HEINRICH HEINE
UNIVERSITÄT
DÜSSELDORF

Intelligent Greylisting

Bachelor Thesis

by

Evgeni Golov

from

Kiev

submitted to

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

November 2009

Advisor:
Peter Lieven, MSc.

Acknowledgments

A lot of people supported me during my work on this thesis to whom I wish to express my gratitude. First of all (of course) Peter Lieven for supporting me with discussions and hints, Michael Singhof and Jürgen Osterberg for proof-reading and corrections and my family for their support. Then the whole Debian community, especially the maintainers of Python, Postfix, PostgreSQL and OpenVZ, for the great and stable operating system, so I could focus on writing and testing my software instead of keeping the server alive. And while speaking about servers, thanks to Patrick Kambach and the whole `kambach.net` team for hosting mine.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

DNS	Domain Name System
DNSBL	DNS-based Black List
DNSWL	DNS-based White List
FQDN	Fully Qualified Domain Name
MTA	Mail Transport Agent
RFC	Request for Comments
RHSBL	Right Hand Side Black List
SMTP	Simple Mail Transfer Protocol
SPF	Sender Policy Framework

Chapter 1

Introduction

E-mail spam is probably known by everyone: one gets e-mails about products and services one does not need or want. That is mainly because of the fact that sending e-mails is cheap and easy, the sender does not have to pay for paper, printers and post-stamps, he only needs a computer and an internet connection.

Since the arise of spam spam-filters have started to exist and with them the problem of valid mail being classified as spam, the so called false positives, and thus filtered out.

In this thesis we will present a method for identifying possible spam without the risk of losing valid mail which might happen when e-mail is rejected based on lists with known spammers – the so called blacklisting, and mostly without delaying it, which might happen with regular greylisting.

This work cannot replace context-filters like SpamAssassin [spa], because it handles the incomming connection before the actual mail is transferred and thus has only very limited knowledge about its content.

1.1 SMTP

Before one can start fighting spam, one has to understand, how mail is delivered in the Internet from the sender (the server which sends the mail) to the recipient (the server that recieves the mail and stores it into the users mailbox).

The Simple Mail Transfer Protocol (as described in RFC 5321 [Kle08]) is used for this purpose. As the name implies, it is a simple protocol "spoken" between the sending and the recieving host. An usual SMTP connection looks like this:

```
sender connects via TCP to receiver, port 25
receiver:  220 bob.example.org running some SMTP daemon
sender:  HELO alice.example.org
receiver:  250 hello alice.example.org
sender:  MAIL FROM: <alice@alice.example.org>
receiver:  250 <alice@alice.example.org> ...  Sender ok
sender:  RCPT TO: <bob@bob.example.org>
receiver:  250 <bob@bob.example.org> ...  Recipient ok
sender:  DATA
receiver:  354 End data with <CR><LF>.<CR><LF>
sender passes the mail
sender:  .
receiver:  250 Ok
sender:  QUIT
receiver:  221 Bye
```

One discerns a pattern in this protocol: the sender uses a command (the common ones are listed in Table 1.2) and the receiver replies with a status code followed by a human readable message. The status codes are devived in four groups shown in Table 1.1.

In the above example, `bob.example.org` would reply with a `450` code after the `RCPT` command, if there is no space left on the harddrive to store the mail and a `550` code if there is no user called bob at all.

`2xx`	The command was accepted and will be processed.
`3xx`	The command was accepted but needs further information to be processed.
`4xx`	The command resulted in a temporary error.
`5xx`	The command resulted in a final error.

Table 1.1: SMTP Codes

command	meaning
HELO	Greet the server.
MAIL FROM	Which address the mail originates from?
RCPT TO	Which address should the mail be delivered to?
DATA	Begin the transfer of the mail data.
QUIT	Quit the connection.

Table 1.2: Common SMTP Commands

1.2 Spam

Spam (not the canned meat by Hormel Food [GS98, p. 11]) is unwanted e-mail, every user of e-mail services already got in his/her mailbox. Senders of such messages do not have our permission to flood our mailboxes with information about products we do not need and services we will never use, but they still do – it is a very cheap way to advertize the products. While spam can be found not only in mails, but also in instant messaging, newsgroups, blogs, forums etc., we will focus on mail spam here, as this is the kind we can fight with SMTP-level methods.

1.2.1 The History of Spam

Spam in form of chain letters, viruses and other junk exists since the late 1980s. Real bulk "mail" came in April 1994, when Laurence Canter and Martha Siegel sent their spam to over 6000 newsgroups on the Usenet [GS98, p. 22] (while newsgroups actually are not e-mail, the technique is quite similar). The business continued in 1995 with Jeff "Spam King" Slaton [GS98, p. 23] and in 1996 with Sanford Wallace [GS98, p. 25]. The later is remarkable because he used only a couple of valid domains for his mail, which led to first spam filters at America Online (AOL) in 1996 [GS98, p. 27]. AOL's filters were simple, they just blocked the domains Wallace was using. Later customers were able to define own lists of senders they wanted to accept (whitelist) or to block (blacklist).

The same America Online stated in 1997 that about a third of their mail traffic was spam [GS98]. That number grew in 2003 to 70-80% (that's about 500.000.000 mails a year) [Sie08]. Latest numbers from one of our own mail servers (`mail.die-welt.net`) show about 20.000 mails a day (in September 2009), with only about 500 passing the filters, making a spam rate of 97.5%.

1.3 Motivation

Every successful project needs a good motivation to start with. For us the motivation to start yet another spam-filtering software was not the amount of spam we got - our setup managed to filter it well enough - but the possibility to loose valid mail.

1.3.1 Personal Motivation

The first time we realized, we were loosing mail was when we wanted to register for the release candidate of Microsofts new operating system, Windows 7. This registration involved receiving a confirmation e-mail. When that e-mail did not arrive for some five or ten minutes, we started to wonder what was happening. The log file of our server explained a lot – our spam filter rejected the mail, because it thought it was spam. We were not able to do anything against it, the mail was gone. The solution for that particular problem was easy – using a different e-mail address (hosted on a different server) which did not have a spam-filter installed. But the problem of lost mail on the first server persisted and had to be solved.

The second time we faced the same problem, was when we were expecting an e-mail from Belarus, which again was rejected by our filter.

That behaviour was inacceptable and we started to think about improvements of the setup without loosing the advantages of distributed knowledge in form of blacklists and without adding the usual greylisting delay.

