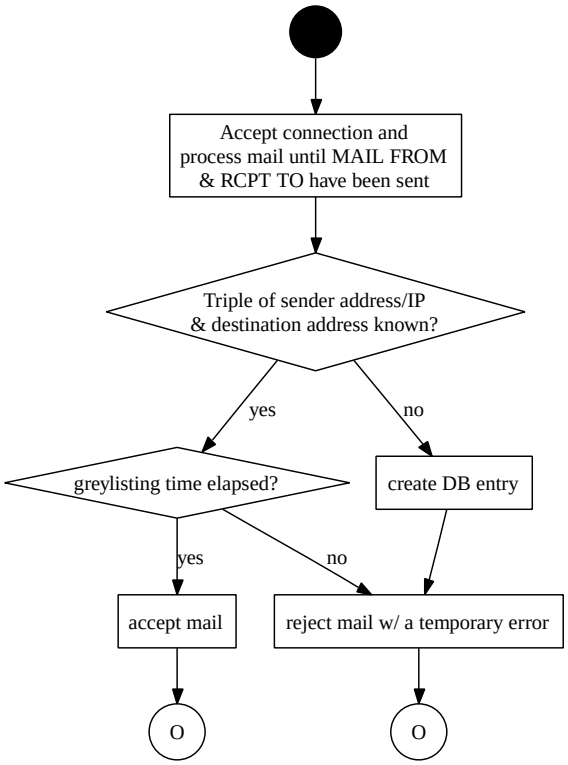

Figure 2.1: Flowchart of Greylisting Process

the first occurance together with a timestamp. Starting from that time every connection matching is rejected with a temporary error for a fixed amount of time. After this fixed time has elapsed each mail with that triple is accepted. Figure 2.1 reveals how greylisting is implemented in most cases.

Greylisting works under the assumption that spammers often do no queueing or use a different sender address when trying the next time. Surprisingly greylisting performs very well at the moment and is able to filter up to 90 percent of incomming spam connections while generating nearly zero false positives [Sle04, Lev05].

2.4 How to Enhance Greylisting

Ordinary greylisting uses *fixed* timers to keep hosts out. The timer value used is generally a trade-off between keeping a sender out as long as possible to get a chance that DNSBLs or high-level signature filters have been updated in the meantime while not

disturbing communication between users for too long. Usually a timeout value between ten to 60 minutes is used regardless of how many delivery attempts have been made by the sender in the meantime. This algorithm is very simple, but a lot of useful information is lost. Normally one would expect the sending host to try again later after a reasonable timeframe. The SMTP RFC [Kle01] suggests to try not more than two times an hour, so normally one would expect at least 15–20 minutes between two consecutive retries. This timer has to be maintained for each target MX—not for each single mail.

The temporary error status code returned during greylisting means that the MX has a temporary problem and cannot accept e-mail. It is impolite to try again too soon or just ignore the error completely and reconnect immediately. So if we evaluate the behaviour while a sender is locked out we can get a first impression what his intention might be. A legitimate sender would be interested in saving its and our resources and wait while a spammer tries to deliver its mails as fast as possible not caring about resources.

Considering the described trade-off the basic idea of what we call *adaptive greylisting* is to let the good ones in as fast as possible while keeping the misbehaving out as long as possible. So even if we delay transmissions for a legitimate host not caring about the local server nobody would normally complain about that sender being ignored for a while.

Sadly the amount of information lost in normal greylisting is even worse. As explained greylisting uses triples and not the same information for each sender IP. This is a good idea if we think of altering sender/destination address combinations, but if we want to evaluate the behaviour of a host we have to evaluate the host detached from sender and destination addresses. As explained before the sender address may be random and the destination address is local otherwise the sending host would not have contacted the local MX.

If there is more than one mail delivery attempt directed at the same MX running at the same time we can even gather more information about correct behaviour. A target MX with many mailboxes is likely to have more than one mailbox the same spamming host is sending mail to. If the MTA on that host does not care about a per desitination MX retry timer we have an even better chance to identify it.

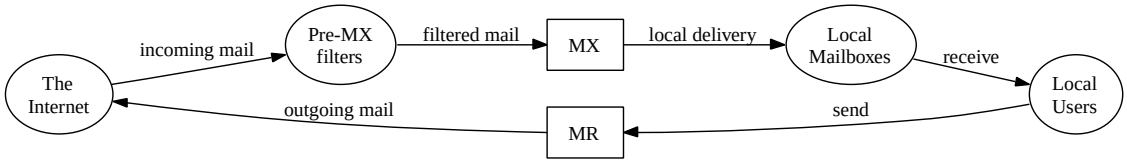

Figure 2.2: Difference between Mail Exchanger and Mail Relay

Furthermore, greylisting is bad from the resource point of view because an MX has to accept an incoming connection and process SMTP messages before information about sender and destination address can be extracted. If one considers only the IP, it is possible to refuse the connection with an error without first interacting with the sender. In business greylisting is considered harmful by some people because it disturbs every interaction between two users communicating for the first time. Especially if salesmen wait for an important e-mail from a customer they are unhappy to wait for it. Considering that we classify the sending server behaviour one will easily agree on the fact that a host that has previously shown that he is able to do proper queueing and act resource-friendly will likely do so for future e-mails as well.

Normally if a sender has managed to bypass greylisting he will also manage to bypass in the future. If we consider all hosts that previously have shown good behaviour as whitelisted for some time a state where most good servers are whitelisted will be reached and new communication between unknown sender/destination addresses is not necessarily delayed. We even circumvent the problem that some MTA systems use altering sender addresses for each delivery attempt which is fatal for triple-based greylisting.

2.5 Mail Exchanger and Mail Relay

When analysing the retry behaviour it is very important to understand that there is a great difference between what is called a Mail Exchanger (MX) and a Mail Relay (MR) and where a pre-MX filter has to be installed (see Figure 2.2).

An MX acts as receiver for local mail from the Internet. For each domain one or more MXs with optionally different priorities are defined in the DNS zone for that domain. While an MX per definition has to accept incoming mail for local domains from every-

one a Mail Relay does not. The MR in today's Internet is the sending MTA for local users wanting to send mail to a foreign system. As most users do not have a permanent connection to the Internet the MR accepts mail from its legitimate users and does the further mail processing including spooling and retransmission if necessary. This task is performed in the background and the user does not need to care about his e-mail after having passed it on to the MR. In order to check if a user is legitimate he has to authenticate via password or has to be in a given IP range when contacting the MR. If such an authentication is missing the relay is considered open [Wikc].

Generally the task of an MX and an MR is performed by the same process. If the destination address is local, the mail is delivered to a local mailbox, in other cases the appropiate MX for the destination address is looked up and the mail is then relayed to that server. If we want to install a filter that protects us from incoming foreign mail, we have to install this filter in front of the MX and not the MR. Otherwise we would involve users wanting to contact the MR in the filtering process.

