Combining SMTP and HTTP tar pits to proactively reduce spam

Tobias Eggendorfer
ITIS e. V. Institut für Technik Intelligenter Systeme
An-Institut der Universität der Bundeswehr
(University of Federal Armed Forces Neubiberg)
Werner-Heisenberg-Weg 39
85579 Neubiberg, Germany

Jörg Keller
FernUniversität in Hagen
Fakultät Mathematik und Informatik
Lehrgebiet Parallelität & VLSI
58084 Hagen, Germany

Abstract – Unsolicited commercial emails (UCE, spam) are currently being fought using reactive methods, filtering being the most common. Reacting means to be always one step behind. So the focus on fighting spam should be on prevention. Current proposals focus on fixing SMTP's lack of authentication, but introduce two major problems: First of all current attempts break existing functionality and, second, it seems to be hardly possible to enforce a world wide change of SMTP.

Therefore other preventive measures should be implemented. The most promising approach is to prevent spammers from collecting email addresses. Several proposals show ways to obfuscate addresses on web pages, another proposal was to create HTTP-tar pits in order to catch harvesters.

Our tests with real world harvesters showed room to improve those tar pits by combining them with SMTP tar pits. We report on the success of our experiments with this combination.

Keywords – Spam, SMTP, HTTP, tar pit, proactive anti-spam-measures

1 Current anti spam measures

1.1 Reactive methods

Currently, most relevant methods to reduce the amount of UCE in a user's inbox rely on some kind of filtering. The most basic approach is probably blacklisting, i. e. each incoming request's IP-address on a SMTP-server is tested against a list of known spamming hosts. Although, when invented back in the late 1990s, it supported the demand of shutting off so called open relays, it often has heavy side effects: Almost all big email providers have already been blacklisted on at least some of the widely available blacklists.

Other solutions are content filters applied to the header and / or the body of a mail message. Filtering is based on a "bad-word-list". Later improvements included scoring-mechanisms to weight words. Those filters require a lot of fine-tuning and maintenance: Spammers are reported to register mail accounts with online services known to have spam filtering and to test their spam against those filters. This leads to a permanent "one-step-behind" situation for filters, no matter how advanced content-filtering becomes [GAN05].

Another still reactive way to reduce spam is greylisting, i. e. forcing the sending MTA of a message to resend it after a short time. As of now, this solution is quite potent, as most spam is sent through so called zombies, usually Windows-PCs infected with some worms. Those worms contain their own SMTP-engine, which is usually quite simple. Most of them are still unable to handle the temporary unavailable condition

used in greylisting and therefore consider this condition as an error condition and stop delivery. Greylisting has two major disadvantages: It slows email communication down and it is likely to be useless when those worms will implement better SMTP-engines, which is to be expected soon.

1.2 Modifying SMTP

The above touched disadvantages of reactive anti-spam methods brought the discussion on fixing the real cause for spam: SMTP lacks authentication. So the key approach is to implement some kind of authentication and authorisation. Beside some side effects seen on current methods, like preventing intended mail-forwarders, the real problem is to enforce the modified standards world-wide. This is not only an organisational problem resulting from competing standards and companies trying to win their share of market by patenting their solutions, but also and mainly due to the broad, not centrally maintained base of billions of SMTP-clients and millions of servers in the internet. Back at ARPANET times it was possible to change the standard to IP almost over night, but the internet has

grown. There are still thousands of open relays out there, although open relays are deprecated and black-listed since at least ten years. Considering this, any change to SMTP would need at least another ten years to be broadly available.

1.3 Preventing harvesters

Considering this, the search for new solutions has been opened. Probably, the most promising is to prevent spammers from collecting email addresses, because spammers currently only use two relevant ways to collect addresses: One is by installing worms and trojans on computers and have them read local addressbooks, emails or even all files, collect email addresses found there and spam to them. There is an obvious solution to this: Have users install decent and safe operating systems, virus scanners and personal firewalls and protect their PCs with external firewalls and application level malware filters.