1.3.2 Professional Motivation

Besides of the two incidents described above, there is also a more professional motivation in reinventing that wheel: Internet Service Providers have customers which pay for hosting their mail. These want the perfect solution, zero spam and zero lost mail. Zero lost mail is impossible when using blacklists. On the other hand, greylisting does not fit either, or how does one explain the sales-department that they have to wait an hour before they can get the mail from the new distributor?

Additionally there seems to be no consent whether it is legal to filter spam (at least as per German law), even when people argue that at least the use of blacklists is legal [Sab08].

1.4 Structure

At first we give an overview of existing spam filtering techniques, focusing on those which do not analyze the mail content, in Chapter 2. Later in Chapter 3 we will present current implementations of the different techniques, their pros and cons. In Chapter 4 we will continue with the needed technical background and our basic idea. Our reference implementation is then presented in Chapter 5 and evaluated and benchmarked in Chapter 6.

Chapter 2

Preliminaries

Before presenting existing solutions, we will explain different approaches of identifying and filtering spam. First of all one has to distinguish between filters that run after the `RCPT` command (the so called Pre-MX filters), analyzing only the source and destination of an e-mail, from filters that run after the actual mail content was transmitted and accepted by the mail server (the so called Post-MX filters), analyzing the whole mail, including headers and content. Furthermore one has to distinguish between filters that reject the mail (when they are Pre-MX ones) or delete it (when they are Post-MX ones) and those which only tag the mail as spam (and probably move it into a special subfolder of the mailbox, which is dedicated to spam).

As we are only interested in filtering spam as soon as possible, we will only look at Pre-MX solutions in this thesis. Post-MX solutions are still useful, but will not be discussed here.

2.1 Blacklisting

Blacklisting (sometimes also called Blocklisting) is the name for every approach that forbids the delivery of mail based on some criteria. As noted earlier (in Section 1.2.1), the first blacklists were introduced by America Online in 1996 to block mail from domains owned by Sanford Wallace for spamming. These were just lists of domains from which mail was not accepted by the mail servers. But today spammers use regular domains to send spam (according to statistics by Commtouch Software Online Labs, the top 5 domains in 2009 are: compusa.com, walmart.com, hotmail.com, gmail.com and yahoo.com [Lab]). That makes filtering by domain almost useless.

As blacklisting usually happens after the sender emitted the HELO, MAIL and RCPT commands, one can not only block by sender's domain, but also based on his IP address, reverse hostname etc. The IP address is a sort of unforgable (unless one uses proxies and similar utilities) source identification, which is often used to separate known spammers from regular senders. The lookup is usually done via DNSBLs in the following manner: Given a sender with the IP address `10.11.12.13` and a DNSBL `dnsbl.example.org`, the software looks up the A DNS record for `13.12.11.10.dnsbl.example.org` (the IP address is written reversed, as also done with reverse DNS records: `13.12.11.10.in-addr.arpa`) and interprets the DNS reply. Such hosts usually resolve into addresses from the `127.0.0.0/24` range (sometimes also `/16` or `/8`, depending on how much information the list wants to share), telling us the IP address is listed for spamming, phishing, etc. The DNS server of the blacklist will send an error instead of a reply, if the IP address is not listed in this DNSBL.

Besides of DNSBLs, there are also RHSBLs which work the same way, but contain information about hostnames instead of IP addresses. These are queried via `somehost.com.rhsbl.example.org`.

2.2 Whitelisting

Whitelisting is very much the same as blacklisting in terms of the technique, just the use is the exact opposite: one explicitly allows mail from a sender found in a whitelist (usually without doing further checks).

As with DNS blacklists, there exist also DNS whitelists (DNSWLs). The most famous and complete is probably `dnswl.org`, queried via `list.dnswl.org`.

2.3 Greylisting

Greylisting is different from black- and whitelisting, because it does not work with a predefined list, but "learns" good and bad hosts as time passes.

RFC 5321 requires every RFC-compliant SMTP server to have a queue for outgoing mail, so it can backoff mail that could not be delivered to the next hop and retry after a period of time. If one can limit the senders to those which correctly follow the RFC, one can reject their first attempt to deliver a mail with a temporary error (usually code 450), note the IP address and wait. The RFC specifies that the first retry should happen about 30 minutes after the error, the last 4-5 days after it [Kle08, Section 4.5.4.1]), which means that most of the senders retrying earlier than 30 minutes might be spammers and usually they will be rejected again. But if a sender returns after a period long enough, it is considered a good one, the mail is accepted and the IP address written on the whitelist, so the sender will not need to wait the next time (see Figure 2.1). Spammers could use rfc-compliant servers and retry sending out their spam after a while, which would look like good one to a greylisting implimentation, however it seems they do not do so (at least they did not do since the proposal of greylisting in 2003 [Har03], [Wik09]). Most probably because backing off the mail and waiting longer than a couple of minutes is just too expensive in terms of ressources and spammers seem to prefer sending out another thousands mails in that time.

2.4 SPF

SPF bases on the idea that the administrator of a domain knows who is responsible for sending mail for that domain [WS06]. If this is known, he can publish the list of allowed hosts and the recieving SMTP servers can check the sender against this list. If the sender

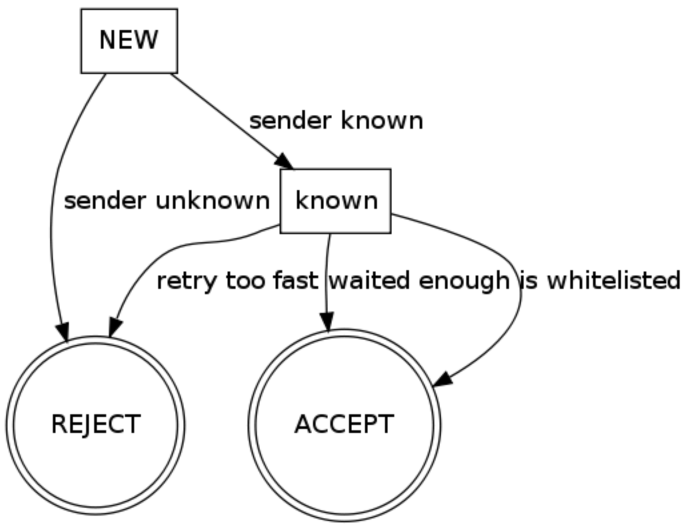


Figure 2.1: A simple Greylisting approach

is in the list, the mail should be treated as genuine and accepted. In case of failure (the sender is not in the list), the receiver might reject the mail directly, or at least handle it as spam and sort it accordingly.