The easiest way to achieve this is to have a local mailing software like exim [exi], qmail [qma] or postfix [pos] running on two different IP addresses. The process on one IP is acting as MR with no further prefiltering attached. The other process is then acting as MX with optional filters in the firewall or a wrapper/proxy before it.

Basically the mailing software is not even required to know this difference. It would be enough to set up two host entries, e.g. mx.mydomain.com and relay.mydomain.com and map them to the two different IP addresses.

Chapter 3

MX Retry Behaviour

The first step in retry pattern analysis is the gathering of appropriate test data. In this chapter we describe how we initially sampled data, which observations we made on the pattern collected and how we tried to incorporate all this in a filter model.

All data collection and testing was done on the mail server of DLH.Net [dlh]. This server contains about 100 mailboxes, most of them existing for almost ten years. Long existing mail addresses are a very good spam magnet since the addresses have often been carelessly posted in newsgroups or appeared in plain-text on webpages making them very easy to spider.

The MX for incoming mail was set to *mx.dlh.net* and the MR was configured to be *authrelay.dlh.net*. With this configuration there should be no legitimate SMTP connection to *mail.dlh.net* itself.

3.1 Collecting Data

The SMTP protocol allows to delay a mail delivery with a temporary error. A short summary of SMTP response codes used can be found in Table 3.1. Generally error code *451* is used in normal greylisting. However, since we refuse the connection at the initial connection we have to use *421* instead to stay RFC-compliant. After refusing a

Table 3.1: Common SMTP Error Codes

Code	Description	Allowed after
220	Service ready	conn. establishment
421	Service not available, closing transmission channel	anywhere
451	Requested action aborted: local error in processing	anywhere after MAIL
550	Requested action not taken: mailbox unavailable	anywhere after HELO

connection with a temporary error it is the task of the sending server to spool the mail and retry the transmission later.

To record comparable data we used a mechanism similar to normal greylisting. A simple TCP wrapper was written that set up a database entry at the first occurrence of a new IP address along with a timestamp. This first and each consecutive retry was rejected with a temporary error for the first two hours after the initial occurrence. After this time each incoming connection was permitted regardless of being evil. Within the first two hours each connection attempt was logged with exact timing information (see Figure 3.1).

Addtionally we set up a fake secondary MX with lower priority which always denied incoming connections. Connection attempts to this host were also logged. Referring to the SMTP RFC MXs have to be contacted in the order of their priority. This facility is useful to check if an MTA contacts the MXs for a domain in correct order.

At last we set up a third logging mechanism logging all connection attempts to any ports except TCP port 25 (STMP) and port 179 (ident). We redirected mail.dlh.net to an unused IP and logged all connection attempts there as well. This logging facility was intended to gather information if a sender does a valid MX look-up and/or was trying to scan or even compromise our own system before sending mails to it. This idea was partly inspired by [CHMS06].

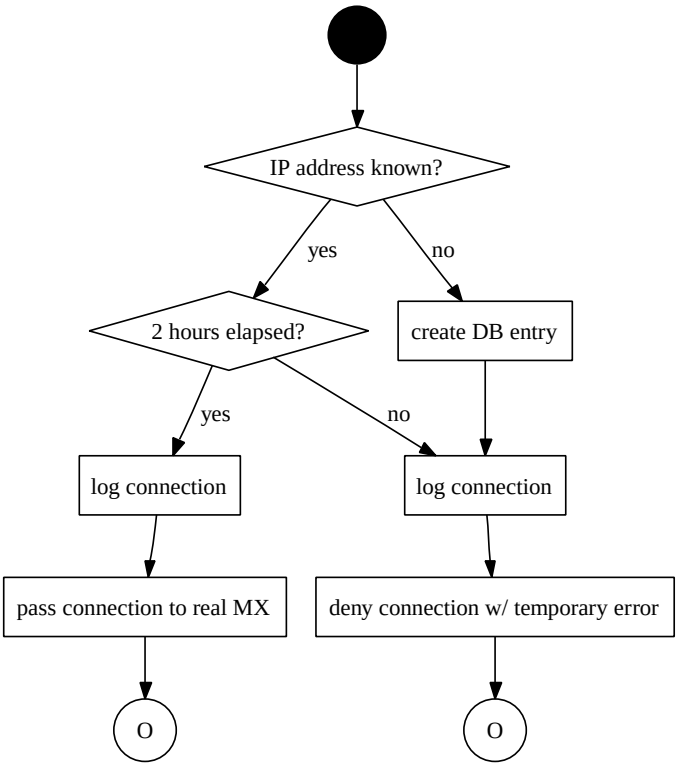

Figure 3.1: Data Collector for Incoming SMTP Connections

3.2 Evaluation

The data collection phase ran over four weeks between May 5th and June 4th 2006. During this period information about the retry behaviour of about 130,000 distinct hosts was collected.

On the one hand, to distinguish between good and bad patterns we decided to consider all IP addresses that were blacklisted by common DNSBLs or were originated from a dynamic dial-up host (likely to be a zombie) as bad. Different from normal DNSBL usage we had the big advantage that we were able to tag a pattern as bad long after it was recorded. This way the general delay for an IP being listed in a blacklist did not matter.

As our records revealed not a single host not performing a real MX look-up—this means connecting to the SMTP port on mail.dlh.net instead of mx.dlh.net—ever passing the installed CRS we decided to consider all those hosts bad as well.

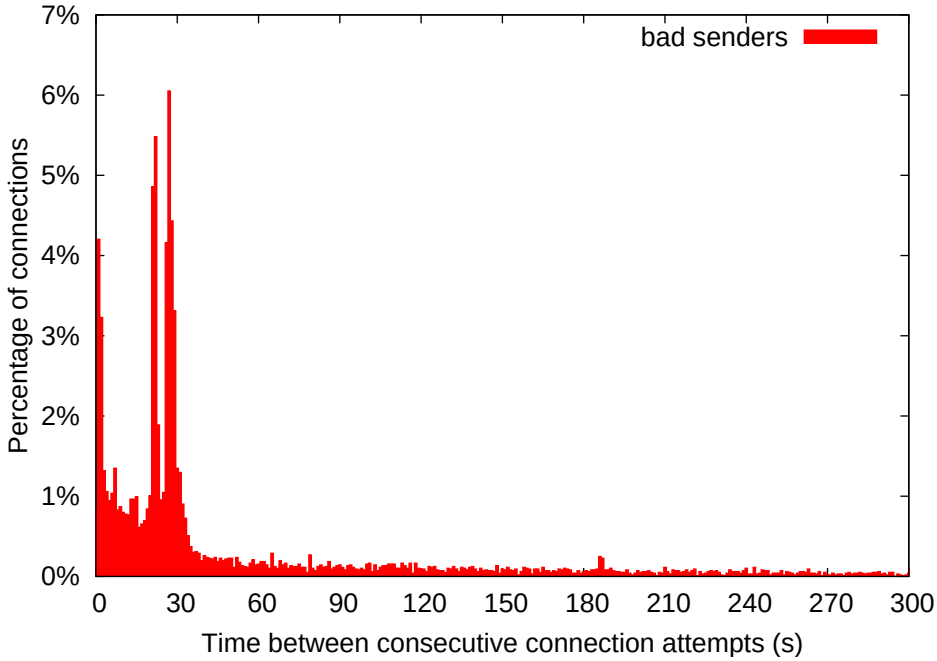

Figure 3.2: Distribution of Retry Times for all Recorded Bad Senders

On the other hand, a pattern was marked good if it belonged to a host that successfully passed the CRS. Using these criteria we had a good indication for definitely bad and most likely good patterns.