The other source of email addresses for spammers is the internet, most notably the www and the usenet. There, they collect email addresses using spidering
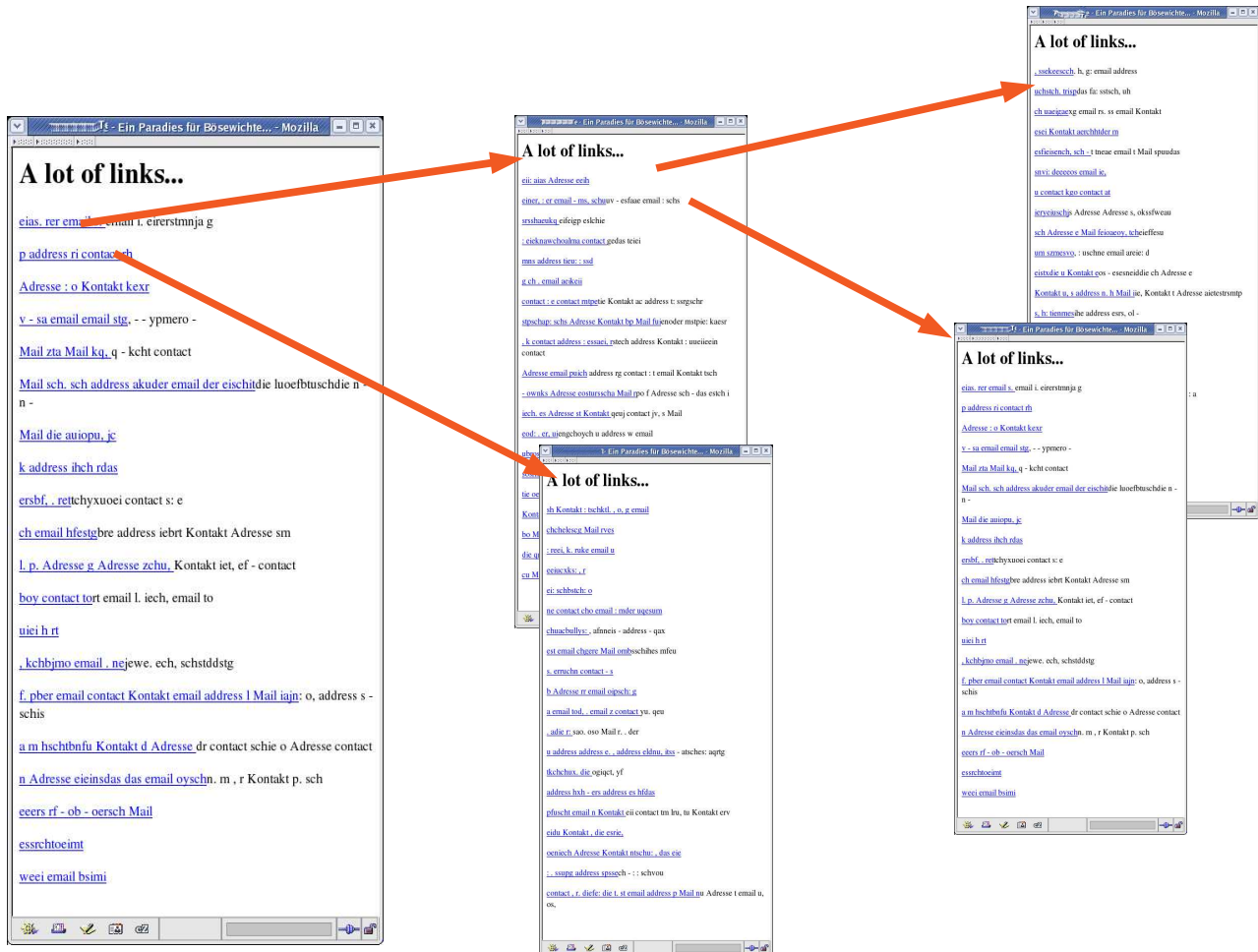

Figure 1 Illustration of how a tar pit pollutes a harvester's list of links to visit

technology known from search engines. The programmes doing this job are called "harvesters".

Again there are some ways how to handle them: One is to obfuscate email addresses, so they would not be recognised by harvesters. In [EGG05a] the author suggested several solutions, that are both compatible to any installed browser and barrier free, and proved their effectiveness in a still ongoing real world experiment [EGG05c]. Later, in [EGG05m], the author proposed an automated solution to dynamically obfuscate email addresses published on the web, thereby solving the problem to modify or redo existing webpages.

2 Tar pits

The other approach to bar harvesters from collecting mail addresses is to trap them in a tar pit. The basic concept is to create random webpages containing links to the same or other tar pits. This pollutes the list of webpages-to-visit the harvester has, and keeps the harvester returning and finally staying in the tar pit. As soon as the harvester is caught, all of its resources are attracted to the tar pit, thereby preventing it to visit any other webpage and collect email addresses there.