SPF information is provided in form of TXT DNS records. An usual SPF record looks like this:

```
IN TXT "v=spf1 +ip4:123.123.123.123/32 -all"
```

This means that only the host with IP address 123.123.123.123 is allowed (+) to send mail for this domain and everyone else is not (-). The administrator can also specify entries prefixed with an ~ and an ?, meaning the entry is probably not allowed to send mail (~) and completely unknown (?). The possible results of a SPF query are listed in Table 2.1.

2.4.1 SPF Best Guess

Often a domain just has no SPF record, making the check return "None" and thus being quite useless. To avoid that, SPF Best Guess [spf] tries to guess an apropriate SPF record

None	There is no SPF record published for the domain.
Neutral	The SPF record does neither forbid nor allow sending mail for that domain. (Should be treated exactly as "None".)
Pass	The SPF record allows sending mail for that domain.
Fail	The SPF record forbids sending mail for that domain.
SoftFail	The SPF record should forbid sending mail for that domain, but the author was not 100% sure.
TempError	An error (e.g. DNS timeout) occoured while checking the SPF record.
PermError	The SPF record was faulty/could not be intepreted.

Table 2.1: Possible SPF Query Results

for a domain. This record usually looks like this:

```
v=spf1 a/24 mx/24 ptr ?all
```

That means that every host which is in the /24 network of the domain's A and MX records is allowed to send mail. So is every host which has a working reverse DNS entry (PTR) in the same domain. All others are treated as "Neutral".

2.4.2 SPF Critic

The biggest critic on SPF is that it breaks mail forwarding: when a user has an e-mail address like `a@domainA.com` which is set to directly forward mail to `b@domainB.org` and recieves mail from `someone@example.org` to the `domainA.com` account (which then gets forwarded), the server at `domainB.org` would think `domainA.com` is forging mail, because the host is not listed in the SPF record of `example.org` [Woo05]. Thus, the user would loose mail if the server at `domainB.org` is set to reject mail based on SPF results.

Chapter 3

Related Work

3.1 Postfix

Postfix is the MTA used in this thesis. It has various ways of handling users and mailboxes (incl. lookup in MySQL, PostgreSQL and LDAP). But when it comes to spam filtering, Postfix' capabilities are quite limited. It can do simple blacklist checks or delegate the filtering to a different process (the technical side of this is shown in Section 5.1).

When doing blacklist checks, Postfix looks up the sender in a defined blacklist and will immediately forbid the delivery of mail with an 550 code if the sender is found in that list. Such filtering may become very dangerous if a list is shut down and its DNS server starts to reply wrong information to every query or when list administrators decide to list (sort of) innocent IP-blocks as spammers, as happened to nic.at in 2007, when Spamhaus.org decided to list all nic.at mail servers as spammers because nic.at refused to delete domains which were used for phishing [SZ07].

`client_ip_eq_helo`	Does `client_ip` match the IP address of the host given in EHLO/HELO?
`helo_from_mx_eq_ip`	Has the MX listed for senders from address the same IP address as `client_ip`?
`helo_numeric`	Is the host given in EHLO/HELO numeric?
`helo_seems_dialup`	Does the host given in EHLO/HELO look like a dialup host?
`client_seems_dialup`	Does the client reverse DNS look like a dialup host?

Table 3.1: policyd-weight Checks

3.2 Postfix Policy Daemons

policyd-weight

To avoid false positives based on a single blacklist checked by Postfix, `policyd-weight` [pol] exists. It's a policy daemon for Postfix written in Perl, which can do (as the name suggests) weighted decisions, based on DNSBL, RHSBL and RFC checks.

`policyd-weight` contains a list of DNSBLs, each with a hit and miss score. That means that if a sender is found in a DNSBL, its hit score is added to the senders score and the miss score is substracted from it, in case the sender is not found. Per default, `policyd-weight` rejects every client that is listed in more than two of these lists or has a score more than eight.

The same technique is applied to the RHSBLs: a hit increases the senders score, a miss decreases it. Now `policyd-weight` temprorary rejects the sender after a score of 1 per default. Whereas a final reject happens after a score of 5. The RFC checks are added to this score too. The most interesting ones are shown in Table 3.1.

policyd

`policyd` is a Postfix policy daemon, written in C, using MySQL as storage (analyzed version 1.82, [posa]). It consists of multiple modules, supporting different spam filtering

Check in whitelist (which is filled by multiple succesful deliveries)
Check in blacklist (which is filled by multiple failed deliveries)
Spamtraping (delivery to special mailboxes)
HELO Check
Greylisting
Throttling of senders/recipients

Table 3.2: policyd Modules

methods. These are shown in Table 3.2, including the order of execution. Each of the checks can reject the mail (either temporary or finally – this is configurable), accept it, or do nothing (and let `policyd` continue checking). This will lead to both previously described problems: delaying and rejecting valid mail, depending on the configuration.

3.3 Greylisting Daemons for Postfix

There exist many greylisting daemons for Postfix. All those work quite similar: they use a database in which the senders are stored. When a new mail arrives, the sender is looked up in the database and either asked to wait (using a 4xx error code), or allowed to deliver the mail because the sender is already known and has waited enough, or asked to wait longer because the sender retried too fast. Table 3.3 shows common methods for lookup of the sender in the database.

Most greylisting daemons for Postfix support some (like `postgrey` [posd]) or all (like `sqlgrey` [sql]) of these methods and work basically as described in in Section 2.3 and usually only differ in the way they save the collected information. Because of that, they all suffer from the same problem: they delay at least the first mail from an unknown sender, before delivering it to the recipient.

tumgreyspf

`tumgreyspf` is a mix between a greylisting daemon and a SPF checker written in Python (analyzed version: 1.35, [tum]). When checking SPF, `tumgreyspf` rejects all mail with a "Fail" or "PermError" result permanently (code 5xx) and that with a "Tem-

IP address	IP address of the sender matches the database
IP address, sender, recipient	IP address, e-mail address of the sender and the recipient matches the database
Class C	Class C (/24) network of the sender matches the database
Class C, sender, recipient	Class C (/24) network, e-mail address of the sender and the recipient matches the database