The histogram in Figure 3.2 shows the distribution of retry times up to five minutes for all bad MTAs recorded during the data collection phase. There is an accumulation of very short retry times below 30 seconds and a cluster of retries below 1 second.

Figure 3.3 reveals the distribution of retry times up to half an hour for all good senders. They will likely reconnect after fixed intervals building clusters at characteristic times like 60, 600 or 900 seconds. However, even some good senders seem to have broken SMTP error code handling and reconnect immediately after a temporary error.

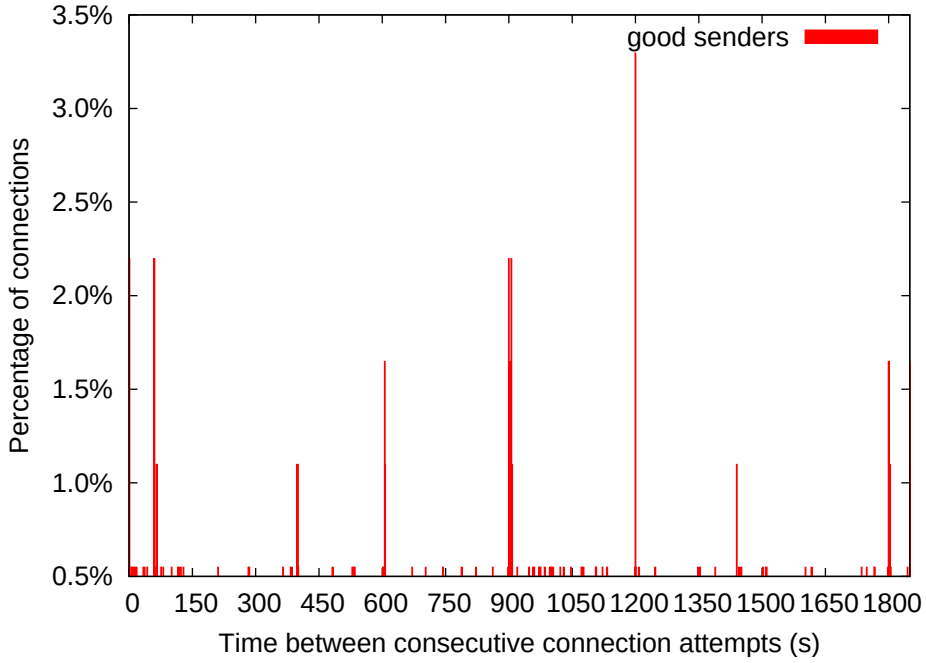

Figure 3.3: Distribution of Retry Times for all Recorded Good Senders

3.3 Basic Filter Idea

Derived from the patterns that were gathered during the initial data collection phase we decided to build a filter based on a penalty system. The *penalty* is a timeframe in seconds after the initial connection we definitely block a sender out. This effects that for a connection to be permitted the whole penalty time has to be elapsed first. During the penalty phase all connections will be refused with a temporary error while, of course, every attempt during that time might cause additional penalty.

Inspired by the fact that most malicious senders have a very short delay between two consecutive connection attempts (short retry) we decided to define an expected retry time between two connection attempts and punish all too short retries with increasing the penalty. To judge consecutive bad behaviour a counter named *consecutive short retries* (CSR) was integrated which is increased for each short retry and decreased for each connection retry time that was longer than the expected retry time. How the penalty is increased can be seen in (3.1).

```
penalty += ( expected_retry_time - real_retry_time ) * ++consecutive_short_retries
```
(3.1)

Table 3.2: Example Penalty System

Action	Penalty increase
Initial penalty at first MX connect	900s
Malicious activity (portscan/fakeMX)	+3h
No PTR set or DNS server failure	+6h
Secondary MX contacted before primary	+3h
Retry time below one second	+2h
Retry time below five seconds	+1800s
(Consecutive) short retry	see (3.1)

If a sender does not care about our error code handling at all and reconnects immediately after a refused connection (hammering) an additional penalty increase is made to honour this *extremely bad* behaviour.

As discussed earlier we also want to be able to take malicious activity, wrong MX priority handling or fake MX look-ups into account. An overview of a complete example penalty system is shown in Table 3.2 . Most values in this penalty system are based on experience and are more or less arbitrary.

3.4 Expected Retry Time

The most essential value in our filter is the expected retry time (ERT). Derived from this value all dynamic penalty increase decisions are made. As the RFC proposed value of 15-20 minutes between consecutive connection attempts would be inappropiate even for most friendly mailservers we decided to use the data gathered during the inital data collection phase and simulate the inital delay of good senders while altering the value chosen for the ERT.

Figure 3.4 shows the worst case, the average case for the penalty all good senders would get and the average delay a good message would have been delayed if the respecting value for the ERT would have been used in the filter.

Figure 3.5 represents the equivalent simulation with the Y-Axis being linear and without maximum delay focussed on the ETR interval between 0 and 600 seconds.