The basic concept is rather simple: On each page of the tar pit, it offers a certain amount of links. The tar pit we used for testing offered an average of 15 links per page. A harvester coming across the tar pit will then extract all links he found from this web page and add those links to his list of webpages to visit.

On its first visit, 15 links were added. The harvester will then go through the list and visit each page listed therein. As soon as it calls an URL published by the tar pit earlier, it will receive another 15 links from the tar pit – all of them pointing back to the tar pit. The basic principle is also shown in Figure 1.

2.1 HTTP Tar Pit Requirements

Although the concept sounds simple, setting up a functional and safe tar pit is not as easy as it might seem at first glance: First, "honest" spiders, such as Google-Bot, should not be trapped. Second: If the tar pit publishes links back to itself, they need to be different, i.e. the harvester should consider them new. And last but not least the tar pit needs to make sure it is not hit by a denial of service condition if a harvester runs in circles through the site.

2.2 Do not catch good spiders

The first requirement, safe guarding the good, is easily implemented: Any decent spider should obey the robots.txt standard [HEM03], [W3CAPPB]. Excluding any spider from the page would do. As of now, harvesters ignore robots.txt. From the harvester's developers point of view this is a logical decision to find even more email addresses.

Practical experiments proved this assumption to be correct. Both downloaded harvesters and those visiting a test tar pit showed this behaviour.

If in the future harvesters learn not to ignore the robots.txt-standard, this would be a positive result of tar pits: Hiding email addresses on web pages becomes as easy as hiding those pages from robots with a robots.txt file.

2.3 Generate different links pointing to the same file

The next step was to generate new pages containing links to the page itself using different URLs. In the test setup, filenames might have between 5 and 30 characters each and there is a choice of different filename extensions like ".htm", ".html", ".shtml" or ".shtm".

Using symlinks on the server with every possible filename and extension would generate approximately $4 \cdot 36^{32} \approx 2,5 \cdot 10^{50}$ links in the filesystem. Despite the huge amount of space needed only for the symlinks, most filesystems are unable to handle so many files in one directory in an efficient manner.

Therefore, the web server should redirect any request received to the tar pit script. There are some ways to do so. One is mod_rewrite [TOF05], [CRA05], another uses aliases and the simplest is to use an ErrorDocument. The later has the advantage to be available for almost anyone: Almost any of the webspace providers offering an opportunity to run PHP on their servers also allow the ErrorDocument directive within a ".htaccess" environment.

The required ErrorDocument-directive is:

```
ErrorDocument 404 /spamfight/index.php
```

The only thing to take care of with this approach is to replace the 404-HTTP-Header with a 200-OK-HTTP-Header. But this is easily done: If a script sends a HTTP-Status, the web server will use this one.

The alternative is using aliases. This usually requires access to the main web server configuration:

```
<Directory /spamfight>

  AliasMatch ^/spamfight/[A-Za-z0-9]+\.
[A-Za-z0-9]+$ /spamfight/index.php

</Directory>
```

mod_rewrite offers amazing possibilities when mapping different URLs to a file. [CRA05] even demonstrated how to implement a content-management-system based on mod_rewrite and some server side includes. But for the purpose of a tar pit, mod_rewrite has a serious disadvantage: It sends a "Location moved"-HTTP-Response. So a harvester might see it has been redirected to a location it has visited before. It might then recognise the tar pit.

2.4 Avoid Denial of Service

The most difficult task is to avoid a denial of service condition: If the tar pit publishes 20 links per page, the harvester will add those links to his list of pages to visit. On each of those links visited, it will receive yet another 20 links. Within a short time, the harvester has some millions of links in his list all pointing to the tar pit. Counting in rounds, the list grows exponentially. A round is defined by induction: The first round is the first visit to the tar pit and each subsequent round is the moment, when all links collected during the previous round have been visited.

If the harvester supports parallel spidering and is running on multiple machines, it might have enough bandwidth to pull the whole server down. If the server is only serving the tar pit, this does not matter – but if the tar pit is run on an a server also used for other purposes, the "real" pages become undeliverable.

To avoid those problems, the number of instances of the tar pit running should be limited to a maximum. But using PHP as programming language, there is no simple way to determine how many instances of the same script are currently running: ps -ef e.g. will not list those scripts.

