JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Improving security when using the Internet Message Access Protocol (IMAP) – in a corporate environment

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

NETWORKS AND SECURITY

Eingereicht von:
Macskási Csaba, BSc

Angefertigt am:
Institut für Informationsverarbeitung und Mikroprozessortechnik

Beurteilung:
o. Univ. Prof. Dr. Jörg R. Mühlbacher

Assoz.Prof. Priv.-Doz. Mag. Dipl.-Ing. Dr. Michael Sonntag

Mitwirkung:
Dr. Dipl.-Ing. Oberrat Rudolf Hörmanseder

Linz, Juli 2011

Abstract (English)

IMAP is a widely used protocol for retrieving e-mail messages. It's popularity is caused by features like support for multiple connections, centralized storage and notification in case of new messages. Remarkably, current security solutions such as firewalls and antivirus software cannot control or limit IMAP communication based on the content of the transferred messages. E-mails cannot be reviewed appropriately regarding threat due to viruses, spam and extrusion prevention while being transferred over this protocol. This thesis analyzes the possibilities of the IMAP protocol and provides possible solutions for the mentioned issues. Finally an own proxy software solution for corporations is built, which has the capability to inspect live, TLS / SSL-encrypted IMAP traffic and alter it on the fly, if required. Documentation for this software is provided in form of a handbook in an own chapter. The consequence is better handling and safer operation of the IMAP protocol in corporate networks.

Abstrakt (Deutsch)

IMAP ist ein weit verbreitetes Protokoll zum Abrufen von elektronischer Post. Es ist populär, weil es Funktionen wie gleichzeitige Verbindungen, zentrales Speichern und das Melden neuer Nachrichten unterstützt. Erstaunlicherweise können aktuelle Sicherheitslösungen wie Firewalls oder Antivirensoftware die IMAP-Kommunikation nicht aufgrund dessen Inhaltes kontrollieren oder einschränken. Nachrichten können nicht angemessen auf Gefahren durch Viren, Spam und Datendiebstahl überprüft werden, wenn sie über dieses Protokoll gesandt werden. Diese Diplomarbeit analysiert die Möglichkeiten des IMAP-Protokolls und bietet mögliche Lösungen für die genannten Themen. Zum Schluss wird eine eigene Software für Unternehmen entwickelt, welche die Fähigkeit hat, in Echtzeit TLS / SSL-verschlüsselten IMAP-Datenverkehr zu inspizieren, und zu verändern, falls dies erwünscht ist. Dokumentation für diese Software gibt es in Form eines Handbuchs in einem eigenen Kapitel dieser Arbeit. Die Konsequenz ist ein besserer Umgang und die sicherere Verwendung des IMAP-Protokolls in Netzwerken von Unternehmen.

Content

Acknowledgment..... 7
1.Introduction..... 7
2. Theory..... 8
2.1.Why IMAP is an issue..... 8
2.2.Available products..... 8
2.3.Possible architectures..... 9
2.4.Timing and performance 12
2.5.Keep Alive Bytes..... 21
2.6.TLS / SSL..... 24
2.7.Determining protected IMAP accounts..... 26
Automatic protection with timeout..... 27
Administrator's O.K. for protection..... 29
2.8.Virus scanning..... 30
2.9.Detecting begin and end tags..... 33
2.10.Partial fetching..... 36
2.11.Inserting a virus warning message..... 40
2.12.Mail User Agents..... 43
3.Implementation..... 46
3.1.An early attempt..... 46
3.2.Final design..... 47
3.3.Flowchart: HandleClientSide..... 51
3.4.Access control..... 53
3.5.Handbook for administrators..... 55
3.5.1.Quick setup of TLS-proxy with IMAP plugin..... 55
3.5.2.Configuration file options..... 56
3.6.Debian package..... 59

3.7.Dependencies....61
3.8.INIT scripts....63
3.9.Testing architecture....64
3.10.MUA test results....69
4.Summary....78
4.1.Results....78
4.2.Problems and shortcomings....79
Bug No. 1....79
Bug No. 2....80
4.3.Ideas for further work....80
Literature....81
Abbreviations....84
Appendix....85
Appendix A: Sourcecode of an early attempt:....85
Appendix B: Prepare-network.sh....86
Appendix C: Init script....87

Illustration Index

Illustration 1: IMAP-filtering architecture with application level proxy....10
Illustration 2: Consistency between IMAP clients....10
Illustration 3: possible IMAP-filtering architecture....12
Illustration 4: Flowchart: Solution to the timing problem....15
Illustration 5: Flowchart: improved solution to the timing problem....16
Illustration 6: IMAP proxy delay....20
Illustration 7: way through the ISO-OSI layers....23
Illustration 8: STARTTLS....24

Illustration 9: End to end point security....26
Illustration 10: Determining protected IMAP accounts....27
Illustration 11: WEB-GUI....30
Illustration 12: Direction of protection....32
Illustration 13: Chunked fetching....39
Illustration 14: Mozilla Thinderbird v3.1.9: fetch by chunks....40
Illustration 15: MUA distribution....43
Illustration 16: Desktop MUA distribution....44
Illustration 17: IMAP-filtering architecture, early attempt....46
Illustration 18: TLS-Proxy entities....48
Illustration 19: TLS-Proxy entities exploded....50
Illustration 20: Flowchart: HandleClientSide....52
Illustration 21: environment during development....66
Illustration 22: environment during long term test: network architecture....68
Illustration 23: environment during long term test: flow of information....69
Illustration 24: Mutt....70
Illustration 25: Infected Message....71
Illustration 26: IMAP plugin, debug output....72
Illustration 27: Testing STARTTLS....73
Illustration 28: Thunderbird, attachments....74
Illustration 29: Thunderbird, success....75
Illustration 30: Windows Live Mail, success....76
Illustration 31: Wireshark, proof of TLS....77

Acknowledgment

I would like to thank all persons who contributed to the success of this Master thesis with their professional or personal support.
A special thank goes to o. Univ. Prof. Dr. Mühlbacher for the supervision of this thesis.
Dr. Dipl.-Ing. Oberrat Hörmanseder provided me with valuable technical information which I am especially grateful for.
I would also like to thank Mr. Wiesauer who made my internship at Underground_8 Secure Computing GmbH possible in the first place and Dipl.-Ing. Aspetsberger who provided me with technical information on a daily basis.
I further want to thank my parents Dipl.-Ing. Macskási Gábor and Mag. Macskásiné Kiss Márta and my grandfather vitéz Dr. Kiss Ernő, who supported me the whole time and made it possible for me to study in the first place.

1. Introduction

One of the most used protocols for e-mail retrieval by mail user agents is IMAP. However, tools to filter this kind of network traffic are very rare and not frequently used. This is a serious security hole as to achieve better security traffic should be either blocked or filtered. This is like the security check at the border of two countries: In order to prevent unwanted persons from entering a country, either all vehicles must be checked for these targets, or alternately no one may be allowed to enter the country. These two options can be considered safe. Right now IMAP is a black minivan which is almost always allowed to pass borders without being stopped. This thesis describes thoughts on how to make these minivans stop for a security screening. The acquired theoretical knowledge is used to develop a proxy application which can handle also the black van called IMAP.
In order to give this thesis some practical meaning and to avoid being stuck on a theoretical level the mentioned application was designed and developed during an internship at a company called Underground_8 Secure Computing GmbH [23]. This company produces firewall appliances which lacked an application level support of IMAP. The gist of the

internship is that they can include the developed proxy into their “MF”-series firewall appliances [19].

2. Theory

2.1. Why IMAP is an issue

Compared to other mail retrieval protocols such as POP3, IMAP is quite complex and difficult to handle. It nativly supports a great number of authentication methods. It can be either tunneled over SSL/TLS by an additional layer, or alternately the „STARTTLS“ command can be used to negotiate unencryptedly which encryption methods are supported and then to switch to an encrypted channel. According to RFC3501, IMAP contains 50 different commands and responses without experimental ones. Unlike in case of POP3, developing a semantic layer, which understands the logic of IMAP is necessary for an IMAP-proxy which alters connection data and communicates over the same connection to the real IMAP server as the client. IMAP is an interactive protocol. Filtering communication data in real time requires many resources and causes mail retrieval not to be as interactive as without filtering. While the proxy is scanning for viruses or spam, the client has to wait. The issues caused by delays are even mentioned in the corresponding RFC:

„The LIST command SHOULD return its data quickly, without undue delay. For example, it SHOULD NOT go to excess trouble to calculate the \Marked or \Unmarked status or perform other processing; if each name requires 1 second of processing, then a list of 1200 names would take 20 minutes!“ [1]

As a consequence designing an IMAP-proxy for our own needs from scratch would be a non trivial issue and is not the optimal solution to the given problem.

2.2. Available products

A more satisfying solution than developing an own proxy is to review the available open source software and to consider extending or altering an existing solution. Conveniently the used development system comes with a variety of IMAP software:

- “imapproxy”: This is a caching proxy for the IMAP protocol which needs an own instance for every backend (destination). It's main objective is session handling for

webmail clients which have a tendency to drop connections easily. From the legal point of view imapproxy could have been used easily as a base for this thesis as it is licensed under the GNU General Public License version 2 [24]. However, after careful reviewing it was clear that because of the difficult backend management, usage as a transparent filtering proxy cannot be considered.

- "imapfilter" [26]: This tool is an IMAP client, which alters mailboxes, copies and moves mail according to predefined rules. It is not a proxy server, but could be useful to remotely manage or filter mails. It is also free software and is included in the Debian [25] distribution.
- "perdition": This software is a mail retrieval proxy server which supports the protocols POP3 and IMAP. Support for SSL/TLS/STARTTLS is built in and it supports multiple backends. This seems to be the optimal candidate as a base for a spam filtering proxy solution. Due to its license which is GNU GPL version 2 [24] it is possible to modify the source code in according to our needs.

There are commercial products such as Junos Security from Juniper Networks [27], which offer similar functionality as needed in this thesis. The issue with these are their licenses. Source code is not available and their usage is not free, so they could not be used as basis for this thesis. Also the research within this thesis resulted in similar functionality and limitations as in case of Junos Security. Such limitations are described in section 2.10. of this document.

2.3. Possible architectures

In order to manipulate the data stream of an IMAP connection the proxy server needs to have application level understanding of the connection and needs support for the semantic level of the communication. This would imply that there is a single connection between the proxy and the backend. This connection is used to retrieve requested data for the client and to filter / scan messages on the server for viruses and spam. This architecture would look something like this:

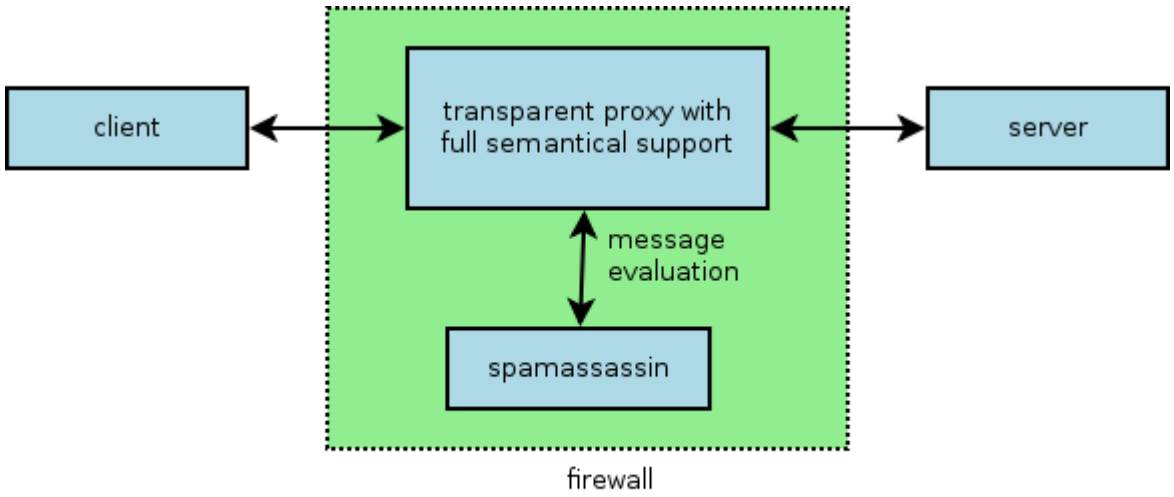


Illustration 1: IMAP-filtering architecture with application level proxy

The spam scanning software (in this case Spamassassin) would have access to mails via the proxy server. After evaluation of the messages it would be the proxy server's job to mark or delete them accordingly. That again implies, that the proxy needs to have a cache, which can be accessed by the client and the spam scanning software. This however leads to consistency problems:

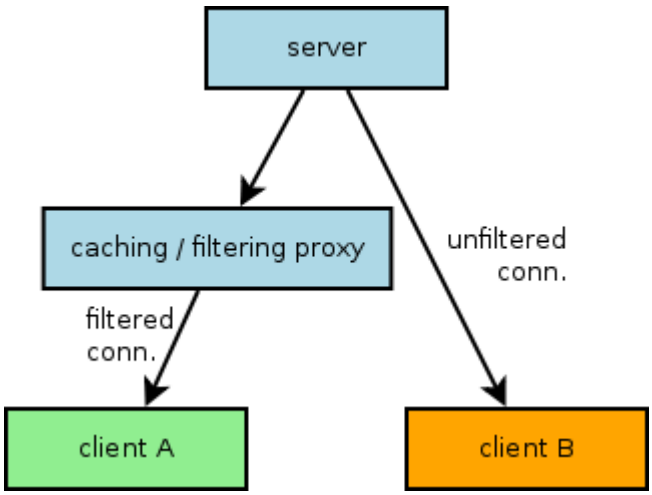


Illustration 2: Consistency between IMAP clients

This image shows how two clients communicate with the same server, accessing the same mailbox simultaneously. The connection of client A is filtered, while client B has a non

filtered connection. This means that client B has access to messages (which are probably spam or contain malware), which are invisible for client A. One could assume that there is nothing wrong with this, as client B sees the “regular” mailbox, while client A has an “improved” version. In fact there are cases where this is not acceptable. Imagine the following: Rob sits in his office and checks his incoming mail with his desktop machine, which connects to the IMAP server over the proxy with spam scanning abilities. Rob likes to be up to date and has an e-mail client running also on his smart phone at all times. The phone however connects to the external IMAP server via 3G and bypasses the spam scanning proxy successfully. The result is inconsistency, where Rob gets alerts on his cell phone from incoming mails, which are not present when viewed from his desktop machine.
The solution for this issue is a proxy, which does not filter the connection, but instead modifies the data directly on the IMAP server. As a result the same messages are visible for all clients, independent of the source of their connection.

As the best candidate for an existing proxy which could be modified to our needs is called “perdition” and does not fully understand IMAP. However, it is only used to handle encryption as a trusted third party and to dump login data which was sent by the client. From this point on the proxy has nothing to do except to shovel data through the TCP connection. The rest is done by another piece of software, which is called “scan connector”:

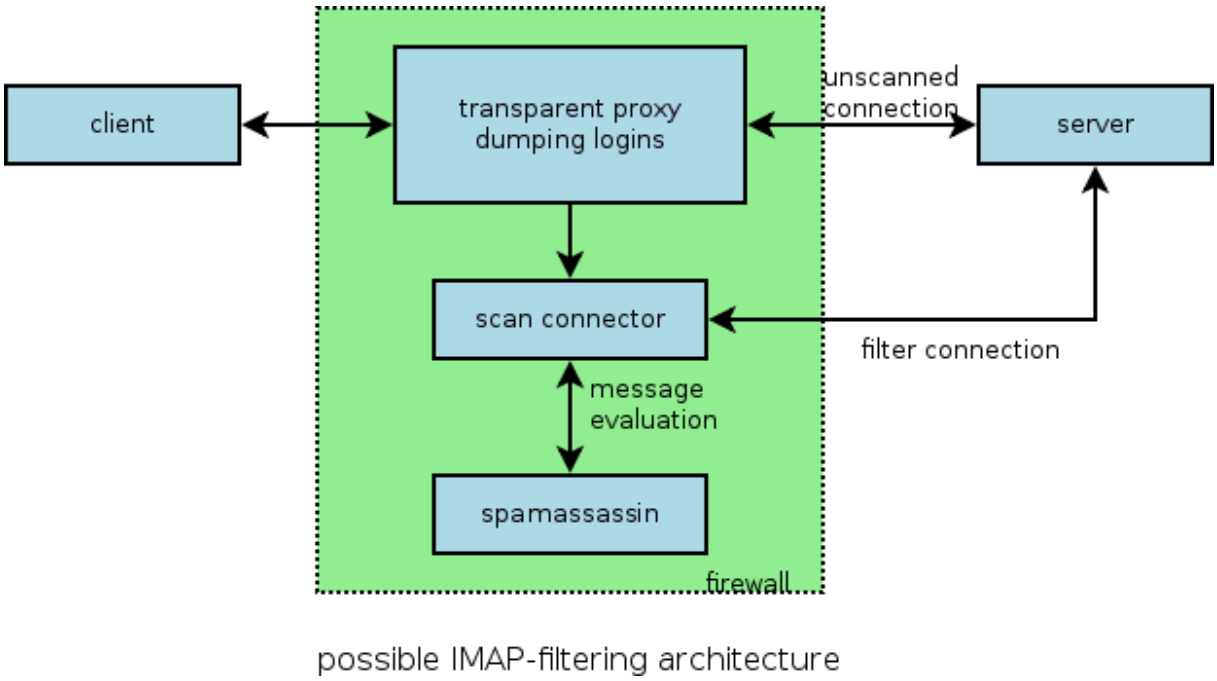


Illustration 3: possible IMAP-filtering architecture

In this design a second IMAP connection is established to the backend. This connection is used to retrieve messages, scan them by a third party spam scanning solution such as spamassassin and to flag or delete inappropriate messages. From the server's point of view, the spam scanning solution is just another client which reads and deletes messages. Unlike POP3, IMAP accepts several connections to the same mailbox and according to RFC3501 (IMAP4rev1) the server is responsible for keeping the clients in a consistent, synchronized state. The advantage of this solution is not only consistency, but also spam scanning for all clients, even those, which are not behind the proxy after the proxy was used the first time. In other words, Rob connects from his work machine to the IMAP server, the transparent proxy successfully gathers login and password and checks the mailbox periodically for spam regardless whether Rob's work machine is turned on or off. This way also Rob's smartphone's mail client is well protected against spam, even if Rob is outside his office.