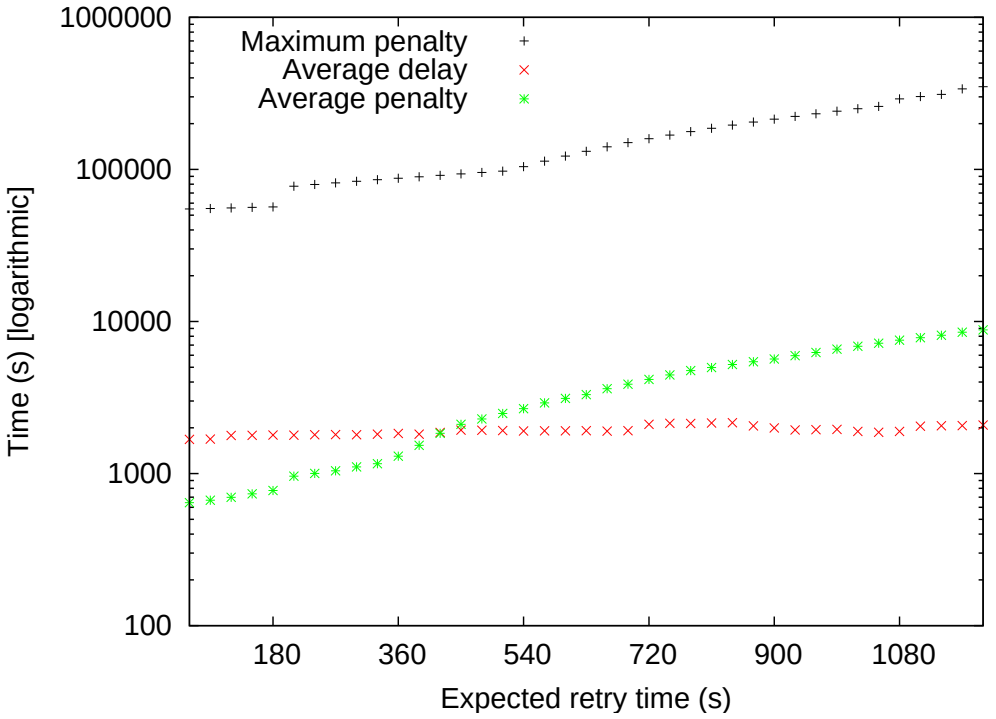

Figure 3.4: Filter Retry Time Simulation

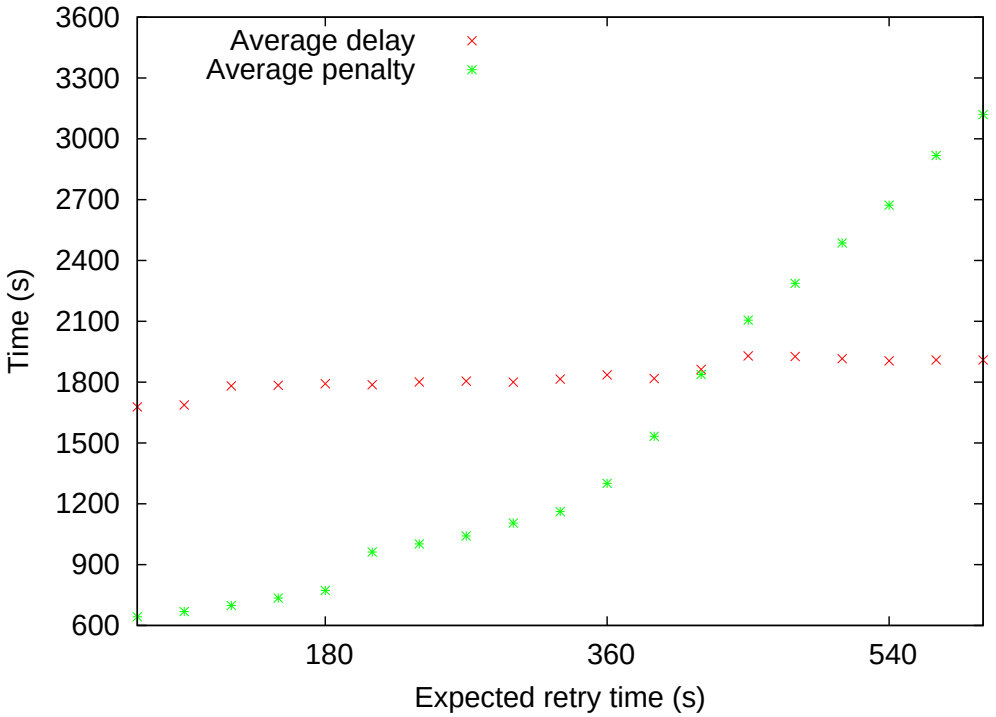

Figure 3.5: Filter Retry Time Simulation (detailed)

Table 3.3: Exemplary Penalty Calculation for *rohrpostix.cs.uni-duesseldorf.de*

Try	t	△t	CSR	△Penalty	Total penalty	Action
#1	0s	-	0	900s	900s	deny
mx2	5s	-	-	-	900s	deny
#2	1389s	1389s	0	-	900s	permit

Table 3.4: Exemplary Penalty Calculation for *mail.gmx.net*

Try	t	△t	CSR	△Penalty	Total penalty	Action
#1	0s	-	0	900s	900s	deny
#2	400s	400s	0	-	900s	deny
#3	1200s	800s	0	-	900s	permit

Derived from the result that there is a significant gap between the calculated penalty with an ERT of 180 and 210 seconds we decided to use an ERT of 180 seconds in our later tests and implementation.

3.5 Exemplary Penalty Calculation

In this section we give some easy examples how penalty calculation actually works.

Table 3.3 shows the retry behaviour of our institute's mailserver and the penalties raised. This mailserver does correctly contact the MXs in order of their priority.

Table 3.4 reveals the retry pattern of the MX of one of Germany's biggest free-mail providers. This server does not contact the secondary MX at all, but has a nice timing behaviour.

As penalty calculation for bad senders usually causes many log entries we will give only a short example how characteristic the behaviour of a ratware MTAs is. Table 3.5 clearly shows incorrect MX priority handling, chaotic timing and port scans (ids) at once. The table has been abbreviated since the original log had more than 300 hundred lines.

Table 3.5: Exemplary Penalty Calculation for *p54830063.dip0.t-ipconnect.de*

Try	t	△t	CSR	△Penalty	Total penalty	Action
mx2	-5s	-	-	10800s	10800s	deny
#1	0s	-	-	900s	11700s	deny
ids	22s	-	-	10800s	22500s	-
ids	22s	-	-	10800s	33300s	-
#2	22s	22s	1	158s	33458s	deny
mx2	391s	-	-	-	33458s	deny
#3	396s	374s	0	0s	33458s	deny
ids	417s	-	-	10800s	44258s	-
#4	417s	21s	1	159s	44417s	deny
mx2	481s	-	-	-	44417s	deny
#5	486s	69s	2	222s	44639s	deny
#6	507s	21s	3	477s	45116s	deny
ids	508s	-	-	10800s	55916s	-
ids	508s	-	-	10800s	66716s	-
mx2	523s	-	-	-	66716s	deny
#7	528s	21s	4	636s	67352s	deny
#8	549s	21s	5	795s	68147s	deny
mx2	900s	-	-	-	68147s	deny
#9	905s	356s	4	0s	68147s	deny
ids	926s	-	-	10800s	78947s	-
#10	926s	21s	5	795s	79742s	deny
ids	927s	-	-	10800s	90542s	-
mx2	1131s	-	-	-	90542s	deny
#11	1137s	211s	4	0s	90542s	deny
mx2	1543s	-	-	-	90542s	deny
#12	1548s	411s	3	0s	90542s	deny
#13	1569s	21s	4	636s	91178s	deny
ids	1570s	-	-	10800s	101978s	-
ids	1570s	-	-	10800s	112778s	-
mx2	1637s	-	-	-	112778s	deny
#14	1643s	74s	5	530s	113308s	deny
ids	1664s	-	-	10800s	124108s	-
ids	1664s	-	-	10800s	134908s	-
#15	1664s	21s	6	954s	135862s	deny
mx2	1903s	-	-	-	135862s	deny
#16	1909s	245s	5	0s	135862s	deny
#17	1930s	21s	6	954s	136816s	deny
mx2	2272s	-	-	-	136816s	deny
#18	2277s	347s	5	0s	136816s	deny
mx2	2296s	-	-	-	136816s	deny
...						