Table 3.3: Common Sender Lookup Methods

pError" temporary (code 4xx). For domains which do not have a SPF record ("None" result), greylisting is applied. All other mail is accepted (that with "Pass", "SoftFail" and "Neutral" results - even when RFC 4408 demands "Neutral" mail to be handled exactly as "None" [WS06]). The greylisting algorithm of `tumgreyspf` is exactly as described in Section 2.3 with all four lookup methods shown in Table 3.3.

gld

`gld` is yet another greylisting daemon for Postfix [gld], written in C, using a MySQL database as storage (analyzed version 1.7). Besides doing greylisting, gld can check a DNSWL (only one) and a local whitelist to bypass greylisting of known-to-be-good senders. Doing so, `gld` will not greylist senders like GMX.net, which care to be listed in a whitelist, but still delays mails from valid senders who do not care to be listed in whitelists or do not know about the existance of such lists.

gross

`gross` follows a different approach as the other greylisting daemons presented here: instead of greylisting every new incoming sender, `gross` looks up the IP address in the configured DNSBLs and greylists only in case it was found in "enough" DNSBLs [gro]. It is also capable of querying RHSBLs and DNSWLs to increase or decrease the senders score. Additionally, `gross` can block senders with a higher score completely, e.g. when it is configured to greylist for scores higher than one and block for scores higher than five. This, however, is turned off by default.

Among all systems discussed, this seems to be the most promising approach: `gross` tries to avoid delaying mail from non blacklisted senders, which however means that all non blacklisted ones will get through without further checking, which might include fast spammers that are not listed in any DNSBL yet.

gps

`gps` is another greylisting daemon for Postfix, written in C/C++ and using MySQL/PostgreSQL/SQLite3 as database backend (analyzed version: 1.005, [gps]). The interesting difference in `gps` is an additional sender matching method: reverse DNS matching – when it resolves a connecting client to `mail-server255.someisp.com`, it saves `someistp.com` to the database, thus allowing all clients from the someisp.com domain to deliver now. This speeds up the greylisting of larger server farms, but still has the problem of initial delaying, which we want to avoid – `gps` does this with a training/init mode, where it learns the hosts, but does not greylist them. This means at that time, no spam filter is active at all.

3.4 Penalty Based Greylisting

Most greylisting implementations use a static delay (usually ten to 60 minutes) between the first appearance of a new sender and the moment the retry will be accepted. One can, however, make this more dynamical, by issuing different penalties for actions (like retrying too fast etc), as shown in [Lie06]. Using this approach, it is possible to use a low initial delay, and still filter out most malicious senders. This, however, does not solve the initial delay problem, we are focusing on.

Chapter 4

Technical overview

4.1 Postfix SMTP Access Policy Delegation

Instead of (or additionally to) checking the mail/sender itself Postfix can, since version 2.1, delegate these checks to an external service via the Postfix SMTP Access Policy Delegation Protocol [posc].

The external service opens a socket (either TCP or UNIX-domain) and waits for a connection from Postfix. When Postfix gets a new mail it writes a couple of parameters (formated as `key=value` lines) followed by a final emtpy line to the socket and expects a reply from the policy server.

For a policy server that will do black- or greylisting the parameters in Table 4.1 are relevant.

When the service has finished checking it replies Postfix over the same socket with an `action=SOMETHING` line (again followed by an empty line). Postfix supports a wide range of actions [posb] but only the following ones are relevant for us:

- `DEFER_IF_PERMIT` - reject the mail with a temporary error code (450) if it would have been accepted after other checks are done

`helo_name`	hostname transmitted in `HELO/EHLO`
`sender`	mail address transmitted in `MAIL FROM`
`recipient`	mail address transmitted in `RCPT TO`
`client_address`	the IP address of the sender (might be IPv4 or IPv6)
`client_name`	the reverse DNS name of the IP address or "unknown" in the case the IP has no reverse DNS entry or the reverse DNS entry does not resolve back to the IP

Table 4.1: Selected Postfix Policy Deligation Parameters

- `DUNNO` - do not reject but also do not accept the mail explicitly (if we would accept it here, we might ignore other checks which can lead to unwanted behaviour)

- `PREPEND` - same as `DUNNO` but add a line to the header of the mail (useful for documenting the assigned score and the results of the checks)

The example from Section 2.3 could be realized as follows:

Algorithm 4.1.1 Example Greylisting Algorithm

```
if (ip, sender, recipient) ∈ DB then
  if whitelisted then
    return action=DUNNO
  else if waited > 30min then
    save (ip, whitelist=1) into DB
    return action=DUNNO
  else
    return action=DEFER_IF_PERMIT
  end if
else
  save (ip, sender, recipient, last_connect=NOW) into DB
  return action=DEFER_IF_PERMIT
end if
```

4.2 Intelligent Greylisting

To achieve the goal of this thesis, a spam filter that neither looses any mail nor delays the incoming mail too long, basic greylisting shown in Section 2.3 is not sufficient. The original idea is very simple: greylist based on DNSBL results. As most legitimate senders are not listed in those (experience shows that some are) they would be able to deliver the mail directly, eliminating the usual 30 minutes delay we see in other implementations which do greylisting not in a selective way.

This can be achieved by changing the last lines of the example algorithm 4.1.1 to a slightly intelligent one shown in Figure 4.2.1. This, however, would mean that we allow all the senders (also spammers) not listed in DNSBLs to deliver mail without any delay (a more detailed analysis, outlined in Section 6.2, showed us that only about 82% of the senders were listed in DNSBLs). This leads us to further checks, mostly based on strict checking of the RFC, shown in Figure 4.1.

Algorithm 4.2.1 Basic Intelligent Greylisting Algorithm

```
if (ip, sender, recipient) ∈ DB then
  if whitelisted then
    return action=DUNNO
  else if waited > 30min then
    save (ip, whitelisted=1) into DB
    return action=DUNNO
  else
    return action=DEFER_IF_PERMIT
  end if
else
  if ip ∈ DNSBL then
    save (ip, sender, recipient, last_connect=NOW) into DB
    return action=DEFER_IF_PERMIT
  else
    save (ip, sender, recipient, last_connect=NOW, whitelisted=1) into DB
    return action=DUNNO
  end if
end if
```

- Does the name given in HELO match the reverse DNS of the client? ([Kle08, Section 4.1.1.1])
- Ist the name given in HELO a valid FQDN? ([Kle08, Section 4.1.1.1])
- Is the client allowed to send mail for this domain? (SPF)
- Does the reverse DNS look like one from a dialup host? While there is nothing wrong with a dialup host sending mail, this is quite unusual and many dialup hosts are infected computers working in spam botnets.