2.4. Timing and performance

Beside quality, also quantity is an entity of mission critical importance. On one hand, quantity in this case refers to the amount of messages, which have to be scanned within a given period of time. If the speed is not assured which is needed to scan the given quantity of mail, IMAP

will be useless due to delays. On the other hand, quality stands for the accuracy of the virus scan's result. IMAP, being an interactive protocol has to be considerably responsive. This is true especially for the LIST command, which according to the corresponding RFC “should return it's data quickly”. This command is issued right when an IMAP connection is established and when the proxy should scan these messages, which have to be delivered instantly. Just to get a feeling how much time the scanning process consumes, some tests have been performed.

The test was performed on one desktop computer which has one 3Ghz Northwood core, 2gb RAM and Ubuntu as operating system. This box run spamassassin and isbg.py (the script, which fetches messages and passes them on to spamassassin) natively. A virtual computer with Debian and sufficient RAM was serving as IMAP-server. The network connection between these boxes was a 1000 Mbps connection bridged to the physical interface.

A simple script has sent 50 spam messages to the server with a rather simple bash command:

```
imap-server:~# for i in `seq 1 50`; do cat message_sample | mail u8; done
```

The file “message_sample” is an unwanted message with 3.1kb. Then the host computer was searching for spam, which was measured be the “time” command:

```
root@mcs-desktop:/home/mcs/imap_proxy# time ./isbg.py --imaphost
192.168.0.249 --imapuser u8 --imappassword u8pw
0 spams found in 50 messages
0/0 was automaticaly deleted

real  2m24.415s
user  1m51.660s
sys   0m4.080s
```

The careful reader will ask the question, why 0 spams were found. There are two explanations for this: Either the sensitivity of spamassassin was not set high enough, or I prepared the sample “spam” not carefully enough: The sender was root and the header (especially the “received”-lines) were valid. This process has been performed several times under slightly different conditions. The differences in run time are not worth mentioning.

Even an other connected IMAP client, which has to be informed all the time about changes in the mailbox via untagged server responses did not produce any measurable drop in processing speed. Don't forget that such updates occur not just if spam messages are moved or expunged, but also when the “\read” or “\recent” flag is removed.

For each message the test system needed 2.88 seconds for scanning. This result was calculated from the code sniplet above. The duration of the command is divided by the amount of scanned mail:

(2*60sec + 24.4 sec) / 50 = 2.88sec.

If the implementation of the proxy server and spam scan is blocking, than the response of the “LIST” command is delayed by 2.88 seconds per message. In case of one or two incoming mails at a time this seems to be acceptable, but just imagine the typical use case of a businessman opening his laptop in the morning or an e-mail address of any company's support department. They would have to wait several minutes until they can view their incoming mails. This does absolutely not comply with rfc3501.

The result of this experiment clarified that the original, blocking architecture of the imap-filter solution cannot be final. The following flowchart shows a solution to this problem:

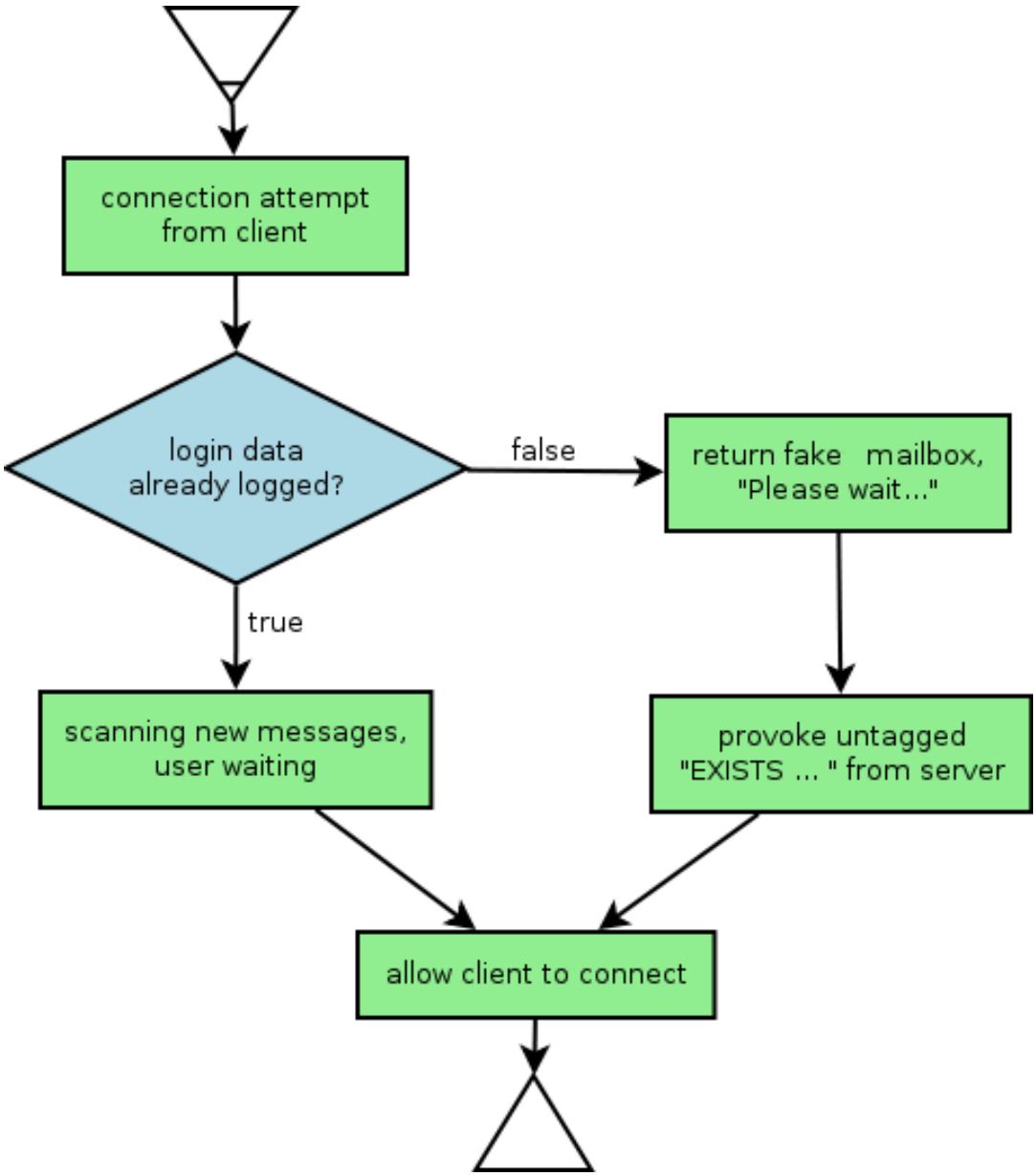


Illustration 4: Flowchart: Solution to the timing problem

Upon the connection attempt from the client, instead of blocking the IMAP connection the proxy shows a fake mailbox with a kind note that the user's mails are being examined. The messages are shown for the user only after the examination process. Although this seems to be a quite elegant solution, consistency is not given in case of multiple devices where at least one device is not behind the proxy. Also for the realization of this flowchart application level understanding of IMAP has to be implemented in the proxy.

It could be a better compromise to show all incoming mails instantly and to run periodic checks where spam is filtered or deleted. This would not delay mail delivery and could

provide reasonable protection against spam with acceptable speed after the first time a mailbox is retrieved. The flow chart of this improved model could look something like this:

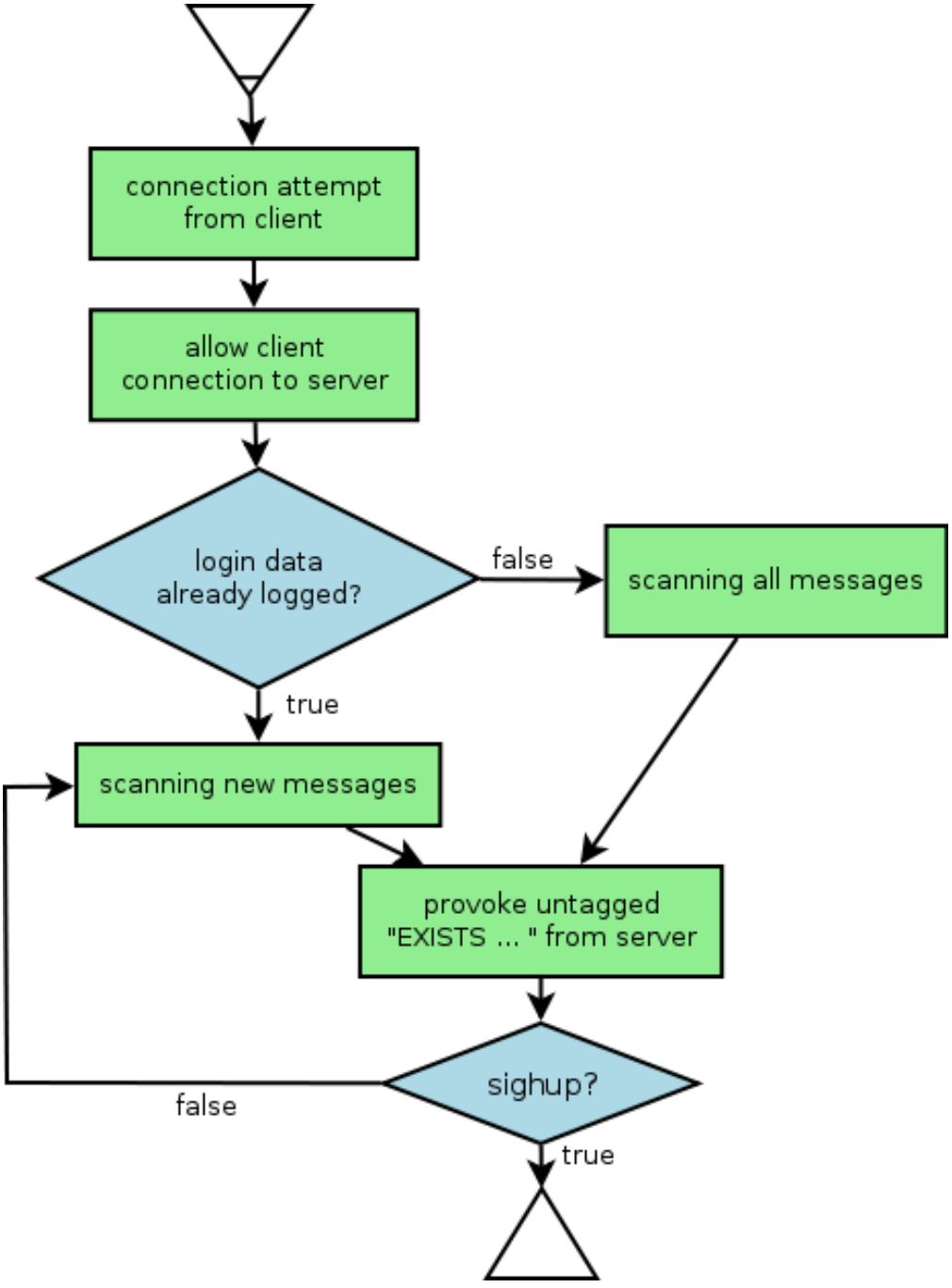


Illustration 5: Flowchart: improved solution to the timing problem

* 'Untagged' refers to the IMAP response from the server without a command number tag

“Sighup” [28] is a standard signal found in POSIX [29] systems which indicates the death of the controlling process / terminal. In this case sighup means immediate termination of the program.

As one can see this flowchart shows a non blocking spam filtering architecture because the first action of the proxy is to establish an unfiltered connection to the real IMAP server for client communication. In case of a newly learned account all mails (also older ones) will be checked for spam. Later just those with the “\unread” flag have to be checked. This means that read messages are not checked twice. This is of crucial importance regarding performance. Just consider how long it would take to recheck a typical mailbox containing several hundreds of mails every time a connection is established.

As for general performance, this spam-scanning proxy does not need more resources than other solutions which are directly connected to the MTA or filter SMTP traffic, because it uses the same backend as other software in it's environment: Spamassassin. That means that the performance of any typical appliance (specified in filtered messages / time) is not affected by the presence of IMAP filtering. The minimal overhead of the proxy server itself is not worth mentioning as it does not have application level understanding of the protocol and basically all it's job is to shovel data from one connection into another one without altering or analyzing it in any way.

The idling time between two spam scans should be the administrator's choice. If this time is chosen too large, spam is not filtered effectively: In case of an interval of one hour the user would receive an hour's spam until it is properly dealt with. On the other hand if this value is too small like a few seconds, it will open connections more frequently thereby unnecessarily using resources. It could even occur that two scanning instances overlap and scan the same account simultaneously. This does not have to be prevented by the software because this case should never occur while using a reasonable configuration. The recommended value for this setting depends on the amount of new messages and the available computing power and varies between 5 and 60 minutes.

Another important aspect of timing is the scanning for viruses. As mentioned in section 2.8. of this document, in contrary to spam scanning, virus scanning has do be performed on the fly as it is not an option to let viruses into the protected network even temporarily. The duration of the scanning procedure depends of course on the computing power of the computer as well

as on the virus scanning software and the size of the message being scanned. In case of a dual level architecture, which means that more than one virus scanner is used, the needed resources roughly double. In case of uniprocessor systems this means that also the duration of scanning doubles. Depending on the actual performance of the proxy system it is not obvious that the added security of having two virus scanners is more important than being able to download the message faster.

Within this thesis the Clam AV software is used for timing tests. This software will also be used on the appliance which the IMAP proxy will run on. In order to get a feeling for the performance of the virus scanner a flash video file with a size of 1.5 mb was scanned. In case of e-mail a message of this size could contain for instance two high resolution pictures. This amount of data can be considered a quite common scenario. Here are the results using the „clamscan“ command:

```
bitumen@turul:/home/bitumen/filmek/youtube$ ls -lh italian_in_malta.flv
-rw-r--r-- 1 bitumen bitumen 1.6M Jun 29  2008 italian_in_malta.flv
bitumen@turul:/home/bitumen/filmek/youtube$ clamscan italian_in_malta.flv
italian_in_malta.flv: OK

----------- SCAN SUMMARY -----------
Known viruses: 852564
Engine version: 0.96.4
Scanned directories: 0
Scanned files: 1
Infected files: 0
Data scanned: 1.51 MB
Data read: 1.50 MB (ratio 1.00:1)
Time: 6.377 sec (0 m 6 s)
bitumen@turul:/home/bitumen/filmek/youtube$
```

For this file with a size of 1.5 mb clamscan needed 6.38 seconds. This is a quite unfavorable result. Imagine that when fetching a message additionally to a probably slow network connection there is a delay of over 6 seconds. That would render the planned software unusable. One could think that 1.5 mb are quite big and that the delay would be smaller in case of messages with plain text content. However scanning a file with a file size of 4 Bytes needed also 6.07 seconds. The difference between the durations of the performed scanning operations are too small to be greatly influenced by file size. Obviously if a file is bigger, scanning it takes more time. However the scanning overhead in case of the 1.5mb file compared to the 4 Byte file was 6.38-6.07=0.31 seconds. That means that scanning 1mb of data additionally takes approximately 0.31/1.5=0.2066 seconds. 200ms/mb is an acceptable number. However, the scanning duration of approximately additionally 6 seconds regardless

of file size has to be improved and cannot be accepted in a production environment. As it turns out, Clam AV loads it's virus signatures into the memory in these 6 seconds. Conveniently that can be done in advance by using the command “clamdscan” instead of “clamscan”. The difference between these two commands is that “clamscan” is a stand alone binary and “clamdscan” uses a scanning daemon which runns all the time in the brackground and holds the virus signatures in memory. A requirement for this is that this daemon called “clamd” is running. Scanning the 1.5mb file from the previous example with the “clamdscan” command looks like this:

```
bitumen@turul:/home/bitumen/filmek/youtube$ ps ax | grep clam
 1993 ?        Ssl    0:05 /usr/sbin/clamd
 4603 pts/3    S+     0:00 grep clam
bitumen@turul:/home/bitumen/filmek/youtube$ clamdscan italian_in_malta.flv
/home/bitumen/filmek/youtube/italian_in_malta.flv: OK

----------- SCAN SUMMARY -----------
Infected files: 0
Time: 0.182 sec (0 m 0 s)
bitumen@turul:/home/bitumen/filmek/youtube$
```

The result shows that scanning with the scanning daemon is even faster than expected: The duration for the 1.5mb test file was 182ms which gives us 121.3ms per mb which equals 8mb per second. Keep in mind that the accuracy of measuring this short durations is greatly influenced by the resolution of the system time. Measuring larger values clarified that accuracy is sufficient. In case of a linear relation for a 10mb file the scanning daemon would need about 1.2 seconds. Tests show that the relation is in fact linear:

```
bitumen@turul:/tmp$ dd if=/dev/urandom of=rand bs=1M count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 3.0535 s, 3.4 MB/s
bitumen@turul:/tmp$ clamdscan rand
/tmp/rand: OK

----------- SCAN SUMMARY -----------
Infected files: 0
Time: 1.093 sec (0 m 1 s)
bitumen@turul:/tmp$
```

Scanning took 1.09 seconds instead of the estimated 1.2 seconds. The conclusion is that if “clamdscan” is used to scan messages, scanning will take approximately 120ms/mb on a machine with a Pentium M 745 processor which provides sufficient performance for messages of a few mega bytes.