Chapter 4

Implementation

After the basic filter idea was modelled we wrote a first proof of concept implementation. This code was a very straight forward perl [per] script run over xinetd [xin]. Since first results were promising we decided to do a reference C implementation because the resources consumed by the perl/xinetd wrapper were much too high. The whole construction (see Figure 4.1) consists of a connection handler that accepts new incoming SMTP connections and forwards the connections to the real MX on permisson or refuses them with an error otherwise. In the background a userspace worker is doing asynchronous tasks like DNS look-ups or DNSBL checking. The communication between both is done via a MySQL [MySa] database. This chapter reveals some implementation details and introduces some additional tweaks to enhance the filter performance.

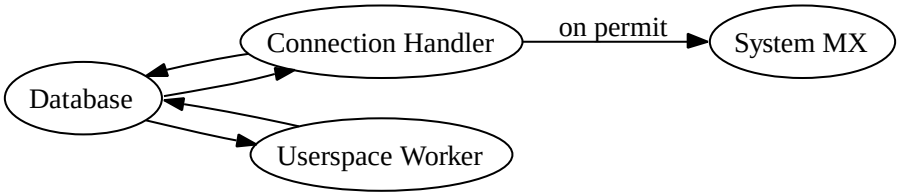

Figure 4.1: Interaction between DB, Connection Handler and Userspace Worker

4.1 Connection Handler

After a new incoming connection has been accepted, the IP address is first matched against permanent whitelisting and blacklisting rules. After that the penalty time for this IP is calculated depending on prior connection attempts if there where any. Depending on the penalty time having passed, the host is set to permit state or the connection is denied with a temporary SMTP error. Every first connection from a new IP address is always rejected and will not be permitted until the userspace worker has processed the new DB entry.

The exact behaviour of the connection handler can be studied in Figure 4.2. If the connection is denied, an appropriate error code and message is returned and the connection is closed. If the sender IP is permitted, the connection is passed on to the real MX transparent to the sending system and the local MX.

4.2 Userspace Worker

The idea behind the userspace worker is that possible time-wasting tasks like reverse or DNSBL look-ups can be done asynchronously from the connection handler. So the connection with the sending server is not held open unnecessarily long caused by slow DNS look-ups.

The userspace worker first resolves all unresolved IPs. If the DNS server fails, the look-up is delayed and retried serveral times. After having resolved the Pointer Resource Record (PTR)—the reverse record for an IP—the IP and PTR along with the local MX is matched against a set of rules in the database. These rules allow to increase the initital penalty, mask an entry as dynamic dial-up host or dynamically whitelist or blacklist a host. With these rules it is e.g. possible to give all hosts without a valid PTR an additional penalty or to declare a host as dynamic and blocking it depending on the local settings. Since the PTR matching is done using Regular Expressions [MySb] one can do quite appropriate PTR matching a lot stronger than with exisiting PTR matching mechanisms like those in Spam Assassin [SA0]. The idea is to give potentially bad hosts a higher

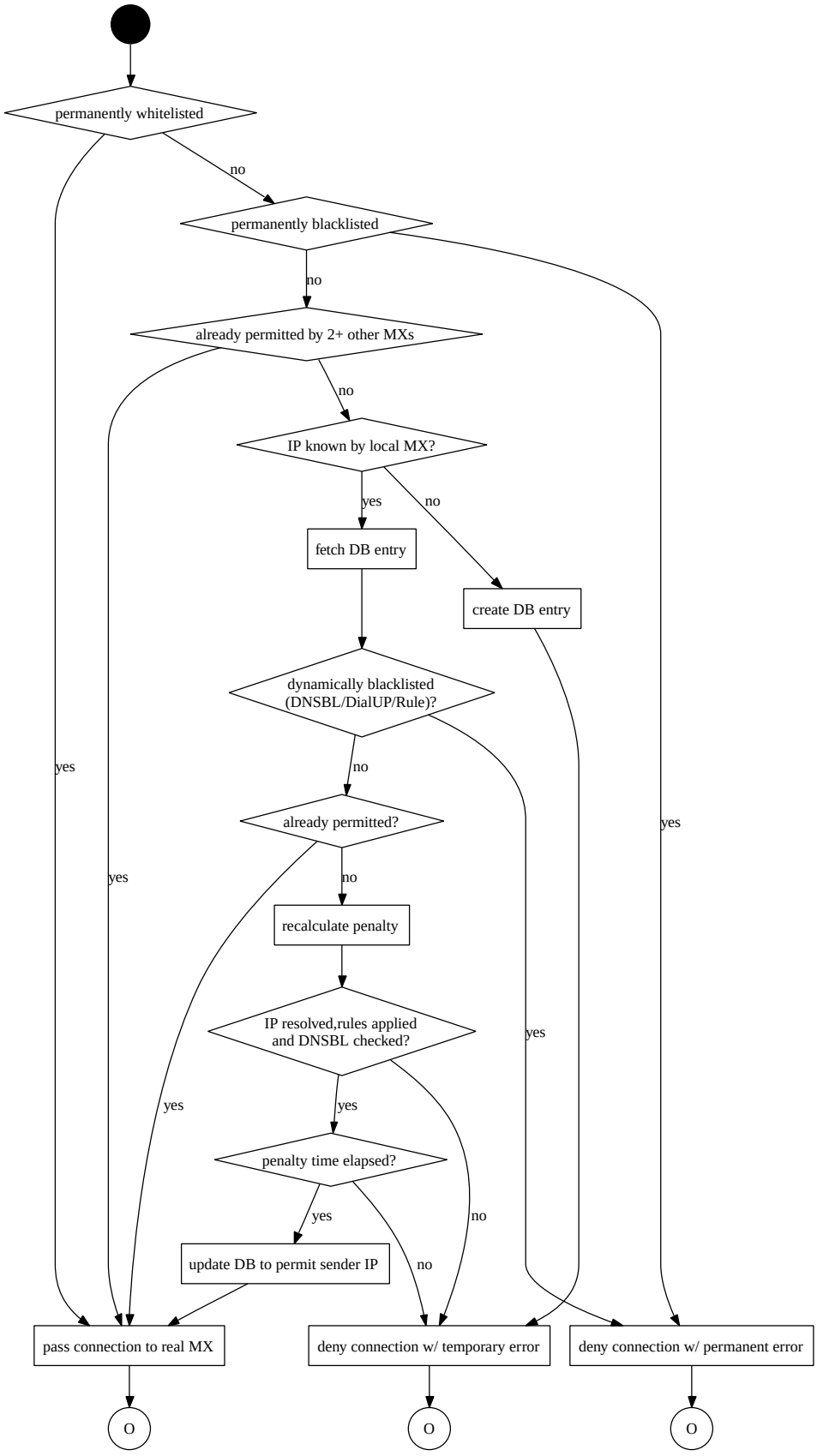

Figure 4.2: Flowchart of Connection Handler

initial penalty so that their probation period is longer. A comprehensive list of rules can be found in the sources attached to this paper.