Figure 4.1: Further Checks for Intelligent Greylisting

Further checks of the domain (like checking it against RHSBLs like RFC-Ignorant [rfc] or the `bogus_mx` check in policyd-weight) are not neccessary according to the statistics done by Commtouch Software Online Labs [Lab]. Spammers mostly use domains owned by others and these usualy will not be in such lists at all.

However, we noticed that quite an amount of spam (about 16%) we got on our test server came from our own domains, more precisely spammers seem to use the same `user@domain.tld` combination for both, the sender and the recipient. That lead us to another check: is the sender equal to the recipient? One could argue that this is redundant to a SPF check if one has correct SPF records for his own domains but it is possible that one has to accept mails for domains where one cannot set these (no access to the DNS) or cannot provide reliable ones (many different people use this domain from different/changing sending hosts[1]). Additionally, it is possible to ommit the quite expensive (in terms of time) DNS queries for a SPF check if the SPF check is done last and it is obvious that the mail would be rejected before the SPF check is started.

[1]like it happens with `debian.org`, where every developer can send his mail from his own server

Chapter 5

bley

`bley` is our approach to create an intelligent greylisting service for Postfix. It is written in Python, using either PostgreSQL or MySQL as a database backend (theoreticaly it would work with every database which has a DB-API 2.0 [pyt] Python connector).

The software is split in three main parts:

- `PostfixPolicy` for communication with Postfix
- the database for saving the known senders
- `BleyWorker` for the actual checking of the senders

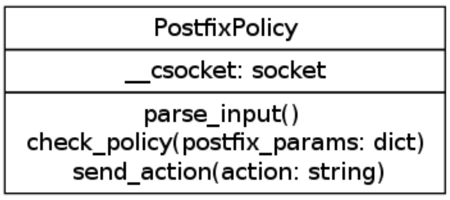

PostfixPolicy
__csocket: socket
parse_input()
check_policy(postfix_params: dict)
send_action(action: string)

Figure 5.1: PostfixPolicy Class

```
class ExamplePolicy (PostfixPolicy):
    def check_policy (postfix_params):
        if postfix_params['sender'] == 'spam@spam.org':
            self.send_action('REJECT')
        else:
            self.send_action('DUNNO')
```

Figure 5.2: ExamplePolicy based on PostfixPolicy

5.1 PostfixPolicy

To be able to communicate with Postfix over a TCP socket, we wrote the (very abstract) `PostfixPolicy` class (based on the Postfix SMTP Access Policy Delegation described in Section 4.1) which later should be inherited by our main application. It mainly consists of one socket and three functions:

- `parse_input` - read from the socket, parse `key=value` pairs Postfix sends and invoke `check_policy` with the read parameters when the empty line arrives.
- `check_policy` - just call `send_action('DUNNO')`, this one should be overridden in the application to do some actual policy checking.
- `send_action` - write the given action followed by an empty line to the socket, so Postfix can act accordingly.

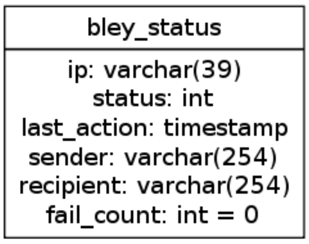
Every application that wants to use this class just needs an own class (we call it `ExamplePolicy` here) which will override the `check_policy` function with something useful (see Figure 5.2) and a short script that will create the sockets and pass them to a `ExamplePolicy` instance (see Figure 5.3).

With this simple code the newly created service will reject any mail coming from `spam@spam.org`.

```
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.bind(('localhost', 12345))
serversocket.listen()

while True:
    (clientsocket, address) = serversocket.accept()
    policy = ExamplePolicy (clientsocket)
    policy.parse_input()
```

Figure 5.3: ExamplePolicy start script


bley_status

ip: varchar(39)
status: int
last_action: timestamp
sender: varchar(254)
recipient: varchar(254)
fail_count: int = 0

Figure 5.4: bley Database Structure

5.2 The Database

The database (as shown in Figure 5.4) consists of only one table: `bley_status`. It contains the IP address (as a 39 character long string to support IPv6 addresses in the future), the sender and recipient e-mail addresses (each a 254 character long string, which is the maximum length of an e-mail address [Kle08, Section 4.5.3.1.3]), an integer for the status of the sender (0 = accepted, 1 = whitelisted, 2 = greylisted), a timestamp and an integer counter for the failed connection attempts (those before the greylisting period has ended).

5.2.1 BleyCleaner

From time to time the database needs some cleaning: hosts which did not connect for a long time can be purged safely. Hosts which did deliver mail succesfully are wiped after 40 days of inactivity, those which only tried to deliver (were greylisted and did not retry again) already after 10 days (the RFC suggests the last retry after 4-5 days [Kle08, Section 4.5.4.1]).

As the cleaning is just executing one SQL query and waiting for ten minutes afterwards, before executing the query again, it is run in a separate thread, which does this in an infinite loop. We decided not to run the cleaning into the usual new-mail handling, as it would have been executed too often there (which would only waste ressources without any further benefit).

5.3 BleyWorker

`BleyWorker` is our main class which runs as a thread and processes the incoming requests from Postfix. It extends the `PostfixPolicy` class from Section 5.1, overriding the `check_policy` function with our definition of an intelligent greylisting implementation:

1. If the mail is for either the `postmaster` or the `abuse` mailbox, accept it immediately ([Kle08, Section 4.5.1]).

2. Check the tuple "IP address, sender, recipient" in the database.

3. If the tuple was not found:

    a) Accept[1] if the IP address is found in a DNSWL (`check_dnswl`). Save the IP address, sender, recipient, current timestamp and status=1 (so the next attempt will be accepted directly) into the database.

    b) Reject[2] if IP address is found in a DNSBL (`check_dnsbl`). Save the IP address, sender, recipient, current timestamp and status=2 (greylisting started) into the database.

[1]In our case "accept" means action=DUNNO or action=PREPEND [posb]
[2]In our case "reject" means a temporary reject issued with action=DEFER_IF_PERMIT [posb]

c) Reject if the sum of `check_helo+check_dyn+check_sender`[3] is more than 2. Save the IP address, sender, recipient, current timestamp and status=2 (greylisting started) into the database.

d) Reject if SPF check fails. Save the IP address, sender, recipient, current timestamp and status=2 (greylisting started) into the database.

e) Accept if none of the previous tests led to a reject. Save the IP address, sender, recipient, current timestamp and status=0 (so the next attempt will be accepted directly) into the database.