Implementing some kind of interprocess communication is needed. The easiest mean are semaphores. Binary semaphores are commonly used to programme mutually exclusive tasks: A semaphore S is created. Each task about to enter its critical section first does a P(S)-operation. This action is atomic, i.e. the operating system's scheduler won't interrupt it. P(S) decrements the semaphore by one if the semaphore is still greater than zero. If not, it will send the requiring process to waiting state and bring it back to the ready state as soon as the semaphore is increased again [TAN02].

For the tar pit, the whole tar pit script is enclosed with P(S) and V(S). Doing so, the maximum of parallel processes is limited by the semaphore's start value, e.g. if 20 processes are allowed to run simultaneously, the semaphore's initial value is 20.

For a more detailed explanation of the tar pit, see also [EGG05o].

3 Optimisation of the tar pit

Although the basic tar pit works in real-world experiments, tests with off-the-shelf harvesters available in the web gave some hints on how to modify the tar pit to be even more effective.

Most harvesters implement some kind of progress meter by listing the last email addresses found. The basic tar pit did not deliver any email addresses. A human operator could then realise that his harvester got caught by a tar pit. He could even blacklist the tar pit and inform other spammers of its existence.

To have harvesters stick longer to the tar pit, the tar pit should offer some email addresses to the harvester. But those addresses need to be existent: Random addresses under random domains might easily contain existing email addresses belonging to someone else who then will receive spam.

The other downside to random addresses is the so called bounce spam. This is spam sent to a non-existent address seeming to originate from another domain or email address than the one the spammer has. For each undeliverable spam message an error message is created and sent to the supposed sender's address, and, if it is also non-existent, to the postmaster of his domain.

Considering this, email addresses published by the tar pit should exist and a mail server should accept messages to them. However, setting up a plain mail server to accept this spam and store it in some kind of database would thwart the efforts in setting up the http tar pit.

Instead, it would be nice to again trap spammers in a tar pit when they try to deliver mail to the addresses collected from the webpage.

4 SMTP tar pit

To achieve this, a SMTP tar pit would be needed. There are different kinds of SMTP tar pits available in the net, as quite a lot of anti-spammers use them, although none of them has been used to our knowledge in combination with the new HTTP tar pit.

SMTP tar pits usually accept mails, but they answer incoming SMTP connections very slowly and thereby waste the time of the sender. There are two basic concepts in slowing down the connection: One is to slow down the connection on the TCP/IP-Level by using minimum frame sizes, sending each frame on its own etc. [LIoA]. The other, more common concept, is to use application level slow downs. To do so, SMTP continuation lines are used [RFC0821]: Each request is answered with dozens of response lines. Those lines are usually sent with short delays in between, adding an extra slowdown.

1:	220 mail.example.com ESMTP Postfix
2:	*EHLO*
3:	250-mail.example.com
4:	250-PIPELINING
5:	250-SIZE 10240000
6:	250-VRFY
7:	250-ETRN
8:	250 8BITMIME
9:	*MAIL FROM:<user@example.org>*
10:	250 Ok
11:	*RCPT TO:<someone@example.com>*
12:	250 Ok
13:	*DATA*
14:	354 End data with <CR><LF>.<CR><LF>
	... Data section goes here ...
15:	250 Ok: queued as DD71B1051A3

Figure 2 An example SMTP-dialogue (client input in italics)

Figure 2 shows an example SMTP dialogue. Lines 2 to 8 are an example for continuation lines: After sending the status code a dash is written. This dash indicates, that additional lines will be sent by the server. Lines 10, 12 and 15 are single-line answers.

The SMTP tar pit will add a lot of bogus continuation lines, forcing the client to wait.

Doing so, bulk mailers should be slowed down on each connection they have to a tar pit. But set up on their own, SMTP tar pits are quite ineffective: They only slow down one connection to a certain mail server at a time, which usually has almost no impact, as one server is able to accept many mails during one connection and most bulk mailers are capable of connecting to many mail servers in parallel.

This turns SMTP tar pits into a not very effective anti spam measure [EGG06].

In contrast, the proposed HTTP tar pit will block even a massively distributed harvester within a short time by inserting an exponentially growing amount of links to itself into the harvester's webpages-to-visit list.