With DNSBL checking enabled the userspace worker also performs a look-up on all configured lists if a host would be permitted at its next reconnect. This way it is guaranteed that the check takes place at the last worker run possible. There is also a periodically rechecking made for already permitted or already blacklisted hosts. If a host gets delisted from a blacklist, the dataset is deleted and the learning process is started over. This is done because all connections are refused with a permanent error while a host is blacklisted and therefore there was no adequate penalty calculation during that time.

The user space worker is also responsible for removing obsolete entries from the host table. Hosts are deleted if they are not permitted and have not retried for a timeframe of four days—the usual maximum delivery time for e-mails. Old host entries for unpermitted hosts need to be deleted to be able to evaluate the retry bevaviour for a new delivery attempt of a new message. It is also convenient to remove all permitted hosts if they have not sent any e-mail within the last months to keep the database from growing endlessly.

4.3 Database

The database backend which stores the information of connecting hosts is based on the MySQL [MySa] database engine. Currently there is a single database server storing all information for all MXs on which we tested the filter. Generally a new dataset is created for each unique combination of IP and MX. There is only one single row for each IP/MX combination so the database size will not grow with each single connection attempt.

All dynamic information is stored in the database for easy backupability. Beside the central IP/MX database there are additional tables which hold information about statically blacklisted or whitelisted hosts and additional rules and rejection messages.

4.4 Additional Tools

To make life with greylisting easier for users and gather even more patterns about the hosts connecting to our servers, we wrote some additional scripts.

This includes a small fake secondary MX server that just refuses incoming connections and logs the connection order. With this tool it is not necessary to have a complete instance of the real MX running as secondary.

The same daemon that was introduced to scan the firewall logs for portscans also analyses the mailserver logs and creates reverse permit entries in the host table for all servers that local users send mails to. So if a local user sends mails to a foreign host all mail coming back from that host are permitted immediately.

Since there are common lists of known servers that are legitimate and do no proper queueing or send time critical e-mails (like ebay) we wrote a small script that imports the common whitelist from PureMagic [Pur] and add them to our whitelist table.

Chapter 5

Performance Analysis

Since the beginning of July the reference implementation introduced in Chapter 4 is in production environment on three major mailservers. The subjective impression of the users on these mailservers was that the overall spam burden has significantly decreased although there was already ordinary greylisting installed on that servers before. However, analysis of a pre-MX filter is rather complicated because it is very difficult to distinguish whether a denied connection belongs to a delivery attempt of a spam mail or not. Since the mail behind a denied connection attempt cannot be classified as it is never received by the MX, we had to find other indications and values to evaluate.

5.1 Impact on the MX

To get a first impression what our filter is able to do we decided to compare significant values on the busiest mail exchanger (mx.dlh.net) on equal weekdays with the filter enabled in one week and disabled in the other.

On mx.dlh.net there is a CRS installed on most mailboxes. In Figure 5.1 one can see the number of mails remaining in the queue of the CRS for one of the busiest accounts on that system. The number of queued mails significantly decreased with enabled filter, and the number of entries blacklisted after being received by a background process doing regular DNSBL checks for already queued mails dropped almost to zero. This provides

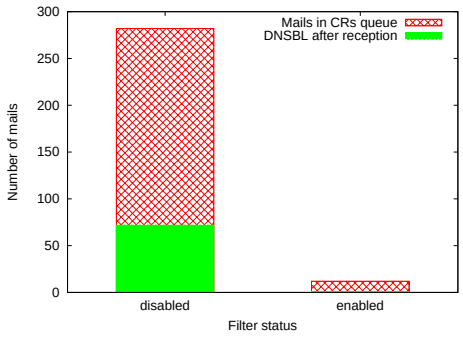

Figure 5.1: Queued Mails in CRS per Day

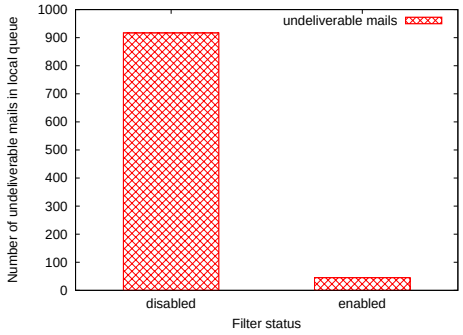

Figure 5.2: Undeliverable Mails in Local Delivery Queue per Day

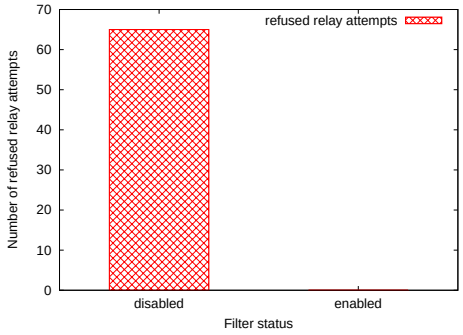

Figure 5.3: Refused Relay Attempts from External Hosts per Day

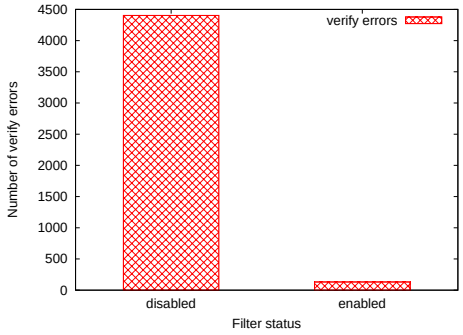

Figure 5.4: Verify Attempts for Non-existing Mailboxes per Day

an evidence for us having achieved our goal that our filter delays bad senders long enough to be listed by DNSBLs.

Associated with less entries in the queue of the CRS the number of undeliverable mails caused by CRS-challenges sent to non-existing addresses is also noticeably smaller which can be seen in Figure 5.2.

To check if our filter missed some misbehaving senders we also randomly checked some mails remaining in the queue of the CRS with enabled filter by hand. All of them have been found to have done proper queueing and timing during the greylisting process. So there was no chance for our filter to classify them as bad and block them out before they entered the queue.

We have also found the number of unauthorized relay attempts on mx.dlh.net caused by spammers checking for open relays dropping to zero with our filter enabled (see Figure 5.3).

Furthermore, we saw the number of failed verify attempts caused by spammers trying to send mail to non-existing mailboxes by using common given names als local part of the mail address dropping enormously (see Figure 5.4). All these facts can be summed up in the observation that the average system load on mx.dlh.net dropped almost 50 percent with enabled filter.