If the tuple was found:

a) Accept the mail if it belongs to a sender with either status 0 (previously accepted) or 1 (previously found in DNSWL). Update the timestamp in the database (so the entry won't be purged due to inactivity).

b) Accept the mail if it belongs to a sender with status 2 (greylisted) and the first connection is older than the greylisting period (incl. the possible penalty based on the failure counter 5.3.1) or the maximum greylisting time. Update status to 0 and the timestamp to the current time in the database.

c) Reject the mail if it belongs to a sender with status 2 (greylisted) and the first connection was less than the greylisting period (incl. the possible penalty based on the failure counter) ago. Increase the failure counter in the database by one.

This flow is also shown in Figure 5.5.

[3]The checks are explained in more detail in Section 5.4

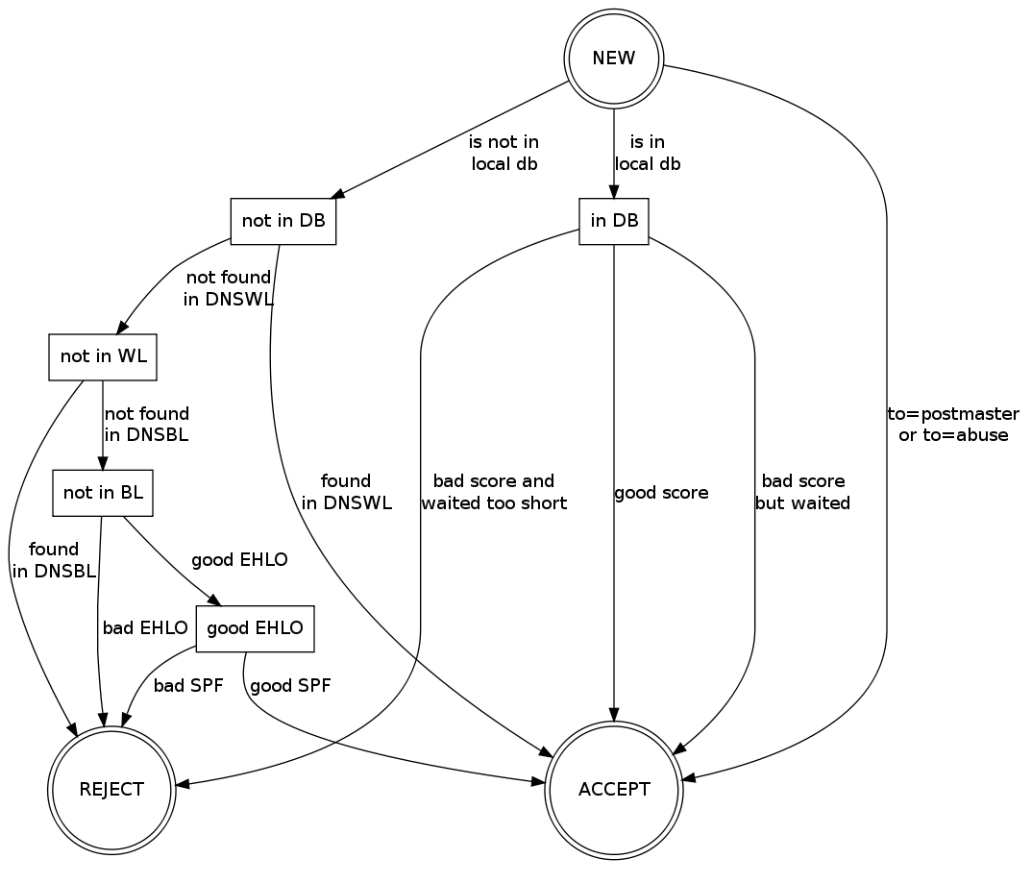


Figure 5.5: bley Flowchart

5.3.1 Penalties

The penalty used in `bley` is computed as

$$fail\_count * greylist\_penalty$$

(greylist_penalty defaults to 10 minutes). We decided to set the maximum penalty to 12 hours, so the senders cannot lock themself out by increasing the panalty with each failed delivery attempt. The 12 hours were chosen to allow senders connecting from dialup hosts which often change the IP address after 24 hours.

5.4 The Checks in Detail

5.4.1 check_dnswl / check_dnsbl

Both checks work exactly the same way: the IP address is looked up in the configured DNSWLs and DNSBLs until it is found in at least one list (this can be configured via the `dnswl_threshold` and `dnsbl_threshold` settings).

5.4.2 check_dyn_host

Dynamic hosts are known to be infected personal computers acting as spam senders. As we cannot check for the computer being infected we have to check whether it is connected to a dialup line. There are DNSBLs containing exactly this information but we decided to analyze the hostname instead to save DNS queries.

For the analysis we have designed two regular expressions – one for hostname parts that most likely belong to a static host and the other for those which belong to a dynamic one:

1. `colo|dedi|hosting|mail|mx[^$]|smtp|static`

2. `\.bb\.|broadband|cable|dial|dip|dsl|dyn|gprs`
   `|ppp|umts|wimax|wwan`
   `|[0-9]{1,3}[.-][0-9]{1,3}[.-][0-9]{1,3}[.-][0-9]{1,3}`

Only if a hostname does **not** match the first and does match the second one, it is considered a dynamic host and the check returns 1 (which will be added to the overall score of the sender). In all other cases, the check returns 0.

5.4.3 check_helo

The RFC demands the name in HELO to be either the FQDN of the host (e.g. `host.domain.tld`), or, if the FQDN is unknown, the IP address in square brackets (e.g. `[123.45.67.89]`) ([Kle08, Section 4.1.1.1]). Based on this, we define a good HELO (the check returns 0) as `helo_name==client_name`. However, sometimes hosts greet with `mail.domain.tld`, while their `client_name` is something like `funnyname.domain.tld`. In order to not declassify such hosts as spammers, the check returns 1 in this case. If the `helo_name` contains the IP address in square brackets the check returns 1 too. All other hosts, especially those which greet with a non fully qualified or not resolving domain name, get a score of 2.

5.4.4 check_sender_eq_recipient

As it is noted in Section 4.2, we noticed that a lot of spam is comming in with the same sender and recipient e-mail addresses. We check this directly instead of relying ond SPF, so we are able to ommit the SPF check and save ressources for resolving DNS records.