5 Combining HTTP- and SMTP-tar-pit

Combined with a HTTP tar pit, a SMTP tar pit could serve as a MTA for the addresses published by the HTTP tar pit. Although the SMTP tar pit has virtually no effect on the performance of a spammer's bulk mailer, it would turn the HTTP tar pit even more effective by adding an additional level of invisibility to it.

5.1 Implementation

5.1.1 Modification of the HTTP tar pit

To combine the HTTP tar pit and a SMTP tar pit, the existing HTTP tar pit has been modified to deliver email addresses under certain domains. As those generated email addresses are shown to the harvester's operator, they should look like real addresses consisting of a first and a family name.

To achieve this, a list of first names and family names has been collected from the web. From each of those lists, 150 names were randomly selected. While running, the tar pit generates an email address by randomly selecting a first and a family name, joining them with a dot and appending one out of the configured domain names.

To make it even harder to identify the tar pit, the amount of links and email addresses published has been changed to random, additional random text is inserted and a few new domains have been registered.

5.1.2 Selection and adaption of an existing SMTP tar pit

As SMTP tar pits are commonly used in the internet, available tar pits were evaluated. One of the key requirements was the ability to easily modify the source code to later store collected emails in a database. Other requirements included, but were not limited to, security considerations and compatibility to the

existing platform without the need to install too much additional software.

The most promising candidates were [REHWWWc], [DON04a] and [GRO06]. [REHWWWc] required the installation of Java on the tar pit and had only very few configuration options. [DON04a] is a wrapper configured to run between a local MTA and the internet. This offers the opportunity to run the tar pit on a real MTA delivering mail to real users. However, in the setup of the existing tar pit, it would have required to install a MTA. To avoid possible security risks introduced by each additional line of code and due to the fact that the wrapper is written in C with all the risks introduced by C's memory management, the decision was to not use it. Furthermore the wrapper requires an explicit black list of IP addresses to trap in its tar pit, which would increase the time and effort to configure it.

[GRO06] instead is written in Perl, which reduces the risk of memory leaks and buffer overflows. Also, a Perl interpreter was already been installed on the tar pit, so no additional software was needed. Furthermore, the programme is very configurable, offering almost all options needed. The well-documented source code has then been reviewed for potential security problems, where only a minor enhancement was to be made.

By default, smtarpit refuses to accept mail with a 500-SMTP-error-code after having wasted the bulkmailer's time. This refusal might finally lead to bounce spam. To avoid this, the tar pit has been modified to either return a "temporarily unavailable" status or accept the message, with a probability of acceptance of 70%. So it is very likely that after some attempts a message could be delivered – after only ten attempts, the probability reaches 99,9994%. The standard configuration for sendmail e. g. would retry delivery for five days every thirty minutes resulting in a non-delivery probability of $3.2 * 10^{-126}$. This is considered to be acceptable.

5.2 Real world experiment

To test the efficiency of the combined HTTP and SMTP tar pit, hidden links to the tar pit were published in the internet in co-operation with some well frequented webpages.

The HTTP tar pit already proved its efficiency: It kept some harvesters returning for more than 20'000 visits, blocking each of them up for one or more days. It is very likely that their operator interrupted them because no email addresses were delivered.

With the combined tar pit, this shortcoming has been resolved: four weeks after the installation of the tar pit, one harvester stayed for seven days and more than 401'000 visits. This is a daily average of 57'285 visits. This harvester was run on a rented server at a Germany based web service provider, who confirmed by phone that he cancelled the hosting contract due to too many spam complaints originating from this server's IP on the same day the harvesting stopped.

By analysing the tar pit's log file by IP addresses, we identified the next two of top three visiting IPs: The second requested more than 94'400 pages within 24 hours, the third 32'197 pages. Both of them used dynamic IP addresses, i.e. IPs that will change after a maximum of 24 hours.

Looking at the top visiting harvesters' user agent, i.e. the identification string a browser sends, we found one of them to be quite unique and therefore supposed it to be used only by one installation of the harvester. It might however be that this user agent is specific for this harvester and not for one specific installation of it. So those numbers are less accurate than those based on the IP, but, due to the usage of dynamic IPs we had to identify something else to determine returning harvesters after they changed IP.

Based on its user agent, the harvester that counted for 94'400 visits within 24 hours from one IP is supposed to have spent a total of more than 537'000 visits in 20 days, this is a daily average of 26'850 visits.