5.2 Filter Statistics

To get more detailed statistical information about the filter performance we analysed the database records generated by our reference implementation.

About one third of all spammer connections could be rejected because the filter was able to delay the delivery long enough for common DNSBLs listing their IP address. Furthermore, almost 98 percent of all hosts trying to connect to mx.dlh.net were never let in for various reasons, almost 50 percent alone by our filter mechanism. Even if all DNSBL checking was disabled, our later analysis revealed that 91.5 percent of all blacklisted hosts would never have been permitted because their behaviour caused so

Table 5.1: Statistical Breakdown of Connections per Month

Description	Count	Percentage
Total number of connections to mx.dlh.net	384 157	100.0%
denied temporarily	219 343	57.1%
denied permanently	137 949	35.9%
dynamic dial-up	62 261	45.1%
dns blacklisted	75 688	54.9%
initially	52 347	69.2%
during greylisting	23 341	30.8%
permitted	26 865	7.0%
obviously no spam (CRS)	14 761	54.9%
blacklisted afterwards	1 391	5.2%

Table 5.2: Statistical Breakdown of Distinct Senders

Description	Count	Percentage
Total number of distinct senders on mx.dlh.net	201 891	100.0%
senders never let in	197 368	97.8%
dynamic dial-up	38 266	19.4%
dns blacklisted	62 840	31.8%
initially	47 110	75.0%
during agreylisting	15 730	25.0%
bad enough behaviour	57 498	91.5%
adaptive greylisting	96 262	48.8%
senders permitted	4 523	2.2%

much penalty that they would have been blocked out in any case. Considering those connections that have been permitted by our filter, about 55 percent were obviously no spam because the corresponding senders managed to pass the CRS. The rest remaining in the queue were mostly *failed delivery notifications* or *virus alerts* caused by spammers using a local address as forged sender.

Tables 5.1 and 5.2 give some additional statistical breakdowns of the filter performance. The data on the reference implementation was collected between June 12th and July 29th 2006. The records in Table 5.1 have been normalised to a 30-day-basis. Table 5.2 contains data for the whole period.

Chapter 6

Conclusion

In this thesis we have described a simple spam filter that enters the mail delivery process at a very early stage. It is able to keep a huge amount of spam out while being very leightweight which is usually considered a very good thing.

Beginning with the oberservation in our logfiles and changed spamming behaviour, we have built a data collection framework to validate our initial assumptions in the later evaluation. Based on the pattern collection we found a very easy model to honour good and bad behaviour and lock out bad senders dynamically.

The later evaluation revealed that this model is able to shift a lot of load away from the mail servers and therefore reduces the overall spam burden significantly.

The resources saved by filtering the mail at a very early stage can now be spent for more complex, higher level filters that can then detect the spam that still managed to find its way through our filter system.

6.1 But What if Spammers Adapt?

As with any spam filter that is newly developed an important question is: *will the filter still be effective if the spammer has knowledge how it works?* Filters that analyse the structure of body and headers of a message might be cheated if the structure

of the spam message is altered. Other filters might scan for characteristic keywords or do probability analysis. In these cases spammers will and already do adapt heavily.

Since our filter does not take any mail content or structure into account, there might be a good chance that spammers might not be able to adapt that easily. If we categorize our filter as a greylisting-like filter, we first might think of the ability to cheat normal greylisting and then see if our implementation has the same weaknesses.

The reason why normal greylisting is quite effective is that spammers currently do not use a unique sender address when targeting a specific recipient address. Even if a spammer does no appropiate queueing, there is no reason why he should not be able to send two or more spam attacks within a short timeframe. Maybe the first one is only to setup the triple in the greylisting table and after the usual static amount of time all mails would be permitted. It would be no big deal to build a hash function that generates a unique sender address out of each recipient's e-mail address. At this point storing triples is nearly useless and normal greylisting is worn out.

The key is that spammers have to send their spam messages very fast and cannot do appropiate queueing and timing without getting ineffective. Of course, every spammer could use a standard RFC-compliant mail transfer agent, but there is a good reason why he usually does not. A spammer sends millions of mails and even if only a small amount of mails had to be queued appropiately, it would be so resource intensive that this would lead to ineffectiveness. Normal mailservers get very high-loaded if only a few hundred mails are queued. Considering only one percent of one million mails not being delivered immediately, the spammer's host would be busy with all the queueing and redelivery attempts and not be able to send out new mails. Since spammers usually send out copies of the same mail to millions of recipients they had to write their own MTA not queueing the mail, but just keeping a database of already tried and temporarely delayed addresses. Currently they do not seem to do this and hopefully they will not bring up that idea too soon.

Spammers will likely behave non-RFC compliant for the next time and as long as they do, our filter will be able to identify them with a good probability. However, a big percentage of mail is filtered out because the sender gets DNSBLed while we let the sender retry. If all servers would use our filter, there would be no content filter which

is able to identify the mail sender as being a spammer and give feedback to a blacklist provider.

Further we have to keep in mind that there will always be a way to cheat that we cannot foresee and the filter's effectiveness will strongly depend on how widely it is spread. It is the same reason why there were no viruses for Linux or OSX for a long time. If it is not necessary to do proper SMTP while still having a high delivery rate, there will be no spammer who is eager to enhance his techniques.

However, we can definitely say that our approach is another way to force spammers to act RFC-compliant and this automatically leads to decreased throughput on their side and is therefore a benefit for all peaceful e-mail users. Spammers have two opportunities when trying to improve their efficiency—they can simply improve the amount of spam sent or they can improve the quality of spam [AF06]. If we force them to stay RFC-compliant with filters blocking them out otherwise, their only option is to increase the quality of their spam which is unequally harder than just compromising more hosts and sending more spam.

6.2 Future Work

Since the first test results of our reference implementation are very promising, there is a wide set of opportunities to get even better filter results. Some of our ideas for improvements and further research are sketched in this section.

A first option to circumvent the listener that forks and then calls a MX process would be a Linux iptables [ipt] module that matches against all IP addresses that are already permitted. Connected to this module should be a userspace module that updates some hash table from the database since a real database connection directly from a kernel module would not be ideal. With such a module it would be possible to allow only connections which an IP match in the hash table to connect directly to the MX. All other connections could be redirected to the connection handler process—now being just a connection refuser—by NAT rules.

Another idea to influence the decision process would be some artificial intelligence mechanisms for pattern classification. The penalty system currently used in our reference implementation is very simple. We followed the idea of using Support Vector Machines (SVMs) [Wikf] for classification, but ran into trouble since these machines need well formatted training data, and sadly our approaches led to too many cases where our SVM was not able to classify patterns at all.