Additionally to the duration of the virus scan there is additional delay due to caching behavior within the proxy. Without proxy the message is delivered by shoveling the requested parts of

the RFC 2822 encoded message (Internet Message Format) over a TCP connection. In order to actually find viruses it is necessary for the virus scanner to scan the message in once piece. It is not possible to apply a virus scanner on a TCP stream. Even if that was possible it would not make sense to do so as if a virus is found in the middle of the stream it is not possible to "unsend" the first part of the stream which the client already received. Instead the message has to be cached, scanned and then forwarded in case of it being free of viruses. The consequence of caching is delay.

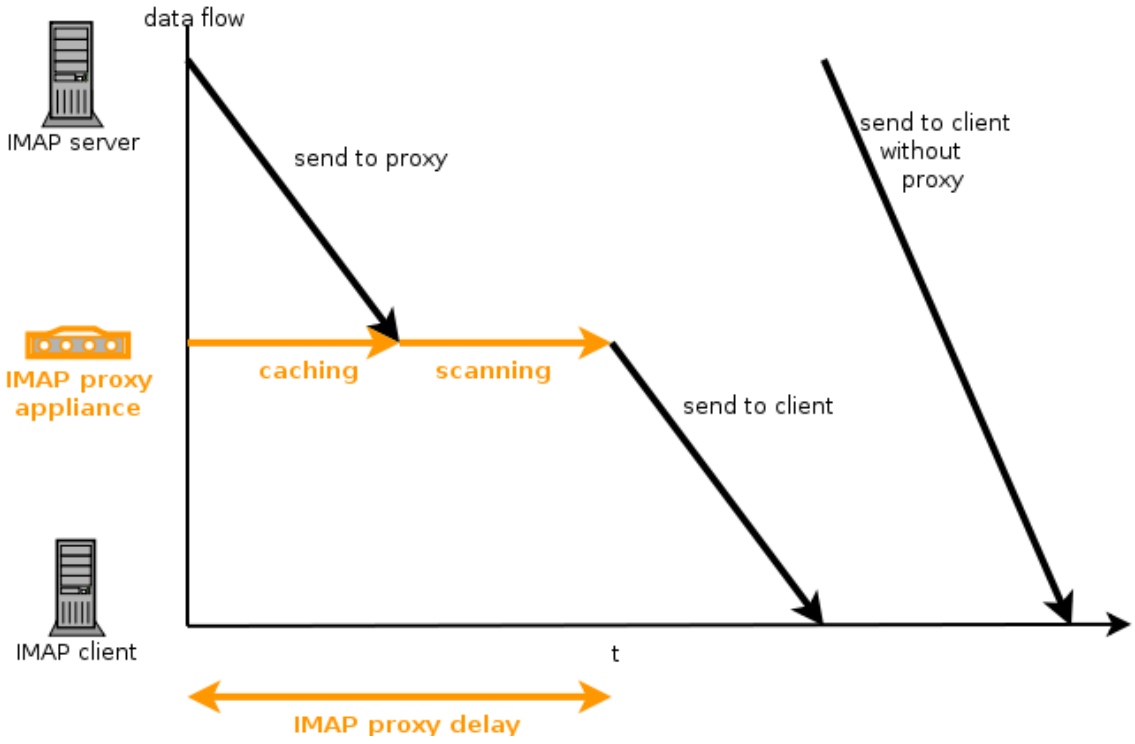


Illustration 6: IMAP proxy delay

Assuming that the network connection between server and proxy and between proxy and client both have the same bandwidth, the following formula specifies the total delay caused by the virus scanning proxy in case of message retrieval:

$$T_{delay} = t_{transfer} + t_{scanning}$$

To get a better feeling for the delay, here is an example from an everyday scenario: The bandwidth between server and proxy is 5MBitps (broadband connection), and between proxy and client 100Mbitps (full speed Ethernet). The e-mail which is retrieved via IMAP consists of 2MB (RFC 2822 size, header included). The proxy needs 120ms/MB to scan the message. Without proxy downloading the message takes:

$$2MB \;/\; 5Mbitps = 2*8Mbit \;/\; 5Mbitps = 3.2sec$$

This value is increased by the imap proxy to:

```
(2*8Mbit / 5Mbit/s)*1000 + 2*120ms + (2*8Mbit / 100Mbitps)
*1000 = 3600ms = 3.6sec
```

Keep in mind, that the integer 1000 as multiplier refers to time (ms) and not to data volume.

These calculations do not include internal computations performed by the proxy such as the overhead of memory allocation or parsing the traffic for keywords. The partial fetching feature of the IMAP protocol can cause much more significant delays. This is usually the case just with certain IMAP clients and is described in chapter 2.10.

Another aspect of timing which has to be considered are timeouts. The only timeout which is specified by RFC 3501 for IMAP is the auto logout timeout on the server side. This should not affect an IMAP proxy at all as it occurs just if the IMAP server does not receive any command from the client for at least 30 minutes. The only case where the proxy could alter the behavior of the auto logout mechanism is if the client sends an idle command every 29 minutes and a few seconds to keep the connection alive and the delay of the proxy causes this command to arrive more than 30 minutes after the last one at the server. In case of the server having a 30 minute timeout it drops the connection. As a result the IMAP client has to log in again.

2.5. Keep Alive Bytes

A more important part of timeouts which is not dealt with by the RFC is how long it may take to send the response to commands like “fetch”. This is especially important for an IMAP proxy as messages which are downloaded have to be cached for scanning before they can be sent to the client. During this time the client may drop the connection and reissue the fetch command because it is not aware of the proxy and it's effect on the IMAP server's behavior. This issue has been experienced in case of large attachments and the MUA Mozilla Thunderbird / Icedove with the partial fetch feature turned off.

One possible solution is that the proxy sends small parts of the message which is being cached to the client before it is evaluated. If too much is sent in advance, the proxy looses it's functionality. However, even a rather short e-mail header with one “received” entry has about 200 bytes. As the smallest amount of data which can be sent to the MUA is one Byte, even in the worst case 200 messages can be sent without the risk of allowing the message body or

even attachments to pass to the client. If we consider that in the default configuration of the mail transfer agents Exim and Postfix in the Debian distribution the maximal message size of e-mails is 50MBytes, even in case of the biggest attachments one Byte of the message can be safely sent to the IMAP client every 50MB*(1024^2)/200 = 262144B = 256KB. The following calculation determines the duration between Bytes which are sent in advance in this worst case scenario if the 3G connection with a bandwidth of 5MBitps from the example above is used:

```
5Mbitps * 1024 / 256*8kBit = 5120kbitps / 256*8 kBit =
2.5/sec.
```

2.5 such keep alive Bytes can be sent per second which allows even a ridiculous timeout of less then half a second on client side. Keep in mind that this value is true for the most probable maximum message size with a small message header. Let us look at a more probable case in which a 2MB message with a 400Byte header is retrieved with a bandwidth of 0.2Mbitps:

```
0.2Mbitps * 1024 / (2*8Mbit*1024/400B) = 5/sec
```

It is known that by delivering bytes of the header one by one while caching or evaluating the message timeouts on client side can be avoided. There is no need for such a timeout to be known in case the header is sent to the client in advance as the system works even in case of very short timeouts. However it would be an advantage if there was a standardized value as that would make it possible to prevent unnecessary keep alive Bytes to be sent to the client. The absence of timing requirements on server side except for the vague description of the “LIST” command is definitely a shortcoming of RFC 3501.

As good as the working principle of keep alive bytes is, it proves to be just as effective as the underlying system makes it possible. In order to see what is under the IMAP plugin of TLS-Proxy, first it has to be placed somewhere in the ISO-OSI model [31]. It's place within the ISO-OSI model is the application layer, which is layer number 7. It cannot be placed on lower layers because it has the ability to alter the communication on layer 7. The fact that the IMAP-plugin cannot talk IMAP itself implies that the proxy lies just “almost” on layer 7. Within a computer's networking stack the information has to be reached from layer 1 up to layer 7 where it is processed by the IMAP proxy before it is sent back down to layer 1 where it is sent in the client's direction:

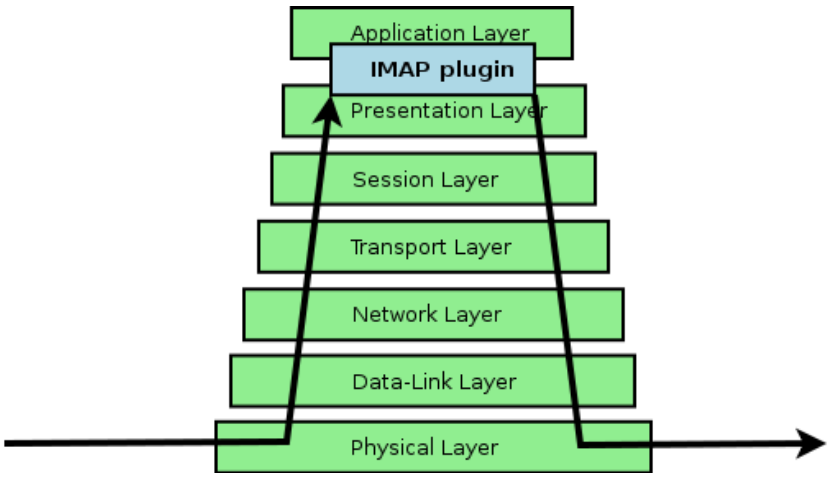


Illustration 7: way through the ISO-OSI layers

On the way of the information through the layers delay is caused by several factors. The most significant aspect are buffers. Especially layer 4, the transport layer which is responsible for flow control can cause delay while waiting to fill buffers before sending data to improve overall throughput performance. IMAP uses TCP on layer 4 so some sort of buffering is always present. IMAP over UDP is not defined. In case of keep alive bytes the time of arrival is of crucial importance because it's main purpose is to prevent timeouts. If it arrives after the timeout has already occurred communication is interrupted and the keep alive byte is useless.

Every layer in the communication causes additional overhead due to protocol headers. So does TCP. The goal of it is to collect the ideal amount of data in a buffer and then send it. If an application tries to send just one Byte, TCP is likely not to send it right away because the protocol headers' sizes are multiple times as big as the one byte which the proxy wants to send. The drop of performance due to small buffer- / window-sizes is described further in section 2.10.

However, the idea behind the keep alive byte is not high throughput performance but more the urgent delivery of the smallest amount of data which can be transmitted at one. For this purpose the Transmission Control Protocol provides an own flag within it's protocol header which causes some TCP stacks to handle it with higher priority. This flag is called the PSH or push flag:

“The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.”, [32]

So in theory the PSH flag solves the timing issue of keep alive bytes. Practically it can work similarly as in case of Telnet and SSH: Keystrokes are sent immediately despite of their small data size. However, it is not guarantied that packets are delivered instantly:

"A TCP MAY implement PUSH flags on SEND calls. If PUSH flags are not implemented, then the sending TCP: (1) must not buffer data indefinitely, and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent).", [30]

This means that the implementation of the PSH bit in TCP stacks is optional. If it exists on the host where the proxy is being run, the PSH flag can be set to increase the probability of timeout prevention.

2.6. TLS / SSL

Using IMAP as a clear text protocol causes similar problems as other legacy protocols such as FTP or Telnet. Due to privacy issues an encrypted version of IMAP was needed. RFC2595 [12] specifies how to use IMAP on top of a TLS layer. There are basically two solutions:

- The first one is to wrap SSL around IMAP and offer this service on the dedicated port 993. Doing this would result in port 143 being obsolete as clear text connections are not wanted.
- The second possibility is introduced by RFC2595 and provides a more elegant solution to the problem. Only the original port 143 is used and an own command named "STARTTLS" was introduced.

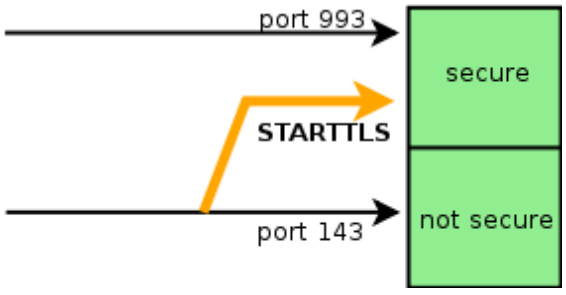


Illustration 8: STARTTLS

At the very beginning of an IMAP conversation the client issues the "CAPABILITY" command. The server answers by sending a list of supported features, usually including STARTTLS. In this way the client knows that encrypting the connection is possible and may

issue the STARTTLS command. After the handshake, which includes negotiating keys and checking either one or both certificates, the TLS layer is set up and all further communication is done over this additional TLS layer. It is crucial to note how important the handling and checking of certificates are in order to prevent person-in-the-middle attacks, which is basically the illegal way of what the TLS-Proxy does under the name "trusted third party". The aspect of certificate checking is also mentioned in RFC2595:

"During the TLS negotiation, the client MUST check its understanding of the server hostname against the server's identity as presented in the server Certificate message, in order to prevent man-in-the-middle attacks." [12]

Another interesting capability of IMAP is called "LOGINDISABLED". If the corresponding feature is enabled, this capability is advertised just until "STARTTLS" is issued. An example for this is given in RFC2595 [12]:

```
"C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS LOGINDISABLED
S: a001 OK CAPABILITY completed
C: a002 STARTTLS
S: a002 OK Begin TLS negotiation now
<TLS negotiation, further commands are under TLS layer>
C: a003 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=EXTERNAL
S: a003 OK CAPABILITY completed
C: a004 LOGIN joe password
S: a004 OK LOGIN completed"
```

After the encrypted channel has been set up it is not advertised any more. This makes sense as "LOGINDISABLED" forbids authentication with login credentials. The result is that sending login name and password over the connection is not accepted unless the channel of communication is encrypted. This measure makes it more difficult to sniff login credentials which enhances security. However, the online scanner which is part of this thesis and scans IMAP accounts for spam requires login credentials to be sniffed by the proxy which is used. This means that the proxy server has to filter the IMAP-server's capabilities so that login credentials would be used. This does not affect security as TLS is still used externally.

Another aspect of security which is worth emphasizing is that TLS does not guaranty that only the recipient of the message is able to read it. TLS makes it nearly impossible to eavesdrop on the network traffic in clear text. However, MTAs may store or transfer these messages without encryption before they are fetched over IMAP. Looking at the delivery chain of a message there are several nodes where security can be compromised:

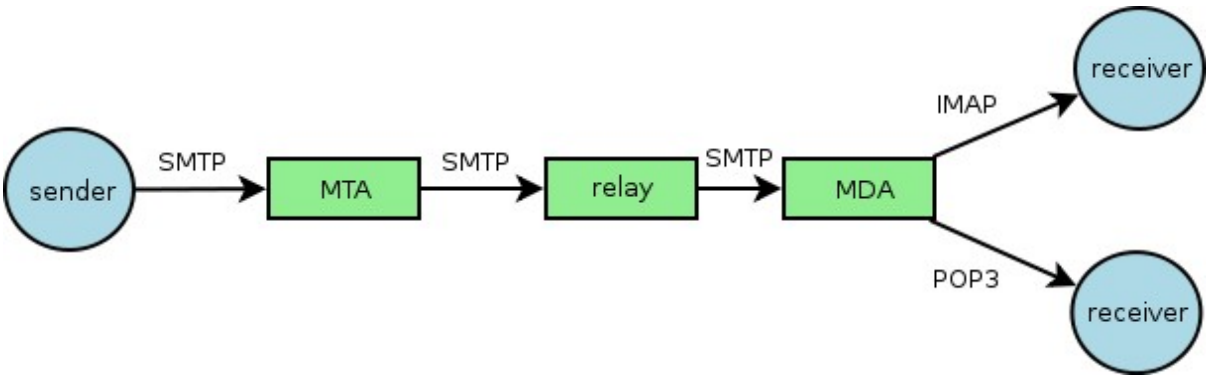


Illustration 9: End to end point security

As the graphic shows beside IMAP there is also the SMTP protocol which can be eavesdropped on. SMTP is used for communication between the Mail Transfer Agent, possible mail relays and finally the Mail Delivery Agent. One possible solution to this security issue is to use end to end point security products such as PGP [33]. This program encrypts the content of messages which means that scanning for spam or viruses is useless. The application, which was developed in this thesis is not compatible with end to end point security products.