Those results went far beyond the expectations we had when combining the SMTP and HTTP tar pit to increase the HTTP tar pits efficiency: We accounted for 20 times as many visits to the tar pit than on a standalone HTTP tar pit.

The SMTP tar pit also attracted some spammers: Within four weeks, we counted approximately 3000 connection attempts. Compared to the HTTP tar pit's numbers, this looks rather inefficient, but it is in good accordance to our theories: During one SMTP session, all messages from one client to this server might be transferred. Therefore much fewer connections are required for SMTP than for HTTP. We also found our theories concerning tar pit aware bulkmailers [EGG06] to be correct: Most connections lasted for less than a second.

Considering those two facts, the SMTP tar pit performed well and served its purpose to further obfuscate the HTTP tar pit.

6 Conclusion

Combining a HTTP tar pit with a SMTP tar pit increases the efficiency of the tar pit because the operator of a harvester might watch his harvester sucessfully collecting email addresses. This increases the time the harvester stayed in the tar pit. In our test setup, harvesters were caught for up to three weeks and spent 20 times more visits than to a simple HTTP tar pit.

Due to the huge amount of tar pitted email addresses the spammer collected, a welcome side effect is the delay of a later spam run by the SMTP tar pit, although this effect is reduced due to more and more tar pit aware bulkmailers.

The combination of both tar pits had a clear impact on the efficiency of the HTTP tar pit. The SMTP tar pit does not serve as first line of defense, as most other SMTP tar pits currently do, but supports the HTTP tar pit. Thereby the disadvantages inherent to the system of a SMTP tar pit have been overcome.

Furthermore, both tar pits were programmed with a focus on security, therefore both should not increase the risk of break ins into the tar pit server. A well documented configuration and installation process offers the possibility to easily implement the combined tar pit by any system administrator.

References

[CRA05]   Crane, Aaron, mod_rewrite as Business Logic: A Case Study of The Register presented at , Stuttgart, 2005

[DON04a]   Donnerhacke, Lutz, Teergrubing Wrapperonline at http://www.iks-jena.de/mitarb/lutz/usenet/antispam.html, 2004

[EGG05a]   Eggendorfer, Tobias, Methoden der präventiven Spambekämpfung im Internet, Masterthesis at Fernuniversität in Hagen, München, Hagen, 2005

[EGG05c]   Eggendorfer, Tobias, Spam proof homepage design. Methods and results of an ongoing study presented at , Stuttgart, 2005

[EGG05m]   Eggendorfer, Tobias; Keller, Jörg, Preventing Spam by Dynamically Obfuscating Email-Addresses presented at CNIS 2005, Phoenix, 2005

[EGG05o]   Eggendorfer, Tobias, Stopping Spammers' Harvesters using a HTTP tar pit presented at AUUG 2005, Sydney, 2005

[EGG06]   Eggendorfer, Tobias, Comparing SMTP and HTTP tar pits in their efficiency as an anti-spam-measure presented at Spam Conference 2006, Cambridge, MA, 2006

[GAN05]   Gansterer, Wilfried et. al., Anti-spam methods - state of the artin: , Institute of Distributed and Multimedia Systems, University of Vienna, 2005

[GRO06]   Grosse, Paul, SMTarPit v0.6.0online at http://www.fresh.files2.serveftp.net/smtarpit/index.html, 2006

[HEM03]   Hemenway, Kevin, Calishain, Tara, Spidering Hacks. 100 Industrial-Strength Tips & Tools, O'Reilly, Sebastopol, 2003

[LIoA]   Li, Kang et al., Resisting Spam Delivery by TCP Dampingin: , University of Georgia, Athens, GA, o. A.

[REHWWWc]   Rehbein, Daniel, Mailvernichter. Ein einfacher Mailserveronline at http://www.bahnhof-hamburg.de/mailserver.html, 2006

[RFC0821]   Postel, Jonathan B., Simple Mail Transfer Protocolonline at http://www.ietf.org/rfc/rfc0821.txt (u.a.), 1982

[TAN02]   Tanenbaum, Andrew S., Moderne Betriebssysteme, Pearson Studium, München, 2002

[TOF05]   Toftum, Mads, Apache mod_rewrite, the Swiss Army Knife of URL manipulation presented at , Stuttgart, 2005

[W3CAPPB] W3C, W3C Recommendations. Appendix B: Performance, Implementation and Designonline at http://w3.org/TR/REC-html40/appendix/notes.html, o. A.