Since our filter is not the final solution to filter all spam out, it would be a good idea to have feedback from higher level spam filters like SpamAssassin [SA0] about very negatively scored IPs and be able to revoke a permitment of that certain IP and add additional penalty. In our filter this could be used to alter the initial penalty value or the expected retry time making it harder to pass the filter. If some hosts from a certain netblock have sent a high amount of spam or behave maliciously, the values in our filter could be adjusted for that netblock in advance making it harder for all further senders from there to get permitted. If mechanisms like SVMs were used for classification they could be retrained using classification from higher level filters to get better classification results.

As mentioned before our filter is ideal for using network effects and sharing information across multiple MXs. Currently we only implemented the ability to share information about good IPs giving them the ability to pass our filter earlier or even at the first connection attempt. It would also be possible to share information about bad behaviour and adjust penalty and other variables therewith.

Beside sharing the information about good IPs within a cluster of MXs that have to trust each other, there could be a new facility of DNS Whitelists (DNSWLs) similar to DNSBLs. With such an DNSWL the information about good IPs could be distributed via the stable and widely spread DNS system and offered third parties without having to trust them and grant them access to the local infrastructure. However, as with DNSBLs one has to trust the DNS [Eis05], but this is beyond the scope of this work.

Bibliography

[AF06] J. Aycock and N. Friess. Spam Zombies from Outer Space. In *15th Annual EICAR Conference*, pages 164–179, April 2006.

[Alo04] Elizabeth A. Alongi. Has the U.S. canned spam? In *Arizona Law Review*, volume 462, pages 262–290, 2004.

[Ban05] Banit Agrawal and Nitin Kumar and Mart Molle. Controlling Spam E-mail at the Routers. In *IEEE International Conference on Communcations*, pages 1588–1592. University of California, Riverside, May 2005.

[CHMS06] Duncan Cook, Jacky Hartnett, Kevin Manderson, and Joel Scanlan. Catching spam before it arrives: Domain specific dynamic blacklists. In *Fourth Australasian Information Security Workshop (Network Security) (AISW 2006)*, volume 54 of *CRPIT*, pages 193–202, Hobart, Australia, 2006. ACS.

[Dan04] Hadmut Danisch. The RMX DNS RR and method for lightweight SMTP sender authorization. http://www.danisch.de/work/security/txt/draft-danisch-dns-rr-smtp-04.txt, 2004.

[dlh] DLH.Net - the gaming people (*as seen Aug 7th 2006*). http://www.dlh.net/.

[Eis05] Peter Eisentraut. Consequences of Spam and Virus Filtering for the E-Mail System. In *22nd Chaos Communication Congress*, 2005.

[exi] exim Internet Mailer (*version 3.36*). http://www.exim.org/.

[Fed05] Federal Trade Commission. Effectiveness and Enforcement of the CAN-SPAM Act. In *A Report to Congress*, 2005.

[Har03] Evan Harris. The Next Step in the Spam Control War: Greylisting. http://projects.puremagic.com/greylisting/whitepaper.html, 2003.

[ipt] The netfilter/iptables Project (*as seen Aug 7th 2006*). http://www.netfilter.org/.

[Iro06] IronPort Systems. Spammers Continue Innovation: IronPort Study Shows Image-based Spam, Hit & Run, and Increased Volumes Latest Threat to Your Inbox. http://www.ironport.com/company/ironport_pr_2006-06-28.html, 2006.

[Kle01] J. Klensin. Simple Mail Transfer Protocol. RFC 2821 (Proposed Standard), April 2001.

[Lev05] John R. Levine. Experiences with Greylisting. In *Second Conference on E-mail and Spam*, Mountain View, California, July 2005.

[LW06] J. Lyon and M. Wong. Sender ID: Authenticating E-Mail. RFC 4406 (Experimental), April 2006.

[Mes06] Messaging Anti-Abuse Working Group. Email Metrics Program - 1st Quarter 2006. http://www.maawg.org/about/FINAL_1Q2006_Metrics_Report.pdf, 2006.

[MySa] MySQL AB. MySQL – The world's most popular open source database (*version 4.1*). http://www.mysql.com/.

[MySb] MySQL AB. Regular Expressions in MySQL (*version 4.1*). http://dev.mysql.com/doc/refman/4.1/en/regexp.html.

[per] Practical Extraction and Report Language (*version 5.8.4*). http://www.perl.com/.

[pos] Postfix (*as seen Aug 7th 2006*). http://www.postfix.org/.

[Pos82] J. Postel. Simple Mail Transfer Protocol. RFC 821 (Standard), August 1982. Obsoleted by RFC 2821.

[Pos84] J. Postel. Domain name system implementation schedule - revised. RFC 921, October 1984.

[Pur] PureMagic Software. List of common manual whitelist entries (*revision 1.16*). http://cvs.puremagic.com/viewcvs/*checkout*/greylisting/schema/whitelist_ip.txt.

[qma] qmail: Second most popular MTA on the Internet (*as seen Aug 7th 2006*). http://www.qmail.org/top.html/.

[SA0] The Apache Spamassassin Project (*as seen Aug 7th 2006*). http://spamassassin.apache.org/.

[Sle04] Tor Slettnes. Spam Filtering for Mail Exchangers. http://ftp.linuxarkivet.se/pub/linuxdoc/HOWTO/other-formats/pdf/Spam-Filtering-for-MX.pdf, 2004.

[Tem] Brad Templeton. Origin of the term "spam" to mean net abuse (*as seen Aug 7th 2006*). http://www.templetons.com/brad/spamterm.html.

[TMD] Tagged Message Delivery Agent (TMDA) (*as seen Aug 7th 2006*). http://tmda.net/.

[Wika] Wikipedia, the free encyclopedia. DNS-based Blackhole Lists (*revision 07:15, 7 August 2006*). http://en.wikipedia.org/wiki/DNSBL.

[Wikb] Wikipedia, the free encyclopedia. Hammering (*revision 23:41, 19 February 2006*). http://en.wikipedia.org/wiki/Hammering.

[Wikc] Wikipedia, the free encyclopedia. Open mail relay (*revision 20:08, 31 July 2006*). http://en.wikipedia.org/wiki/Open_mail_relay.

[Wikd] Wikipedia, the free encyclopedia. Ratware (*revision 20:46, 17 July 2006*). http://en.wikipedia.org/wiki/Ratware.

[Wike] Wikipedia, the free encyclopedia. Spam (electronic) (*revision 10:01, 6 August 2006*). http://en.wikipedia.org/wiki/Spamming.

[Wikf] Wikipedia, the free encyclopedia. Support Vector Machine (*revision 09:32, 1 August 2006*). http://en.wikipedia.org/wiki/Support_vector_machine.

[Wikg] Wikipedia, the free encyclopedia. Zombie computer (*revision 18:43, 27 July 2006*). http://en.wikipedia.org/wiki/Zombie_computer.

[WS06] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408 (Experimental), April 2006.

[xin] xinetd–a secure replacement for inetd (*version 2.3.14*). http://www.xinetd.org/.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 11. August 2006 Peter Lieven