2.7. Determining protected IMAP accounts

A general issue while designing or configuring security solutions is the philosophy of protection. There are two extremes: Either the firewall allows all traffic and has no filtering effect at all, or it blocks all traffic, making communication impossible and rendering the product totally useless as well. The goal is to have a design which does not need to be configured excessively and preferably offers protection just by activating a checkbox on some web GUI. In case of IMAP filtering inside of a firewall two different ways can be considered possible, which are shown in the following flowcharts and are described below:

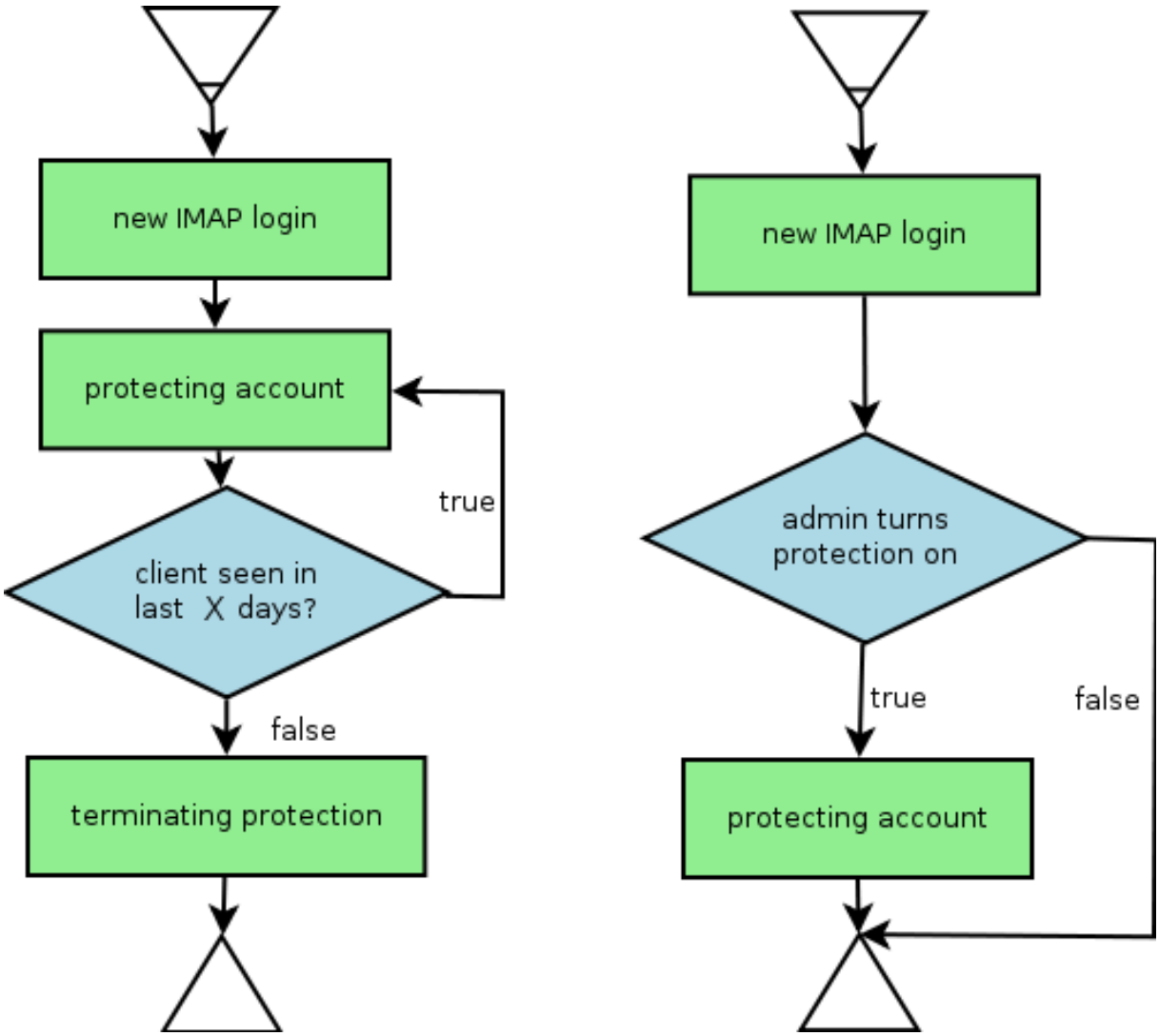


Illustration 10: Determining protected IMAP accounts

Automatic protection with timeout

Using this method the proxy scans authentications on the fly in case of plain text authentication or even as a trusted third party with an own, valid certificate in case of encrypted connections and learns login credentials of IMAP mailboxes on the fly. Triggering protection is done just by the presence of an IMAP connection if filtering is activated on the web GUI of the software. This solution requires just one check box to be set, but is affected by the guest problem.

Think of a company network where mail servers and their administration are outsourced. It is a desired behavior that all employees messages are checked for spam and viruses. Private

accounts can be included as well due to possible company policies. However the question, whether guest's mailboxes should be checked or not does not have a clear answer:

- On one hand filtering spam can be considered harmless and useful also for guests.
- On the other hand it is most probably not the desired behavior that guest's mailboxes are filtered even after they left the office. This could affect also other companies' mail accounts! There is no trivial way of determining whether the guest has already left the building / the company's site or not. To use GPS [34] tracking or MAC-address registering and tracking would have to be used to determine the physical location of a device. This however could be overridden by malicious users by faking mac addresses and would hereby open a whole new spectrum of issues. Also such a tracking system is way beyond the borders of this thesis.

This also concludes that if automatic protection without explicit confirmation of an administrator is only acceptable if there is a reasonable timeout in case of user inactivity. This way also guests' accounts are filtered. After they leave the protected network with their devices their accounts are still checked until a timeout triggers the removal of the given account from the list of protected accounts. This timeout has to be chosen very carefully:

If chosen too small, even own employees' accounts can be left unscanned if their desktop computer is turned off for too long or their notebooks are out of the office network. The consequence of the accidental removing of accounts is that until the next connection to the mailbox, messages are left unfiltered. This way successful spam protection cannot be provided.

If the timeout is chosen too big, the above mentioned guest problem occurs. Depending on the amount of guests and mail traffic also performance problems can occur as the system tries to scan an unnecessarily large pool of mail accounts. Note that all messages with the “\unread” flag are checked. Accounts which are not checked frequently can contain a large amount of messages which probably should not even be filtered. An example for this case is an account with automatically generated mail like logs. It is quite likely that a frequently running cron job or a daemon will have some problem and will start to send error messages for root. For such messages spam filtering just does not make sense.

Also automatic protection of mailboxes can be a potential attack vector for denial of service attacks. If a malicious user named “Rob” fills some mailboxes with spam collected from the

internet or any other generated messages and then connects to these mailboxes simultaneously from within the protected network the firewall's resources will be used to filter these messages which are never going to be read or used in any way. Think of following scenario: Our test hardware which has been used to determine the software's performance (see chapter 2.4.) needs in average 2.88 seconds to check one message. So all Rob has to do is filling 5 mailboxes with 150 messages. This is still a rather small value! Each mailbox requires 150*2.88 sec = 432 sec of processor time. Considering 5 mailboxes and assuming that spam filtering is done by one core 432sec*5= 2160 sec which is equal to 36 minutes. By a relatively small amount of mails and with little effort Rob could cause excessively high load for more than half an hour. This issue could be worked around by giving spamassassin a relatively low priority and thereby preserving the responsiveness of other application running on the same hardware. However this behavior is likely to be considered a design flaw. A down to earth solution is not to check any mailboxes automatically thereby eliminating the guest problem as well as the possible attack vector for DoS attacks.

Taking the edge of the guest problem does not mean that there will not be any performance issues at all. There has to be sufficient computing power available to scan non-guest mailboxes.

Administrator's O.K. for protection

Instead of automatically scanning newly learned accounts another possibility would be to store the names of these accounts in a list, present them to the administrator and let him decide whether that account should be checked or not. This could be done over a web interface with one main switch which turns on IMAP filtering and one check box for each learned account. In this case the system would not interfere with guest's computers and their mail accounts. It would be also possible to choose just those mailboxes which really need scanning and thereby save resources.

Beside of all these advantages the only known problem with this design is that human intervention is needed for the system to work. Especially in case of a bigger network with several hundred IMAP connections this would result in a performance issue on the human side. Options on the web interface which allow to select all mailboxes and uncheck those which are not needed could provide a satisfactory solution.

Illustration 11: WEB-GUI

This is an example of how the web interface could look like after integration into the Underground_8 MF70 firewall. Here the administrator has to enable scanning of accounts one by one. Note, that the lines in the image above represent the IMAP login name and the IMAP server in the format "login@server". They are not necessarily E-mail addresses. This format is common. It is for example used by the command line client of the secure shell.

2.8. Virus scanning

After already having a first impression of how to implement the spam scanning software during an internal meeting the question was raised, what is more important: Spam scanning or virus scanning? While spam is annoying and more a convenience factor, computer viruses can compromise workstations or even a whole network. At this point it became clear that spam scanning has to be the secondary goal of this thesis, as the main purpose is to enhance security which is done by keeping viruses and other malware outside of the protected network. In

order to secure the internal network it is essential to scan all traffic from the internet. Because of this there are two choices: either blocking the IMAP ports (143 and 993) or to filter the traffic over these ports which requires a transparent proxy with virus scanning capability.
Also the design of an asynchronous online scanner as described in section 2.3. had to be rethought as virus protection has to be done on the fly. It is a no go criteria if a potentially dangerous message can be downloaded over IMAP before being scanned by the asynchronous scanner at a later point in time. It became clear that the practical part of this thesis will be split into a transparent proxy with virus scanning capability and an online scanner which deletes or moves spam on the IMAP server. Following criteria were agreed upon:
must have:

- transparent proxy
- logging login information on the fly for the asynchronous scanner
- no caching (storage of messages on the firewall)
- scanning of the message body for viruses on the fly
- account / domain exclusion (“blacklisting”)
- scanning of TLS connections done by the existing TLS-Proxy

optional features:

- asynchronous client / online scanner
- anti virus and anti spam solution for remote IMAP clients which do not communicate over the IMAP proxy
- account management interface (WEB-UI)

After evaluation of available proxies it became clear that in order to support the scanning of TLS encrypted connections an existing, general TLS proxy with this feature has to be extended by an IMAP module which supports the scanning and finding of viruses. This leads to the question, where viruses or other malicious code can be located within the IMAP traffic. Two attack vectors were found:

- The first is the sending of manipulated commands which do not comply with RFC 3501. Implementations of IMAP servers, which contain bugs may be vulnerable to certain buffer overflow or denial or service attacks. If for example an IMAP server does not perform sufficient bounds checking on message UIDs or the length of login credentials, the whole service could be taken down by a manipulated fetch or login

command. This thesis is not going to deal with the enforcement of syntactic and semantic correctness of IMAP traffic and therefor cannot protect against such attacks.

- In case of the second attack vector the target of the attack is the MUA. The proxy is just used to forward the malicious code to the IMAP client. Such code can be located within the header of the message which includes the subject, or in the body which can contain text or attachments. The subject is important on its own as the IMAP protocol makes it possible to download just the subject or just the header of a message so it can be considered an own attack vector. The subjects of the messages have to be returned by the IMAP server upon request of the client immediately, so scanning for viruses is not recommended here. However, the message body with attachments is the most probable place for malware to be located at. This means that the developed software must be able to locate message bodies within the stream, cache them until the server has sent the whole message, evaluate it and depending on the result of the virus scan either send it to the client, or cut out the content, leave the header, insert a virus warning and then send it to the client.

At first glace it seems to be straight forward that only the data stream has to be analyzed where the server is the source and the client is the destination. However IMAP also has a feature where messages can be uploaded to the mailbox. This happens for example if messages are moved to the “sent” folder or the draft “folder”.

If traffic, where the server is the destination and the client is the source is not scanned, messages containing malicious code can be uploaded to the mailbox located on the IMAP server. This could be potentially dangerous for other IMAP clients which access the same mailbox but are not located behind the proxy server.

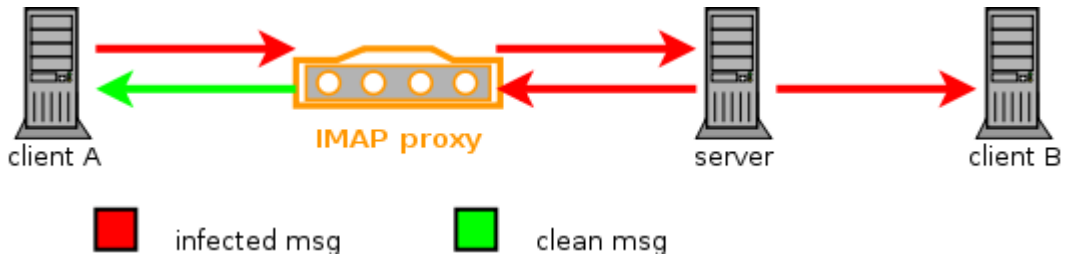


Illustration 12: Direction of protection

On this picture it is visible that the network behind the IMAP proxy is secure even if one of the machines within the network uploads messages with malicious content to the IMAP

server. However, these messages can infect those clients, which connect to the same server and are not using the proxy (e.g. client B).

Another important aspect of message scanning is the input format of the e-mail being scanned. The knowledge which is available about these messages are that they comply with the Internet Message Format as described in RFC 2822 [18]. In order for the virus scanner(s) to be successfully scanned multipart messages have to be split and attachments have to be extracted into separate files to be scanned independently. The following command can be used for MIME decoding [17]:

```
~$ ripmime -i <msg> -d <output_dir>
```

The separated parts of the multipart message can be found in the directory “output_dir”. If the decoding of MIME is not done, many virus scanners will not produce any usable result because they cannot decode MIME themselves and as a result their virus signatures will not match. Viruses will not be found.

Testing of the IMAP plugin's virus filtering capabilities had to be tested. As sending real viruses in e-mails over several networks is not something what a responsible software developer would do, the EICAR signature [35] was used. This rather short string's signature is implemented in all up to date virus scanners and was designed especially for testing purposes. The signature itself looks like this:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

After MIME decoding of messages the IMAP plugin handles messages with this content as infected.

2.9. Detecting begin and end tags

In order scan messages for viruses on the fly the most difficult and important aspect is to detect the beginnings and ends of messages. If either of these does not work, the proxy looses functionality: Either it lets messages pass without being scanned because they are were not found, or it just caches without writing to the client as all IMAP traffic is handled as part of the message.