5.4.5 check_spf

For checking SPF we use the existing `pyspf` library [Gat]. The actual check is very simple: the spf module checks the tuple client_address, sender, helo_name. If it returns either "Fail" or "SoftFail", the check returns 1, if it returns "Pass", the check returns 0. For every other result the check is repeated with the built-in best-guess algorithm (a regular SPF lookup with the SPF record set to `v=spf1 a/24 mx/24 ptr`) and returns the same values as before.

5.4.6 Summary

Using these checks we are able to find most of the hosts that are either known to be sending spam or their behavior is similar to the one of spammers. Based on this identification, we are able to decide whether we want to accept a mail from the host immediately, or greylist it and let the host retry the delivery later.

Chapter 6

Evaluation

We started with IP address only matching. But after we have seen not satisfying results we have switched to matching of the tuple IP address, sender address and recipient address. We will present both sets of results here.

6.1 IP Address only Matching

While running the service with IP address only matching on our test server, we had to check 19596 mails in a 24h frame. Only 140 (~1%) mails were finally accepted and delivered to users mailboxes. However, we have to admit that 4310 mails were rejected by Postfix directly because there were no corresponding mailboxex for the addresses the senders tried to deliver to, so the effective reject rate of `bley` was only 77% (15146) mails.

The reject reasons of the 15146 mails are shown in Table 6.1. The accept ones for the 4450 mails are shown in Table 6.2.

Especially the high number of accepts after correct greylisting made us suspicious. Looking at the logs revealed that this was caused by new mails (with different sender/recipient) originating from a known host, which our software identified as successful retransmissions after greylisting.

no. of mails	in %	reject reason
6309	32.20%	greylisting active, did not wait long enough
7659	39.08%	sender was found in a DNSBL (greylisting started)
1018	5.19%	sender used a not RFC-compliant HELO
148	0.76%	sender connected from a dialup host, used same address as sender and recipient or HELO contained not 100% correct information (the sum of these three tests was >= 2)
12	0.06%	SPF check failed

Table 6.1: bley Reject Statistics with IP Address only Matching

no. of mails	in %	accept reason
11	0.06%	sender unknown but found in DNSWL
119	0.61%	sender unknown but no bad checks
43	0.22%	sender known (previously found in DNSWL)
853	4.35%	accepted after greylisting
3424	17.47%	accepted after previous accept

Table 6.2: bley Accept Statistics with IP Address only Matching

6.2 IP Address, Sender, Recipient Matching

After switching to the stricter matching we again analyzed a 24h frame now containing 18843 delivery attempts, 97% (18304) of which got refused by `bley`.

The much lower numbers of hits in the local database and the much higher numbers of DNSBL and bad HELO hits in Table 6.3 could be explained by the fact that with the new matching the senders were not found in the database (they tried another sender/recipient combination before) and thus were checked again. In Table 6.4 we discern a higher "unknown sender" rate due to the stricter matching. Both "accept after greylisting" and "accept after previous accept" drop significantly for the same reason.

Figure 6.1 shows accepts and rejects for nine days of `bley` use.

no. of mails	in %	reject reason
496	2.63%	greylisting active, did not wait long enough
15444	81.96%	sender was found in a DNSBL (greylisting started)
2178	11.56%	sender used a not RFC-compliant HELO
142	0.75%	sender connected from a dialup host, used same address as sender and recipient or HELO contained not 100% correct information (the sum of these three tests was >= 2)
44	0.23%	SPF check failed

Table 6.3: bley Reject Statistics with IP Address, Sender and Recipient Matching

no. of mails	in %	accept reason
69	0.37%	sender unknown but found in DNSWL
423	2.24%	sender unknown but no bad checks
15	0.08%	sender known (previously found in DNSWL)
20	0.11%	accepted after greylisting
12	0.06%	accepted after previous accept

Table 6.4: bley Accept Statistics with IP Address, Sender and Recipient Matching

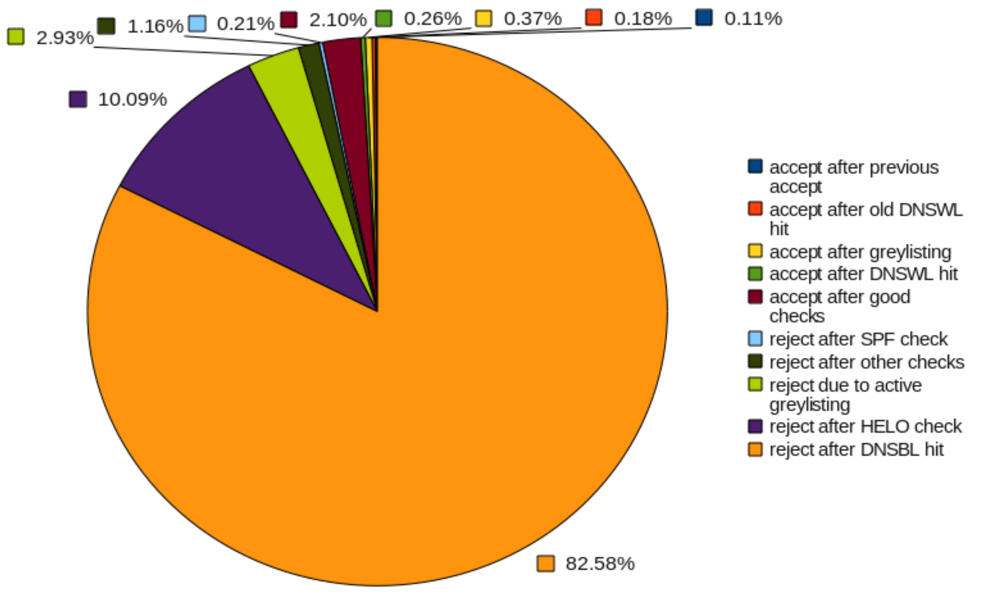


Figure 6.1: Piechart for Rejects and Accepts

6.3 False Positives

To get a rough idea of how many mails we identified as false positive, we searched our logfiles for mails that only got accepted after an initial delay. While there were some mails in that list, we could only find two real false positives - one from `listia.com` (an auction platform) and one from `zyb.com` (a mobile synchronization service).