As writing a parser for IMAP traffic which can detect messages semantically from the context was out of the scope of this thesis, another method of handling at least the fetch command had to be found in order to detect the transfer of e-mails within the IMAP stream. The first idea was to look for the „FETCH“ command originating from the client to detect when a message

is requested. This command can be executed according to RFC 3501 with the following arguments:

```
"ALL, FAST, FULL, BODY,  BODY[<section>]<<partial>>, BODY.PEEK[<section>]
<<partial>>, BODYSTRUCTURE, ENVELOPE, FLAGS,  INTERNALDATE, RFC822,
RFC822.HEADER, RFC822.SIZE"
```

Due to the large number of options and possible responses handling the "FETCH" command issued by the client seemed to be an overkill for the needed functionality. This is the reason why this command is not described further. Also not all responses to these commands have to be scanned. If just the size of the message according to RFC 822 or the header is requested, scanning is not necessary as viruses are not located there. If these commands were used to detect messages, also the tagged server response would have to be analyzed as it is possible that instead of the message body the server simply sends "NO - fetch error: can't fetch that data" or "BAD - command unknown or arguments invalid". Both responses comply with RFC 3501 and could cause unwanted behavior of the proxy. Also the structure of a message transferred over IMAP can not be known in advance for sure as it is not specified completely by RFC 3501: *„Any following extension data are not yet defined in this version of the protocol. Such extension data can consist of zero or more NILs, strings, numbers, or potentially nested parenthesized lists of such data.“*, [1]

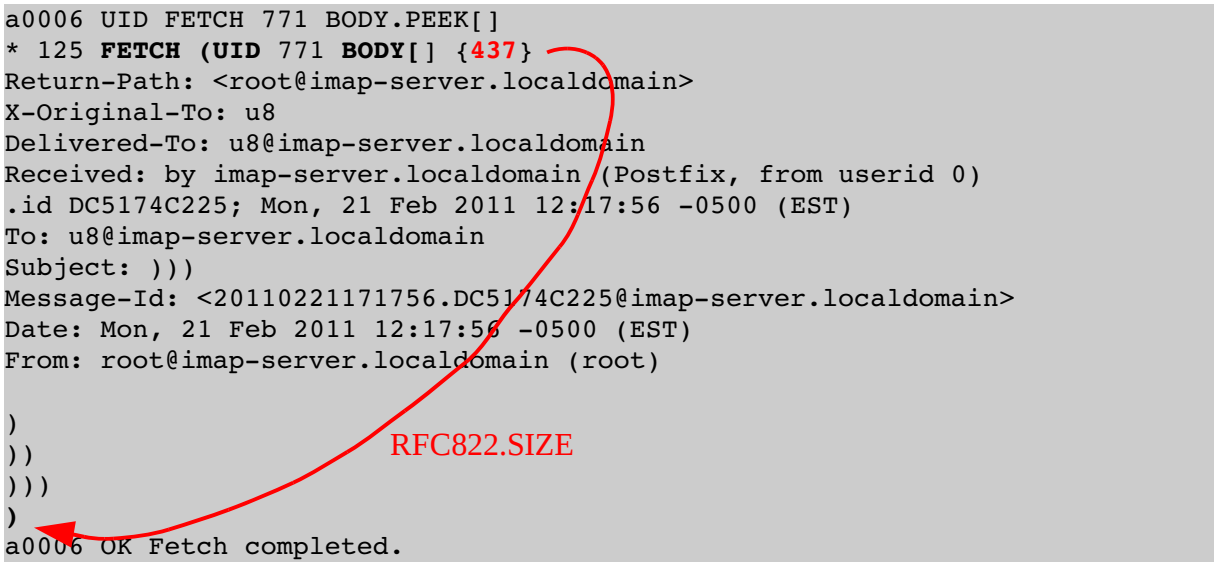
A much simpler and more robust solution to this issue is not to pay any attention to the command which the client issues, but rather parsing the traffic originating from the IMAP server for message bodies. This way fetch requests which don't cause any message bodies to be sent are ignored. Negative server responses such as "NO" or "BAD" are no issues either, as message bodies are not transferred and not detected. A consequence of this behavior is that messages can be scanned just in one direction: when they are transmitted from the server to the client. The effect of this one way scanning is described in section 2.8.

The only requirement for this solution is that a reliable way of finding the begin- and end-positions of messages in the traffic which is sent by the server has to be found. This can be tricky as if not the RFC822 message or an equivalent macro is requested as message format, the e-mail can contain arbitrary character sequences including symbols which have semantic meaning in the IMAP protocol. This has to be taken very seriously to prevent code injection. In such a scenario the attacker could fake a tag at the beginning of the message which signals the end of an IMAP response and then append any kind of malware to the message. The proxy would scan the message just until the faked end tag. This is not acceptable as it is an

exploitable design flaw. The solution to this lies within the header of the IMAP response. Before sending the message to the client, the server calculates the size of the message which is being transferred and places it's size in bytes in the message header. If an end tag is found within the message at a position which is smaller than the message size, that end tag is faked and must be handled as plain text without semantic meaning. The amount of bytes in the message header can be safely skipped before starting to scan for end tags without the risk of overlooking end tags which would be fatal as well.

The tags which have been determined to be usable are the following: The beginning of a message being transferred over IMAP can be determined by the string "FETCH (UID" followed by "BODY[" in the same line. Both strings must be present in order to get a valid start tag. The end of the message is usually locatable by searching for CRLF followed by “)” and CRLF. However, some mail transfer agents such as Courier Imapd do not put the “)” character in a new line in case of attachments which makes the search for this pattern unreliable. So the CRLF in the beginning has to be dropped and the remaining pattern 0x29, 0xD, 0xA which is equivalent to “)CRLF” delivers satisfactory results.

These two tags identify the body of a message which transferred over IMAP. After extracting the data between these two tags, it can be scanned for malware.



In this example start- and end tags have been marked by bold characters. The additional “)” characters have been inserted to show that even tough these characters are not escaped in any way, just the last one was recognized by the MUA due to to the RFC822.SIZE attribute in the header. The line below the end tag contains the command tag (a0006 in this case) and the “OK” response which indicates that the message was sent successfully.

In this sense messages have a structure which is similar to TLV – Type Length Value entries. The type is RFC822 [9], the length is specified by the RFC822.SIZE attribute and the value is the message itself. Internally the IMAP plugin handles messages in this format.
To understand the size attribute it is important to know that the RFC822.SIZE is not actually specified in RFC822. It is determined by formatting the message according to the internet message format (RFC822) and then counting it's size in bytes.

2.10. Partial fetching

The fetching of messages is of great importance for this thesis because messages are that part of the IMAP traffic which has to be scanned for viruses. The simplest way to fetch a whole message is to use the command “86 FETCH 1 body[all]” where 86 is a freely chosen number by the client and 1 is the index (UID) of the message on the server. The reply to this command will be the imap header, the requested mail header with the message body and finally the line “01 OK – fetch completed”. This last line indicates that the command which was issued with the number 01 was completed successfully. So far fetching messages seems to be straight forward.
In order to make things more complicated the Network Working Group which is responsible for IMAP 4rev1 decided to add an additional feature to the protocol called partial fetching. The idea behind partial fetching is that messages can be retrieved in several pieces. The syntax is the following:

```
FETCH <uid> BODY[<section>]<<partial>>
```

The command `“FETCH 87 BODY[1]<245764.16388>”` would fetch 16388 Bytes of the message with the UID 87 with an offset of 245764 Bytes.
Partial fetching is not optional, but must be supported by the IMAP server.
This feature adds the functionality to continue the fetching of a message at an arbitrary point if the connection was dropped. In such a case parts of the message do not have to be fetched twice. The environment in which this feature makes sense includes 56k modems, GSM data calls and ever so unstable WiFi connections over long distances during harsh weather conditions. In a network which provides sophisticated ISO OSI layer 2 and layer 4 solutions, partial fetching has no right to exist. One could argue that it is helpful in case of large attachments such as CD or DVD images. The counterpoint is that e-mail with attachments

was not designed for such amount of data. There are perfectly good solutions like scp or rsync to transfer large amounts of data from one host to another. In the worst case even FTP could be a better choice than sending large files over e-mail. Also depending on the configurations of the mail transfer agents between the sender of the message and the receiver the maximum message size is most probably limited to a few megabytes. The MTA Exim 4 for instance has a quite high default size limit according to it's config file:

```
"# Message size limit. The default (used when MESSAGE_SIZE_LIMIT # is unset ) is 50 MB",
```
[15]

In case of Postfix the limit is the same:

```
bitumen@tuxworld:~$ cat /etc/postfix/main.cf | grep message_size
message_size_limit = 52428800
```

These values were taken from a Debian GNU/Linux, version 6.0 installation. Typically this limit is set even smaller and can be reduced also by the maximum upload size of web interfaces in case of web mail. The point is, that in the worst case of a dropped connection almost at the end of a message being transferred causes about 50mb to be resent without using partial fetching. Typically this value is much smaller. If an administrator of an MTA decides to increase this setting to a much higher value, that results in a unique case which cannot be considered here and is likely to be a misconfiguration. As shown, with the exception of unstable network connections partial fetching is not needed.

Apart from the mentioned advantage in obsolete networks, this feature can raise issues in several other layers, one of the most important being layer 4 and TCP. TCP implementations have mature and stable code with support for the TCP sliding window: *"The well-known Sliding Window protocol caters for the reliable and efficient transmission of data over unreliable channels that can lose, reorder and duplicate messages.",* [16]

Trying to perform the job of a lower OSI layer in an upper layer compromises the functionality of the lower layer and most probably delivers worse performance. After all, the OSI layers were designed like this for a reason. If the MUA for instance requests a message in chunks of 16kBytes, TCP cannot increase the window size over 16kBytes because there is not that much data available at once. This is inefficient not just on OSI layers 2-4, but also on application layer due to the overhead in IMAP headers. Some IMAP clients such as Mozilla Thunderbird / Dovecot do not only use partial fetching, but also use a kind of window scaling. In other words the MUA may implement a sort of "IMAP sliding window" like TCP does on OSI layer 4 for all traffic. The MUA mentioned above starts to fetch with a window size of

16kBytes and upon quick response of the server it increases the size by 8kBytes until the maximum IMAP-window size of 64kBytes is reached. 64kByte is exactly the layer 4 packet size where TCP window scaling starts to be efficient and can utilize the available bandwidth properly. More information about TCP window scaling can be found in the book TCP/IP Illustrated, Volume 1 [37].

Instead of letting TCP do it's job, the IMAP window size is kept at maximally 64kByte and because TCP does not have more data at it's disposal the window size cannot be increased anymore. The result is a transfer of messages in chunks of maximally 64kBytes, but possibly just 16kBytes in case of a connection with rather big round trip times due to routing or using a proxy. This also means that every 16 or 64kBytes the system has to wait for the duration of the round trip time because of the issuing of the following IMAP fetch command after the last "OK – fetch completed" response from the server. The effect of this is visible on the following example:

A message with the size of 2MB (headers included) is downloaded over a connection with a bandwidth of 0.5Mb/s and a round trip time of 600ms which is an admissible value in case of a 3G connection. Without partial fetching the message can be downloaded in

$$2Mb / (0.5Mb/s) + 600ms = 4600ms = 4.6sec.$$

With partial fetching and calculated with the maximum window size of 16kByte the same operation takes

$$(2Mb / (0.5Mb/s))*1000 +(2Mb * 1024 / 16kByte) * 600ms = 80800ms = 80.8sec.$$

Due to this feature the fetching of the message took 80.8 seconds instead of 4.6 seconds. The communication was slowed down by a factor of 17.57 even in the case of a not too big message with the size of 2MB and where the initial window size of 16kBytes was not even considered!

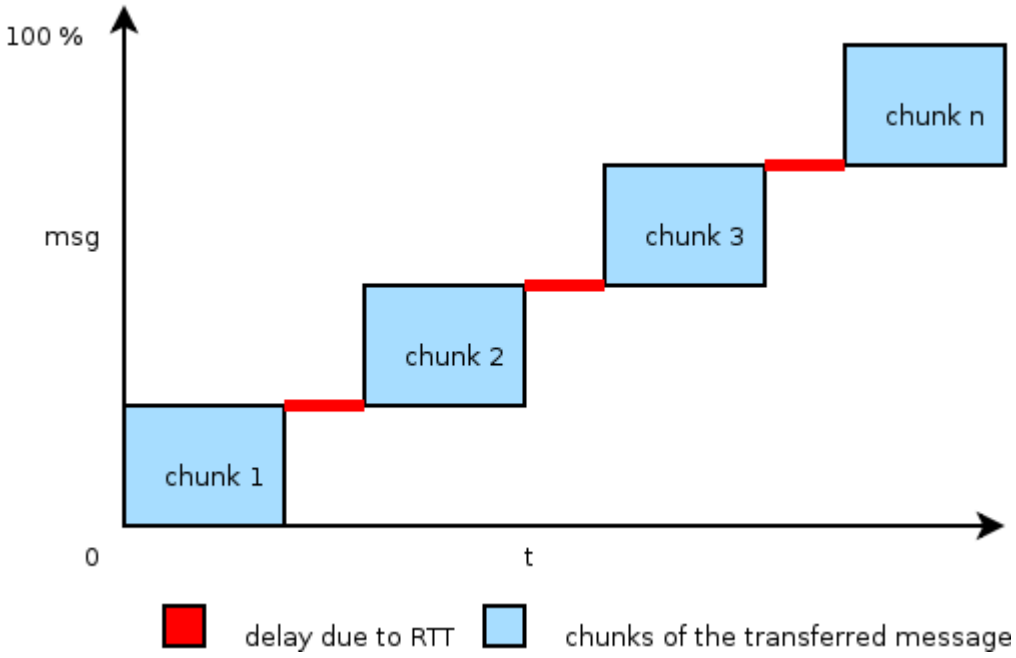


Illustration 13: Chunked fetching

The reason why partial fetching is mentioned in such extent in this thesis is that a proxy not only decreases the bandwidth due to virus scanning, but also increases the round trip time due to memory operations. The effect of combining partial fetching with a slightly higher round trip time have been shown to be fatal. For anything else than short plain text messages partial fetching has to be disabled in the MUA to get acceptable performance. However, there is one exception: If a message with considerable size is transferred and the connection is dropped, not all message has to be transferred again, because all previously successfully transferred parts can be used. The good news is that from the two MUAs supported by the software which is developed in this thesis, just Mozilla Thunderbird / Icedove has the described problem in the default configuration while Microsoft Outlook does not use this feature. If an IMAP proxy is used, the following setting has to be edited in the advanced configuration editor of Mozilla Thunderbird / Icedove:

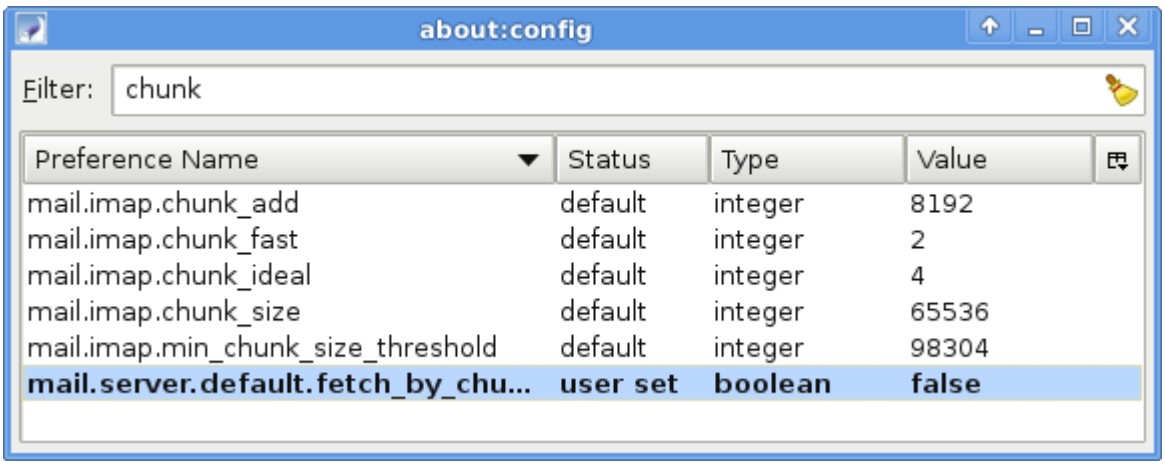


Illustration 14: Mozilla Thinderbird v3.1.9: fetch by chunks

Switching “mail.server.default.fetch_by_chunks” to “false” will probably reduce the duration of message downloading by a factor of 16.7 as calculated in this section.

2.11. Inserting a virus warning message

Until this point of the document it was discussed what could be potentially dangerous, where malware can be located within the IMAP stream and what the technical possibilities are to locate these parts of the communication. However it was not described how to proceed once an e-mail with malicious code was found.

The most obvious thought is that the message has to be prevented from reaching the mail client. One possibility is to simply drop the connection or not to send any more data of that message. This way the client is protected without question, but this cannot be a satisfactory solution as the MUA still waits for the message which it has requested to be sent. Within one connection several messages can be downloaded so with this method the delivery of legitimate, virus free messages would also be blocked. Dropping the connection could also be used to trigger a Denial of Service attack by sending virus infected messages [38]. Also it is unpleasant for the user if the MUA seems to stop working without an error message and mails cannot be downloaded. This behavior is also against RFC 3501 which states that a fetch command must be followed by untagged fetch responses containing the requested parts of the message, followed by a tagged response which informs about the success of the command's

execution. Also the developed software has to comply to the standard in order to guaranty interoperability.

The proper way of handling these situations is to copy the cached message which contains malware to a separate buffer, remove the malicious part and insert a message that parts of it have been removed due to security reasons. Determining which parts to remove can be done by separately scanning parts of the message. First, separate files have to be created from the MIME encoded, cached message. This can be done for example by a tool called “Ripime” [17] which can differentiate between the text part of the message and attachments. After having distinct files for the mentioned parts, they can be evaluated one by one by a virus scanner. Harmless parts can be assembled to a clean message. After inserting a note about removing parts, the message header has to be rewritten so that it would contain valid information about the MIME structure of the message. Also the encoding of the Internet Message Format has to be verified so that characters or character sequences within the message would not have unwanted semantic meaning in any upper layer protocol such as IMAP. The freshly assembled message can be written to the client.

The easier way to proceed is instead of cutting out malicious parts of the message, is to simply remove the whole message body and to insert a virus warning instead. Also in this case the message is copied to a separate buffer, which is used for manipulation. As during this scenario the header of the message will not be modified, it has to be left in it's original state and it is necessary to seek to the beginning of the message body. This position can be found quite easily according to the internet message format: *“The body is simply a sequence of characters that follows the header and is separated from the header by an empty line (i.e., a line with nothing preceding the CRLF).”* [18]

“CRLF” stands in this case for a new line. So in order to find the beginning of the message body the buffer has to be searched for the first occurrence of the Byte sequence 0x0A 0x0D. After these two bytes it is safe to simply delete the rest of the buffer's content thereby removing the whole message body. A user defined string can be appended as a virus warning message. In the developed application the string “Virus found. Mail body removed.” was used. The modified message is best terminated by “CRLF)CRLF”.

This is also visible in the following example. Here is a short message with IMAP header and tagged response included:

```
a0006 UID FETCH 476 BODY.PEEK[]
* 1 FETCH (UID 476 BODY[] {498}
Return-Path: <root@imap-server.localdomain>
X-Original-To: u8
Delivered-To: u8@imap-server.localdomain
Received: by imap-server.localdomain (Postfix, from userid 0)
.id E1B3A4C1AD; Mon, 29 Nov 2010 03:59:04 -0500 (EST)
To: u8@imap-server.localdomain
Subject: testmail
Message-Id: <20101129085904.E1B3A4C1AD@imap-server.localdomain>
Date: Mon, 29 Nov 2010 03:59:04 -0500 (EST)
From: root@imap-server.localdomain (root)

X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
)
a0006 OK Fetch completed.
```

Just for this example the Eicar signature is treated as a virus and the message has to be nullified:

```
a0006 UID FETCH 476 BODY.PEEK[]
* 1 FETCH (UID 476 BODY[] {467}
Return-Path: <root@imap-server.localdomain>
X-Original-To: u8
Delivered-To: u8@imap-server.localdomain
Received: by imap-server.localdomain (Postfix, from userid 0)
.id E1B3A4C1AD; Mon, 29 Nov 2010 03:59:04 -0500 (EST)
To: u8@imap-server.localdomain
Subject: testmail
Message-Id: <20101129085904.E1B3A4C1AD@imap-server.localdomain>
Date: Mon, 29 Nov 2010 03:59:04 -0500 (EST)
From: root@imap-server.localdomain (root)

Virus detected. Mail body removed.
)
a0006 OK Fetch completed.
```

Note especially the bold parts. The first line is the fetch request from the IMAP client followed by the untagged response from the IMAP server which starts in line two. The number in brackets “{}” is the RFC 822 size of the message. It is calculated by counting all bytes of the untagged response of the server starting from the e-mail header (not the IMAP header!). CRLF at the end of every line is included. This value has to be recalculated after modifying the message as the IMAP client reads this amount of bytes as untagged response. If this is not done, the MUA is likely to hang while waiting for a proper response.

The e-mail header which does not correspond with the message body and it's attachments can also be the source of issues for the MUA as it might not display the message correctly. If this occurs, the header has to be altered aswell.

2.12. Mail User Agents

After having gathered some knowledge about the inner working of the IMAP protocol and the IMAP plugin which has to be designed the question has to be raised what server and client software should be supported. Despite being comporatible with RFC 3501 programs using IMAP will do things slightly differently. This can cause issues with the IMAP plugin if it is not designed carefully enough. So first of all it has to be determined which IMAP servers and mail user agents should be supported. Of course it is nearly impossible to guaranty compatibility with all kinds of MUAs. To determine what software to support in order to work in most use cases a mail client statistics from lintian.com [13] has been used:
"Data collected from 250,000,000 email recipients using our Fingerprint analysis tool. This chart shows the top 10 email clients by market share. Compiled 24 February 2010.", [13]

Email client	**Market share**
Outlook	43.00%
Hotmail	17.00%
Yahoo! Mail	13.00%
Gmail	5.00%
Apple Mail	4.00%
Iphone	4.00%
Thunderbird	2.40%
Windows Live Mail (Desktop)	2.00%
AOL Mail	1.2%
Lotus Notes	0.40%
Others	8.00%

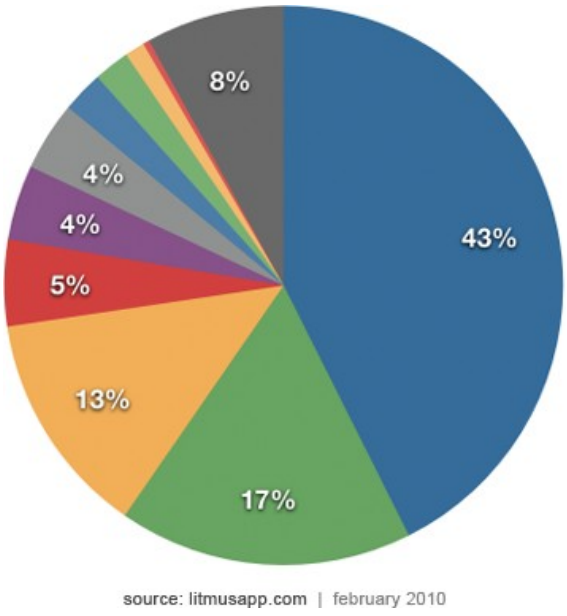


Illustration 15: MUA distribution

This data contains some information, which is misleading for us. Before using it some numbers have to be recalculated. As already stated, if the IMAP proxy is used it is expected, that the mail user agent software is run from within the internal, protected network and the IMAP server is located somewhere on the internet which is outside our network. This implies that the IMAP is "spoken" over a connection which goes through the firewall where the

IMAP proxy is located on. This assumption is correct as long as a desktop application is used as MUA. The table above however shows that at least 35% of e-mails are handled by web based mail systems (Hotmail, Yahoo! Mail, Gmail, etc.). In this case either these web based services fetch mail from other accounts via IMAP or POP3 which means that traffic is not routed through our firewall using these protocols or that they do not fetch mails at all because the MTA is included in the same system as the webmail interface. This is the case if addresses with the FQDN of the web based mail system is used like “@hotmail.com” or “@gmail.com”. However keep in mind that this refers only to these systems acting as MUA. If messages are retrieved by desktop clients such as Outlook or Thunderbird from the POP3 or IMAP servers of these online services obviously the traffic is routed over our firewall either in form of POP3 or IMAP.

The gist of all this is that web based MUAs have to be removed from the table above and the market shares of desktop based MUAs have to be calculated separately. The results of these calucations are the following:

Email client	**Market share**
Outlook	67.40%
Apple Mail	6.27%
Iphone	6.27%
Thunderbird	3.76%
Windows Live Mail (Desktop)	3.13%
Lotus Notes	0.63%
Others	12.54%

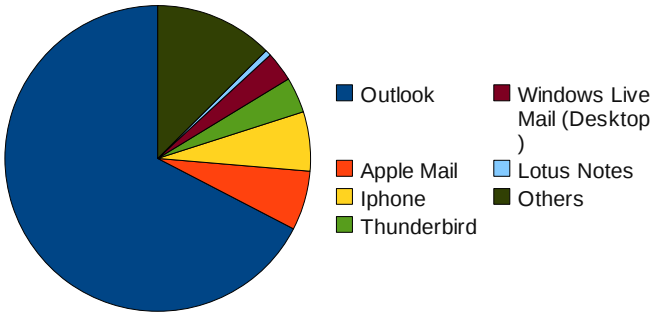


Illustration 16: Desktop MUA distribution

The consequence of these calculations is that it was decided to provide support for the MUAs Microsoft Outlook, Mozilla Thunderbird and Mutt. The choice of supporting the first two is based on the aim to support the biggest possible market share. The easiest way to perform tests of the developed software was a virtual machine with a very basic setup of Debian without a graphical user interface. This is the reason why mutt was mainly used in the first stages of development to test the proxy as Mutt is text based and needs very little resources. Other mail clients such as Apple Mail and Iphone will not be tested due to lack off access to them. Clients with no significant market shares such as Opera Mail or Eudora will not be tested either. Considering that 12.5% of mail clients are either unknown, are very rare clients or are other webmail clients it is possible to cover at least 2/3 of the market. Furthermore if there is no official support for a MUA that does not mean that it will not work with the IMAP proxy. Support in this case means just that software has been tested and is guarantied to be usable in combination with the developed product.

After writing most of the source code the alpha version of the TLS-Proxy's IMAP support has been tested. Opensource software has been working flawless. Using Mutt and Thunderbird issues or bugs have not been found yet. Microsoft Outlook however tends to drop the IMAP connection. The reason for this behavior has yet to be determined.

On server side Dovecot has been used during development. Testing was performed also with the Courier IMAP server [39]. Compatibility of TLS-Proxy does not depend directly on the MTA, but on the IMAP server. These two can be implemented by the same software solution. This is the case in the examples mentioned above.

Compatibility with other IMAP servers which have not been mentioned above has not been tested and can be the target of work beyond this thesis.

3. Implementation

3.1. An early attempt

After the first steps of functional analysis and design considerations a working proof of concept has been written in shell script (see appendix A). This script uses Perdition, which was also intended to be part of the final solution. First of all existing rules of Iptables are deleted that they would not interfere with this experiment:

```
iptables -t nat -F
```

Then IMAP connections to the TCP port 143 are matched by the prerouting chain:

```
iptables -t nat -A prerouting -p tcp --dport 143 -j DNAT --to 127.0.0.1:143
```

The prerouting [40] chain is the right place for this, because rules in this chain are applied while matching packets before the route is calculated or the packet is forwarded. This is the chain which has to be used for filtering for packages which are addressed not to the host, but are just routed over it.

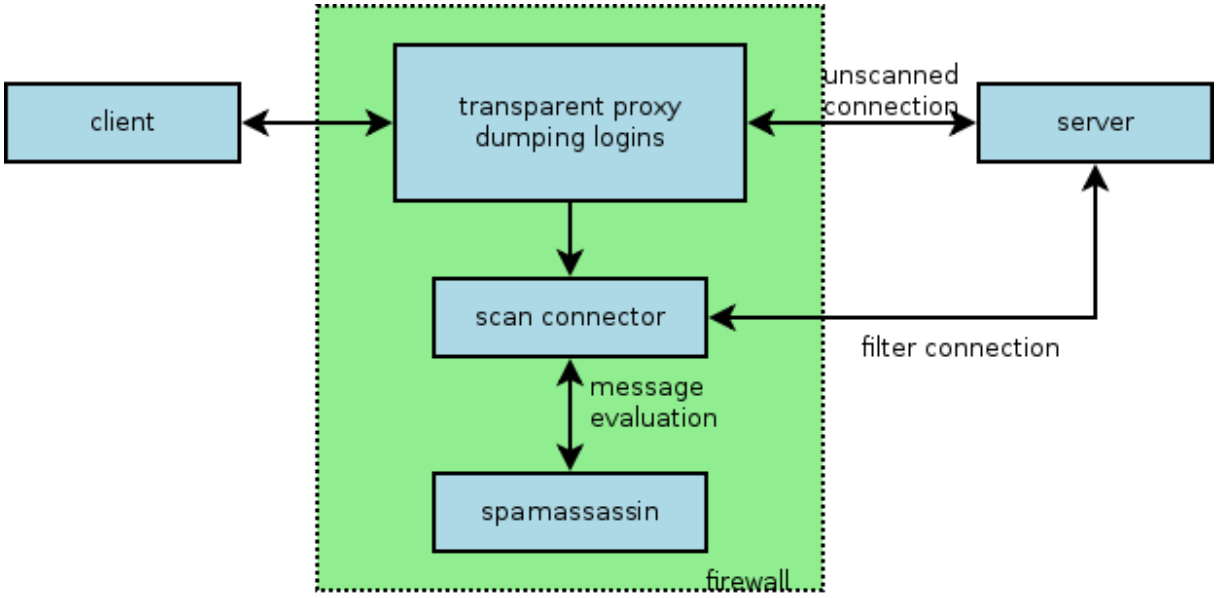


Illustration 17: IMAP-filtering architecture, early attempt

The matched connections are rerouted via DNAT to the perdition proxy, which dumps username and password of successfully established IMAP connections into the logs which are parsed. Login data is extracted as well. With this data a tool called Isbg [2] is called, which provides the functionality to connect to IMAP servers and to handle spam in different ways

such as marking, moving or deleting. Isbg is being run periodically after a given delay has passed. Simultaneously the client can transparently access the mail account over Perdition[4].:

It is an additional advantage, that the AS-line of Underground_8's anti spam appliances [41] use Spamassassin as well. This means that this script fits well into the software environment of the used systems. No extra software except for Perdition and Isbg has to be installed.
Disadvantages of this early attempt are the inelegant invoking of programs and extraction of login credentials from log files. Also a centralized way to manage settings and software options is missing. The biggest issue however is, that the client can establish a connection and check mails before the spam-scanning has been completed. This has to be fixed within the source code of Perdition and was not done because Perdition was abandoned within this thesis.
After careful inspection of the source code of Perdition some issues occurred. Perdition uses Vanessa libraries such as “libvanessa-logger0” or “libvanessa-socket-pipe” [5]. These libraries greatly simplify logging and handling of Unix sockets in the C programming language. Generally it can be said that they are useful. However in this case it is necessary to buffer IMAP traffic, to parse for strings which indicate the retrieval of messages and to deny the forwarding of traffic if it is dangerous. This cannot be done directly within the source code of Perdition as it calls the socket-pipe from the Vanessa [5] library and traffic does not necessarily directly pass Perdition in a way that it can be buffered or altered. After several unsuccessful attempts to dump network traffic in a useful way it was realized that if Perdition is used as Proxy server, also the Vanessa socket-pipe library has to be modified. Altering it would potentially break other packages. This was not acceptable as the library is used by other software within the given distribution as well due to dynamic linking [42]. A possible solution to this issue is to compile Perdition with statically linked libraries and by that accepting redundancies in the system.
The idea of using Perdition as a proxy server was finally abandoned.

3.2. Final design

As Perdition did not meet the given requirements the available options were either to write an own proxy server or to search for another existing one which can be extended by the required

features. The decision fell on an existing proxy server called TLS-Proxy [3]. The main advantage of it is, that it supports the trusted filtering of SSL/TLS connections if the provided certificates are installed on the client. This is basically a trusted man in the middle [43] feature. Be aware that in order to to have privacy, SSL/TLS is not sufficient and content encryption such as PGP [32] has to be used. The proxy uses an encrypted connection to communicate with the real server and uses it's own certificate to encrypt the connection between the client and itself. TLS-Proxy is a framework and currently contains support for the application protocols HTTPS and SMTPS as well as raw TCP connections for test purposes. During the creation of this thesis a plugin for IMAP over SSL/TLS (IMAPS) support was written. The following diagram illustrates the architecture of this software solution:

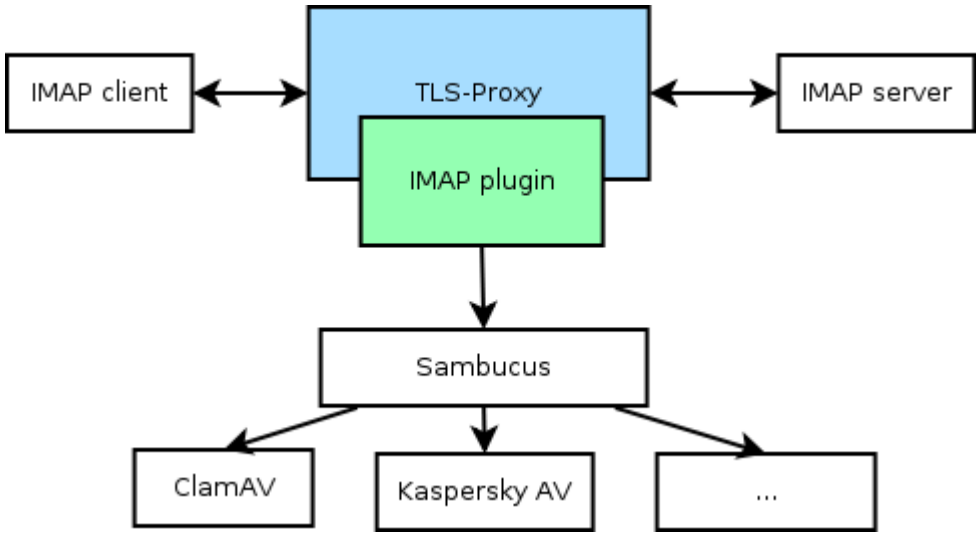


Illustration 18: TLS-Proxy entities

TLS-Proxy is responsible to handle the client- and the server-side of the IMAP(S) connection. Traffic is routed through the IMAP plugin which scans the connection for viruses. The attack vector of IMAP connection is the body of a retrieved message as this is the place where viruses can be transmitted. To identify these places in the IMAP traffic parsing for the keyword “FETCH (UID” followed by the UID in form of an integer and immediately followed by “BODY[]” is necessary in the stream, where the source is the IMAP server and the destination is the client. Like this incoming message bodies are identified. The filtering of the outgoing traffic is out of the scope of this thesis as the main concern is to protect the internal network from the internet and not vice versa. Other commands also exist to retrieve

parts of a message, like “FETCH” combined with “BODY.PEEK[]”, but viruses are essentially in the attachment of the message so infection is only possible if the whole message is retrieved and not just the header or the first few ASCII lines of it. The end of the message is identified by reading the amount of bytes which are specified in the beginning of the IMAP server response and indicate the message size according to RFC-822 and RFC-2822[6]. After that parsing for a line with the content “OK Fetch completed” terminated by CR LF is necessary.