The `listia.com` mailservers have reverse DNS records in the form `123-45-67-89.static.cloud-ips.com` but wrongly greet with `listaX` (`X` being a number) in the HELO, thus failing our HELO check. The `zyb.com` ones were greeting fine but had no reverse DNS at all thus again failing our HELO check. Both mails were accepted on the fourth attempt, about an hour after the first one. This was caused by retries after 10 and 30 minutes, both causing a 10 minutes penalty.

All other mail was real spam (verified by looking into the actual mail or by the fact it was refused due to a non-existent mailbox).

6.4 Comparison with policyd-weight

As our previous setup used `policyd-weight` in its default configuration, we used this for our benchmarking. The aim was to provide as strict checking as `policyd-weight` but to eliminate the possibility of loosing mail.

As mentioned earlier `policyd-weight` accepted about 500 mails out of 20.000 delivery attempts which results a reject rate of about 97.5%. After final tuning `bley` rejected about 97% of the mails and achieved nearly the same rate as `policyd-weight`. However, the main improvement over `policyd-weight` is the fact that we theoretically can not loose any mail: the mail from `tut.by` was accepted without problems, other false positives were accepted after a delay of 30-60 minutes (as long the sending server was acting accordingly to the RFC).

No. of hosts	% of hosts	HELO looked like
6992	43.98%	not a FQDN (did not contain any dots)
7221	45.42%	a FQDN (contained dots and ended with a TLD)
1048	6.59%	kind of FQDN (contained dots but did not end with a valid TLD)
638	4.01%	an IP address

Table 6.5: HELO Hostname Statistics

6.5 Interesting Findings

While designing the HELO check, based on the fact that the RFC demands a correct FQDN in HELO, we assumed that infected hosts would just use their local hostname (something like `MYPC` or `WINDOWSXP100`) there.

However, our logs revealed quite different results (based on 15899 rejected hosts), as shown in Table 6.5. This means that relying on a valid HELO is not sufficient anymore and other checks might be needed in the future to detect spammers.

Chapter 7

Conclusion

Our intention was to design a Pre-MX spam filter which neither faces the problem of lost mail due to false-positives, nor delays valid mail before delivering it to the user. The presented existing solutions have at least one of these problems and therefore are not usable in a corporate environment, where users want their mail fast and reliable (they do not want to wait for mail and they especially do not want to loose mail).

We designed a Postfix policy daemon which combines black- and greylisting with various static checks to a selective greylisting. The resulting implementation is working and ready for productive use. It has a hit rate of about 97% with only a few false positives which are delayed for 30-60 minutes.

7.1 Future Work

While the current implementation is fully working there is still room for improvements. One could try to make the sender matching more intelligent, i.e. by allowing all mail from one IP address or a whole subnet after a couple of succesful deliveries or by penalting all incoming connections from an IP address instead of penalting only the IP address, sender, recipient tuple.

Another interesting addition would be the reporting of possible spammers to DNSBL providers, so users of other software can benefit from our findings. This, however, might be complicated as we have no real evidence that the not delivered mail was spam, so the DNSBL providers cannot check our reports and have to rely on our checks.

Bibliography

[Gat] Stuart D. Gathman. pyspf. `http://pypi.python.org/pypi/pyspf`. [Online; accessed 08-September-2009].

[gld] gld. `http://www.gasmi.net/gld.html`. [Online; accessed 14-October-2009].

[gps] gps. `http://mimo.gn.apc.org/gps/`. [Online; accessed 15-October-2009].

[gro] gross. `http://code.google.com/p/gross/`. [Online; accessed 15-October-2009].

[GS98] Simson Garfinkel and Alan Schwartz. *Stopping Spam*. O'Reilly, first edition, 1998.

[Har03] Evan Harris. The Next Step in the Spam Control War: Greylisting. `http://www.greylisting.org/articles/whitepaper.shtml`, 2003. [Online; accessed 7-August-2009].

[Kle08] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Proposed Standard), October 2008.

[Lab] Commtouch Software Online Labs. Top 10 Spam-Sending Domains. `http://www.commtouch.com/sites/all/themes/commtouch/iframes/SpamLab.html`. [Online; accessed 4-October-2009].

[Lie06] Peter Lieven. Pre-MX Spam Filtering with Adaptive Greylisting Based on Retry Patterns. Bachelor's thesis, August 2006.

[pol] policyd-weight. `http://www.policyd-weight.org/`. [Online; accessed 16-September-2009].

[posa] Policyd. `http://www.policyd.org/`. [Online; accessed 18-October-2009].

[posb] Postfix manual - access. `http://www.postfix.org/access.5.html`. [Online; accessed 7-August-2009].

[posc] Postfix SMTP Access Policy Delegation. `http://www.postfix.org/SMTPD_POLICY_README.html`. [Online; accessed 7-August-2009].

[posd] postgrey. `http://postgrey.schweikert.ch/`. [Online; accessed 12-October-2009].

[pyt] Python Database API Specification v2.0. `http://www.python.org/dev/peps/pep-0249/`. [Online; accessed 01-September-2009].

[rfc] rfc-ignorant. `http://rfc-ignorant.org/`. [Online; accessed 7-October-2009].

[Sab08] Sabine Sobola. Spam - eine Datenschutzfalle? *Linux Technical Review*, 7, 2008.

[Sie08] Paul Ferdinand Siegert. *Die Geschichte der E-Mail.* transcript, 2008.

[spa] SpamAssassin. `http://spamassassin.apache.org/`. [Online; accessed 06-October-2009].

[spf] SPF: Best guess record. `http://www.openspf.org/FAQ/Best_guess_record`. [Online; accessed 16-October-2009].

[sql] sqlgrey. `http://sqlgrey.sourceforge.net/`. [Online; accessed 12-October-2009].

[SZ07] Daniel AJ Sokolov and Peter-Michael Ziegler. Spamhaus.org setzt Österreichs Domainverwaltung unter Druck. `http://www.heise.de/newsticker/meldung/91417`, 2007. [Online; accessed 01-October-2009].

[tum] tumgreyspf. `http://www.tummy.com/Community/software/tumgreyspf/`. [Online; accessed 13-October-2009].

[Wik09] Wikipedia. Greylisting. `http://de.wikipedia.org/wiki/Greylisting`, 2009. [Online; accessed 5-October-2009].

[Woo05] David Woodhouses. Why you shouldn't jump on the SPF bandwagon. `http://david.woodhou.se/why-not-spf.html`, 2005. [Online; accessed 16-October-2009].

[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.

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, 6. November 2009 Evgeni Golov