If the beginning of the transmission of a message is detected, writing to the client side is blocked, until the content of the message is identified as harmless. That means that the message has to be buffered and written into a temporary file which is ripped of the mime extension and passed to the anti virus software for scanning. Currently ClamAV[7] is used for scanning, but in future the calling of an antivirus solution will be done by a currently unpublished scanning daemon called Sambucus. The advantage of this solution is that there will be just one API which has to be called for virus scanning, but the implementation of Sambucus can support multiple virus scanners internally. This means that more reliable results can be achieved by combining the analysis of at least two virus scanners. This additional layer of abstraction also results in flexibility: If one anti virus solution is swapped for an other, the code within the IMAP plugin of the TLS-Proxy does not need to be modified. It is sufficient to remove one scanner and add another one within Sambucus.

Depending on the result of the virus scan either the existing buffer is written to the client, or the content of the message is nullified. This is done by creating a new buffer which the original header of the message is copied into. Then a message has to be appended which states that a virus was found and the mail body inclusive attachments was removed. An additional feature is to include the number of infected files as well as the name(s) of the virus(es) found.

After that the line “OK Fetch completed” is appended after the current command number which is the response of the IMAP server and indicates that the execution of the command is finished and further lines do not follow. If the correct command number is missing or the message “OK Fetch completed” is not placed at the proper position IMAP clients may not work properly. This was observed with a command line e-mail client called Mutt[8]. It ended up being in an endless loop waiting for the end of the command complying to the IMAP 4rev1 RFC.

Before sending the nullified message to the client, the message size attribute has to be calculated and sent within the first line of the IMAP response. The IMAP size is calculated by counting the bytes of the message. This is referred often to as RFC 822 [9] size. If this value is not set correctly the behavior of the IMAP client depends on it's implementation. In case of Mutt the nullified message is cut to the specified size, which can result in a partly displayed message.

After completing these tasks the nullified message with the virus warning included is ready to be sent to the client. The IMAP plugin can do this by writing to the available file descriptor in the TLS-Proxy framework.

An additional capability of the IMAP plugin is to capture the command “STARTTLS” at the beginning of an IMAP session so that TLS-Proxy could start the trusted man in the middle feature for scanning encrypted connections.

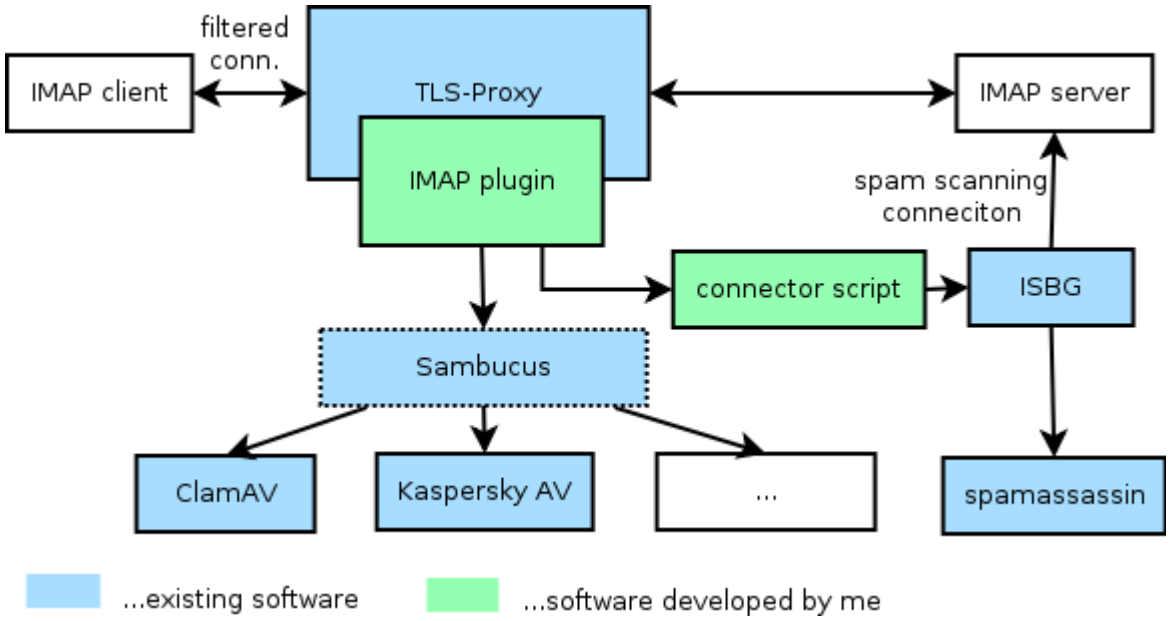


Illustration 19: TLS-Proxy entities exploded

This diagram shows the complete structure of the software components used for IMAP scanning. It includes not just the TLS-Proxy with the IMAP plugin and the virus scanning solution, but also the custom connector script for spam scanning. An early version of this script was described in section 2.3 and has been referred to as asynchronous online scanner. The main difference in this new architecture is that the connector script has to communicate with the IMAP plugin directly and does not get any login information from Perdition as it is not part of this thesis.

3.3. Flowchart: HandleClientSide

The TLS-Proxy scans and manipulates IMAP streams on the network if required. Streams have some properties which are not trivial to work with. For example streams can be processed just in one way. It is not possible to iterate backwards, to seek like in a file or an array. Sending of data cannot be undone. A proxy server has to be able to work under these conditions without causing major delays. Caching data in order to be processed in a block with specified size is thereby not recommended. Also in most cases it is unknown how much data will follow in the streams and when it ends.

The TLS-Proxy provides a method for every protocol which is responsible to receive and to send data. These are called “HandleClientSide()” and “HandleServerSide()”. The maximal amount of data which is received at once is limited to 64kByte. However, it does not have a lower limit. As data arrives over TCP, the buffer of the proxy is filled with it and the already mentioned methods are executed. As a consequence these methods receive data in form of chunks of arbitrary sizes. In the first, experimental design of the IMAP plugin every chunk was inspected and in case of a found start or end tag the whole chunk was considered to be part of a message. This worked well for test messages of less than the maximal buffer size of 64kByte. However if several e-mails were received after each other or in case of attachments which are usually bigger than 64kByte the IMAP plugin was destroying messages by cutting them at arbitrary positions and appending data from other chunks which did not belong to the message at all. It was necessary to develop an algorithm which deals with this inconvenient stream of data which arrives from the IMAP server. This issue could be solved by introducing a separate buffer in which messages are cached. Note, that no other parts of IMAP traffic but messages are cached like this. Also a flag was needed which showed whether the current position within the stream is within a message or not. The position was within a message if a “start tag” of the FETCH command had already been processed and an “end tag” not. The flag for this purpose is named “fetch” in the flowchart below which shows a simplified version of the used algorithm:

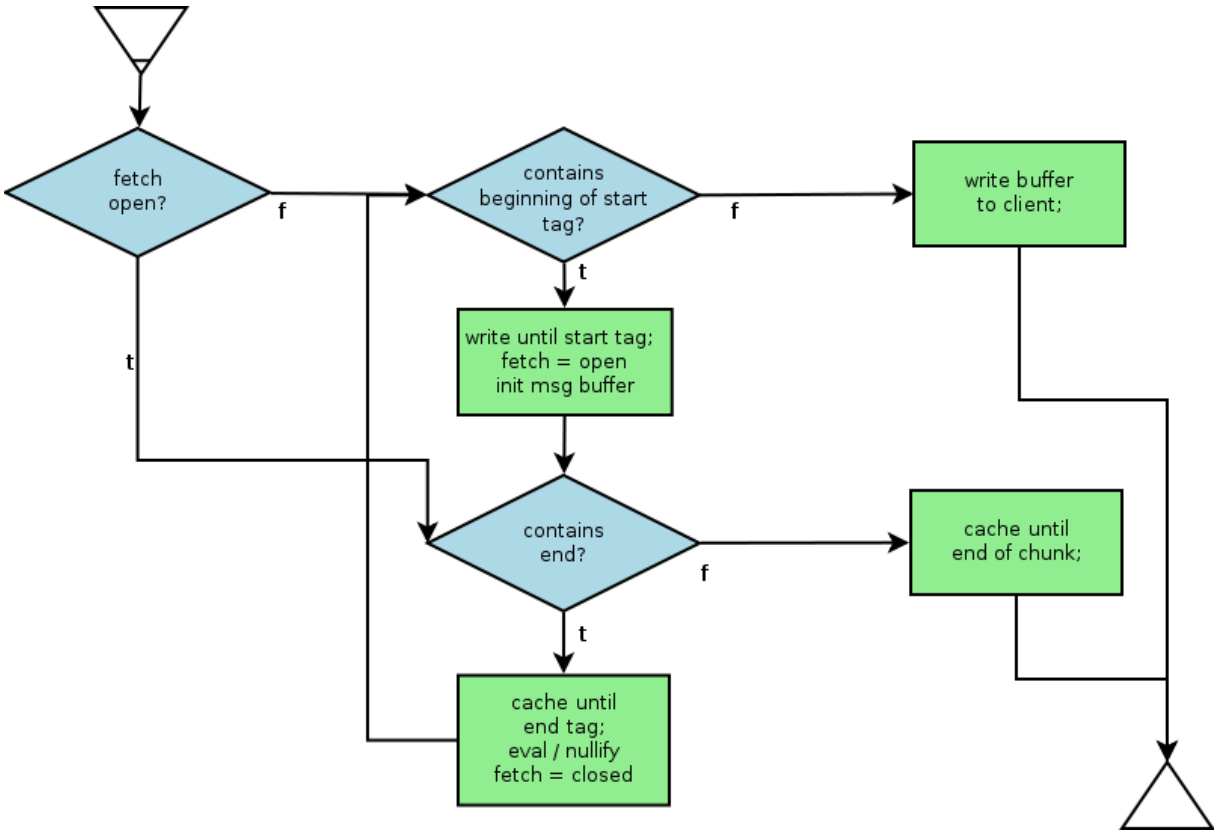


Illustration 20: Flowchart: HandleClientSide

Parts of the stream which are after an end tag and before a start tag must be written to the client immediately without being scanned or delayed in any other way. Messages must be detected even if they are fragmented over several chunks of data. This happens in case of attachments. The end of a message has to be detected so that caching the stream could be stopped and evaluation could be started. This caching and parsing over several chunks had to work very reliable as in any other case the usage of the IMAP plugin would result in destroyed messages which renders the software unusable. The algorithm which is shown on the flowchart handles this issue satisfactorily. Debug output of the retrieval of a message using this algorithm looks like this:

```
end check method, got attr: msgSize: 84243, msgBufLength: 68050, pos: 14630
end check method, got attr: msgSize: 84243, msgBufLength: 68050, pos: 14630
end check method, got attr: msgSize: 84243, msgBufLength: 68050, pos: 14630
end check method, got attr: msgSize: 84243, msgBufLength: 68050, pos: 16223
- caching until end (end found at 30854) pos=0
- eval msg
read filename: /tmp/imapB2AEug

result buffer: /tmp/imapB2AEug.dir: OK

----------- SCAN SUMMARY -----------
```

```
Infected files: 0
Time: 0.023 sec (0 m 0 s)

E-Mail clean.

- writing until end of chunk. pos=30854
```

In the example above the algorithm loops through the chunks delivered by TCP and searches for the end of the e-mail in the data stream. It caches data until position 3054 which is the end of the e-mail in the buffer. This cache is is then written to the disk, evaluated by anti virus software and the result which is located in a file is loaded into the memory. After considering the e-mail to be clean the whole buffer is written to the IMAP client.

3.4. Access control

One of the mandatory features of the developed program was some kind of access control. Several variants were plausible: Having configuration files, black- and whitelists, a web based management interface, different settings for the proxy itself and the online scanner seemed to be reasonable solutions. However the idea of disabling the scanning of specified accounts within the proxy was abandoned in an early stage of development: Beside logging login information, the main functionality of the IMAP proxy is to actually scan messages for viruses. Disabling this functionality is not a matter of convenience, but a matter of security. Reasons for disabling scanning are mostly the unwanted changes which are made to the IMAP mailbox on the remote server. This affects the online scanner, but not the IMAP proxy itself. This is the reason why it is mandatory for messages to be scanned by the proxy and not just an option.

For the online scanner however user management greatly affects useability as without it in case of a user, who does not want spam scanning the whole online scanner would have to be turned off. The internal working of this user management is the following: The online scanner uses two plain text files called “listAvailable” and “listEnabled”. While being invoked by the IMAP plugin of the TLS proxy, the online scanner receives a list of accounts with hosts, usernames and passwords. It dumps the mailboxes in the fomat “username@host”, one entry per line into the file “listAvailable”. This file can be read by an arbitrary management interface – like a web based user interface – to list the available accounts. The lines containing accounts where scanning is desired are copied into the file “listEnabled”. While being run, the

online scanner has the whole database of login information in memory, but scans just those accounts, which are contained in the file “listEnabled”. It is crucially important that these files do not contain the passwords of the accounts. Passwords are stored in an Sqlite3 [44] database within the IMAP plugin of the TLS proxy and are transferred to the online scanner's standard input. If an account is present in the file “listEnabled” but does not have the corresponding password in memory, it is ignored until the password is learned by the proxy on the fly. The file “listAvailable” provides just a source for accounts for the management interface and does not contain accounts which were added manually into the “listEnabled” file.

There are two issues of this behavior which have to be considered if the asynchronous scanner is used:

1. The IMAP plugin simply invokes the third party tool which is responsible to scan IMAP accounts for SPAM without monitoring it. The functionality of the IMAP plugin is not more than a trigger to start the scanning procedure and a safe storage facility for passwords. The consequence is that the validity of login credentials is not ensured. Passwords which are sniffed from the network traffic are sent to the third party scanner and may or may not be valid. An attacker who knows about this behavior could try to use a large number of incorrect login credentials which the scanner would try to use periodically. With a large enough number of logins this is considered to be a DoS attack [38].
2. The idea behind an in-memory database is that data which is not written to disks is safer. Also topics like encryption and file permissions are avoided thereby keeping the design simple an almost without drawbacks. The issue with data which is stored in memory for security reasons is virtual memory. If an attacker can manage to examine the hard-drive' s content, passwords can be revealed. Risk mitigation is done by turning off virtual memory (paging) or limiting the physical access to the hard-drive in the long term. Also non-pageable memory can be used to prevent login credentials to be swapped to the hard disk. A slight drawback of an in-memory database is also that login credentials have to be learned again after every reboot. The impact of this behavior is not big as proxies and firewalls should not be rebooted often in the first place.

3.5. Handbook for administrators

3.5.1. Quick setup of TLS-proxy with IMAP plugin

Before trying to run the developed software some thought has to be given to the network architecture which this system is run in. It is not trivial to set up a test system on one host. It is already clear from the previous chapters that what we are dealing with is a transparent proxy server. As a consequence it has to run on a host which is a gateway between the „real“ client and server from the application protocol's point of view. In order to understand this it is important to know how the „redirect“ command of Linux's Iptables [10] is used:

```
iptables -t nat -A PREROUTING
      -p tcp --dport 143
      -j REDIRECT --to-port 4430;
```

The effect of this is that all TCP traffic from port 143 which is routed through the gateway is intercepted and redirected to the port 4430 on localhost where the transparent proxy is listening. Note the importance of the “PREROUTING” chain in the command above: It means that this rule matches only packets which are routed from one network interface to the other. However it does not match packets which are sent by localhost. For that the “OUTPUT” chain would have to be used. If someone decides to use the output chain instead of the prerouting chain for testing purposes so that the proxy could intercept connections from localhost too, the person will encounter the following problem: A packet is intercepted on port 143, is redirected to the proxy on port 4430. After being processed it is sent by the proxy again on port 143 to the server, but it is intercepted again because the wrong chain has been used in the Iptables rule. This results in a continuous loop which is undesirable. The gist of this theoretical case is not to modify the chain of the packet filter and to use a clean setup with multiple hosts for testing purposes. However, by using two separate virtual hosts as client and server and realizing the network infrastructure with VLANs it is possible to create a clean single-host setup. A script demonstrating this can be found in appendix B. This setup is described in great retail in chapter 3.9.

After the variables in the script from appendix B are set to match the network environment and the script was executed the next step is starting TLSProxy:

```
./TLSProxy -P IMAPS -c /etc/tlsproxy/proxy.conf
```

The option “P” specifies the protocol to use. Other supported protocols include SMTPS and HTTPS. These are described in the thesis “TLS-Proxy” [3] and are not further explained here.
The option “c” specifies the configuration file. If it does not exist or this option is not specified at all hardcoded default values are used. The configuration options will be described in chapter 3.5.2.
To be able to intercept also encrypted connections two extra requirements have to be fulfilled: The client has to accept the certificate which is used by TLSProxy to encrypt connections and the proxy must be able to generate certificates. The first depends on the design of the client. Most mail user agents ask whether the received certificate should be accepted or not and some also offer to permanently accept it. It must be empathized how important it is to check whether the certificate's origin is really the TLSProxy and not another program trying to perform a man-in-the-middle attack.
The second can be done easily by starting the certificate store daemon by executing the “./CA” command in the corresponding folder. This daemon generates certificates for every connection and used to be part of TLSProxy. Meanwhile it has been extracted and made into a separate daemon [11].
By now TLSProxy should be up and running and should automatically intercept connections.

3.5.2. Configuration file options

Additionally to the undocumented options of the TLSProxy the IMAPS plugin supports following options, which have to be set in the configuration file of TLSProxy. If they are not specified, a hardcoded default value is used:

Option	Default value
ENABLE_ONLINE_SCANNER	yes
IMAP_SCAN_INTERVAL	300
IMAP_ONLINE_SCANNER_PATH	/usr/bin/onlineScanner.sh
IMAP_FORCE_LOGIN	yes
IMAP_HIDE_STARTTLS_CAP	No
IMAP_CLAM_DAEMONIZED	No
IMAP_USE_SAMBUCUS	Yes
HANDLE_UNSCANNABLE_AS_INFECTED	Yes
IMAP_PRINT_ALL_CLIENT_TRAFFIC	No

IMAP_PRINT_ALL_SERVER_TRAFFIC	No

- ENABLE_ONLINE_SCANNER: Enabling the online scanner is the option which turns on spam-protection. As already described above in this document, spam filtering is done by the asynchronous client and not by the proxy itself. If this feature is enabled spam scanning is done by a separate thread. Possible values are “yes” and “no”.
- IMAP_SCAN_INTERVAL: The IMAP scan interval only gains meaning if the online scanner is enabled. It specifies how much time has to elapse between the asynchronous client finishes scanning IMAP folders and is executed again. Specifying the scanning delay this way was intentional, so that the scanner could not be started while another instance is still running in case that scanning takes a considerable amount of time due to the number of mailboxes which are scanned. The value of this option has to be specified in seconds.
- IMAP_ONLINE_SCANNER_PATH: The online scanner path is, as the name already suggests, the absolute path to the executable online scanner script. In case that TLSProxy was installed from the Debian package this value should be already set correctly.
- IMAP_FORCE_LOGIN: Force login is an option which hides certain capabilities of the IMAP server which allow a challenge-response style authentication. You may want to prevent this kind of authentication because it makes it impossible to sniff username and password which are needed for the online scanner. If this option is disabled and the IMAP client decides to use challenge-response, the online scanner will not work.
- IMAP_HIDE_STARTTLS_CAP: Hiding the starttls capability is not really productive, results most probably in security holes by using a clear text protocol and was designed for testing purposes only. Severe problems can occur if this is combined with the force login option: If the authentication succeeds, username and password will be sent in clear text over the internet which should be prevented. However many IMAP servers advertise the “nologin” capability when communicating over an unencrypted channel. This means that authentication with username and password is not possible and other methods like NTLM are prevented by the proxy. The result is

that the client cannot connect to the server because of the bad proxy configuration. The intended way of using these features is to allow starttls and to force logins. Like this login credentials can be sniffed by the proxy and the connection is safe anyway because of the encrypted TLS layer underneath. The theoretical case that an IMAP server requires the usage of challenge-response authentication and does not support TLS is absolutely not common and suggests a misconfiguration of the IMAP server. However if TLSProxy has to be able to work with such an IMAP server, the "IMAP_FORCE_LOGIN" option can be disabled thereby allowing connections.

- IMAP_CLAM_DAEMONIZED: As already mentioned in the timing section 2.4. of this thesis, ClamAV provides two commands which can be used for scanning: Clamscan and clamdscan. One is faster, the other one does not require the ClamAV daemon to be installed. To give the decision which of the two commands to use into the hands of the administrator, the parameter "IMAP_CLAM_DAEMONIZED" was introduced. It can be set to the boolean values "yes" or "no".
- IMAP_USE_SAMBUCUS: As the IMAP plugin of the TLS-Proxy was designed to fit especially into MF security gateways also the antivirus solution was integrated, which is available in that environment. It is called Sambucus and has an own configuration parameter: "IMAP_USE_SAMBUCUS". If it is set to "yes", the Sambucus daemon is used to evaluate messages. This parameter was necessary to provide usable functionality also in other environments. If it is set to "no", ClamAV is used.
- HANDLE_UNSCANNABLE_AS_INFECTED: This parameter was introduced quite late and specifies how the TLS-Proxy behaves in a special case. ClamAV can terminate prematurely without returning results. This happens if the system runs out of memory while a scan is being performed. In this case the IMAP plugin caused a segmentation fault and terminated due to the lack of input validation while opening the scan results. During the design phase of adding proper input validation the question raised what should be done with e-mails which could not be scanned. On one hand if they are treated as infected it is likely that they will cause a false positive. On the other hand if it is assumed that the messages do not contain malicious code it is possible that a virus enters the network. Neither is a good solution and this issue can be easily prevented by running the IMAP plugin on a machine which sufficient

available resources. However if the system runs out of memory, the administrator should be able to decide how the proxy behaves.

- The parameters “IMAP_PRINT_ALL_CLIENT_TRAFFIC” and it's counterpart “IMAP_PRINT_ALL_SERVER_TRAFFIC” are not present in the default configuration file. They can be inserted manually to override the hardcoded default value which is “no” for both parameters. The functionality which can be enabled by setting them to “on” is the dumping of all IMAP traffic which passes the proxy to stdout in plain text without TLS. This was very useful during development in order to see what the proxy actually does. However these options could be easily used for eavesdropping and are neither present in the default configuration file, nor in the binary which is intended for distribution, for this reason. If access to TLS-Proxy with IMAP support is provided to the public, this functionality may not be compiled into the binary in order to prevent script kiddies to take advantage of this functionality.

3.6. Debian package

As this thesis was written not just for academic purposes, but also for a company developing firewalls and other network security appliances it was an objective to enhance the usability of the developed software. During the process of development files were put to many different locations on the file system. This is acceptable on the host which is used for development, but it is not on a production system. Instead of providing the software just in form of source code, it was decided to build precompiled packages for the i386 architecture in form of Debian archives. Installing and removing the software is done by DPKG [45] , which is considered to be an elegant way of managing software. The advantages of this format include the simplicity of creating archives for newer versions or other CPU architectures. The building process is automatized by a simple shell script which calls “dpkg-deb”.

The most important part of a Debian archive is the control file. It specifies the package name, version number, dependencies and the maintainer of the package. It has to be created very carefully as erroneously specified dependencies can cause malfunction of the software. The whole power of the Debian archive management lies in the careful specification of dependencies. If dependencies or their version numbers are set incorrectly, dynamic linking will not work in case of libraries and the program will not run.

The current control file of the TLS-Proxy package is the following:

```
Package: tlsproxy
Version: 1.0-1
Section: base
Priority: standard
Architecture: i386
Depends: libsqlite3-0 (>=3.5.9-6), bash (>=3.2-4), iptables (>=1.4.8-3),
gawk (>=1:3.1.7.dfsg-5), python, libc6 (>=2.11.2-7), ripmime (>=1.4.0.9-1),
isbg (>=0.99-1)
Maintainer: Macskasi Csaba <bitumen@tuxworld.homelinux.org>
Description: Transparent proxy with following features:
 TLS MITM support, HTTPS, SMTPS, IMAPS with on the fly virus
 checking, spam scanning online scanner
```

Note that the Python package does not have a version number because it is a metapackage. It is so to say “implemented” by arbitrary implementations of the Python language. The version numbers of the dependencies were copied from the list of installed software on the development host which was up to date. TLS-Proxy may work also with older versions of the listed software. However this was not tested as systems should be kept up to date. As TLS-Proxy is responsible for virus- and spam scanning it is usually run on a firewall located on the border of the network which should be protected. It is one of the critical points regarding security. At least security updates should be performed frequently. However in case of a freshly developed firewall appliance it can be expected that recent software is used which also means that the dependencies stated above should not prevent any reasonable system from installing the archive.

On the development system a source tree exists, from which the binary files are copied by a shell script to the correct subfolders within the building environment of the archive. After also copying the control file into it's folder the package is built by executing the command “dpkg-deb –build”.

A Debian package has to satisfy strict expectations to comply with Debian standards and to eventually become part of the official Debian archives. A tool named Lintian exists which checks packages for compatibility with these requirements. It was designed specially to grant the high expectations of the Debian distribution. [13]

The TLS-Proxy package which was built as part of this thesis has also been checked by Lintian. Several shortcomings exist which will most probably not be fixed. There are no intentions to write man pages [36] for the whole application nor is it planned to make the whole directory structure to conform Debian guidelines. The main issue is that TLS-Proxy is not free software and it is not possible to make arbitrary changes to the whole software as it is under

copyright. This part is well documented in form of this thesis. TLS-Proxy is not free software, it will not be distributed freely and has no chance of becoming part of the Debian distribution. It is intended for internal use on firewall appliances only. Furthermore full Debian compatibility was not part of the requirements for this thesis.

Originally it was intended to split the package into several sub packages so that features could be installed separately. That way packages like tlsproxy-base, tlsproxy-https, tlsproxy-smtps and tlsproxy-imaps could be built. The issue regarding this otherwise elegant modularized architecture is the following: When compiling TLS-Proxy, the binary either contains certain features or it does not. If the binary is included in the base package but the other packages are not installed, the program will not work and produce an error when being invoked. However if the binary is included in the plugin packages, there is no way to determine which package contains the main binary. The only reasonable solution is to create just one package containing all modules, all features. The split architecture can be achieved only if the source code is modified. This is not part of the thesis.

The current version number “1.0-1” was freely chosen. The Debian build version which is the part of the version number after the dash is one, as it is the first try to build this package and it has not been distributed yet. The version number of TLS-Proxy itself is freely chosen aswell as it's version is neither indicated in the documentation nor in the source code.

The current package (v1.0-1) contains next to the TLS-Proxy also the asynchronous scanner which is responsible for scanning spam on IMAP accounts.

It is intended to build virtual packages for all plugins of the TLS-Proxy in the future. As all binaries will be contained by the main package. These would be used just for additional dependencies, configuration files and INIT scripts.

3.7. Dependencies

In order not to reinvent the wheel, TLS-Proxy and the IMAP plugin depend on libraries and other software. This helps to keep the design of the software clean and without redundancies. The availability of libraries was not an issue because all which were needed can be installed from Debian packages. The IMAP plugin requires just one additional library in comparison to the TLS-Proxy without IMAP plugin. This library is libsqlite3, which is used to store the extracted login credentials in memory. If TLS-Proxy is built from the source code, the

configure script checks for the availability of the required libraries before the software can be compiled. If TLS-Proxy is installed from a Debian package, the package management system of Debian (Dpkg) will install all dependencies which are specified in the Debian package of TLS-Proxy.

However there was an issue in case of two dependencies of the IMAP plugin: The availability of the tools “Ripmime” [17] and “Isbg” [2] was not checked during compile time as these programs are not included in form of libraries into TLS-Proxy, but are called as binaries instead. Also there was no Debian package available for these programs neither in any GNU/Linux distribution, nor on open source software websites. Including these binaries into the Debian package of the TLS-Proxy did not seem to be an elegant solution because they are third party software products and one should have the choice to install them independently. It was decided that separate Debian packages have to be assembled for these two tools, which can be listed as dependencies in the package of the IMAP plugin.

After compiling Ripmime the following control file was created:

```
Package: ripmime
Version: 1.4.0.9-1
Section: base
Priority: standard
Architecture: i386
Depends: libc6 (>=2.11.2-7)
Maintainer: Macskasi Csaba <bitumen@tuxworld.homelinux.org>
Description: Tool to remove MIME extentions and extract the contained
 files to a directory. It is mostly useful to automatically process e-
mails.
```

As Ripmime is a simple C program without any extra dependencies, just the C library was listed as dependency which is available on every system. Packages were built for the architechtures i386 and x86_64 (amd64) which should cover the majority of CPUs running TLS-Proxy. For Isbg, which is a Python script used for scanning remote IMAP mailboxes for spam, the following control file was created:

```
Package: isbg
Version: 0.99-1
Section: base
Priority: standard
Architecture: all
Depends: spamassassin (>=3.3.1-1), python-openssl (>=0.10-1), python2.3
(>=2.3.5-16) | python2.4 (>=2.4.6-1+lenny1)
Maintainer: Macskasi Csaba <bitumen@tuxworld.homelinux.org>
Description: Tool to remotely scan IMAP mailboxes for spam. Unwanted
 messages can be moved into the spam directory if configured so. Support
 for SSL is included, but not for STARTTLS. Afaik isbg has poor
 error handling.
```

Isbg depends on spamassassin which is used for spam detection, and on Python binaries with SSL support for IMAP over TLS/SSL. As Isbg is a script and is not compiled for any specific architecture, this package fits for every possible hardware architecture which provides an interpreter for the Python language.

After building these packages the dependencies of the IMAP plugin are satisfied by available packages so the following line was adjusted in the control file:

```
Depends: tlsproxy (>=1.0-1), bash (>=3.2-4), gawk (>=1:3.1.7.dfsg-5),
python, ripmime (>=1.4.0.9-1), isbg (>=0.99-1)
```

The issue of dependencies was hereby elegantly resolved. Version numbers of the listed packages were taken from the installed software on the development computer. The version numbers are not necessarily the minimal requirements. However a host which is running a Debian based distribution from the stable branch can satisfy the dependencies easily.

3.8. INIT scripts

Integration of the developed software into an existing environment is an important part of this thesis. The TLS-Proxy is different from software which is directly used interactively by the user. Apart from the binaries being copied to the directory hierarchy on the host's file system launching and stopping the application is more complex. TLS-Proxy should be started in previously defined runlevels automatically because it is run as a daemon and it has to be terminated correctly when the host is shut down. In Unix-like environments there are in general two alternatives of doing so:

- System V which is still being used in *BSD systems and
- Init which is typical for GNU based systems such as GNU/Linux.

Note that Solaris style services were not considered.

As the MF Security Gateway which the TLS-Proxy should be run on is using GNU/Linux as operating system, it was decided to write a script for the INIT system. This script can be used for 4 use cases:

- Starting,
- stopping,
- restarting the deamonized TLS-Proxy and
- giving information about it's current status (running / stopped).

An aspect which also had to be considered is that the TLS-Proxy currently supports the application protocols IMAPS, SMTPS, HTTPS and raw TCP connections. Support for all these protocols is located within the same executable file. However, one instance can handle just one protocol, which is specified in a command line parameter. So in order to enable support for IMAPS and SMTPS at the same time on a security gateway, two instances of TLS-Proxy are required. According to the original design, each protocol should have an own config file with protocol specific parameters. Starting all services would look like this:*

```
$ TLSProxy -P IMAPS -c /etc/tlsproxy/imap.conf &
$ TLSProxy -P SMTPS -c /etc/tlsproxy/smtp.conf &
$ TLSProxy -P HTTPS -c /etc/tlsproxy/http.conf &
```

In order to enable or disable support for these protocols independently three separate INIT scripts are needed, each controlling a separate instance. All three protocols use the same binary and PID-files are not used by TLS-Proxy, which means that stopping one can not be done by issuing the command “killall TLSProxy”, or all three instances would be killed. In order to differentiate between instances it is necessary to parse the command line parameters which were used while executing the program. This information can be extracted from the output of the “ps” utility by using the following bash commands:

```
$ tmp=`ps ax | grep TLSProxy | grep $PROTOCOL`;
$ pid=`echo $tmp | awk '{ print $1 }'`;
```

The process ID of the TLS-Proxy instance which uses $PROTOCOL will be located in the variable $pid. This way the process can be safely terminated by issuing “kill $pid”. Such an INIT script has been written just for IMAP support. Modifying it to support other protocols consists just out of changing variables in the “config” section of the script. The whole INIT script can be found in appendix “C”.

3.9. Testing architecture

It is well known, that testing is an important part of software development. This is especially true for this thesis as the IMAP plugin of the TLS-Proxy which was developed is planned to be used in a commercial product. It will not be an application which is run by the end user and can be restarted any time it crashes so it has to be ensured by intensive testing that it works

* Note: The dollar sign „$“ at the beginning of a line represents the prompt of the command interpreter and not the beginning of a variable. For variable assignments „$“ is not needed in GNU/Linux.

reliably also in the long term. It will be installed most probably on the MF series security gateways [19], which should be able to run over a very long time without user intervention or forced reboot. This is especially important regarding memory leaks which are present in many applications written in C language. However, memory leaks turned out not to be an issue.

The first part of testing begins already during development. Instead of running the TLS-Proxy on an MF security gateway, a virtualized network was created for this purpose. The environment of the proxy was given lots of thought due to architectural limitations which are also stated in the kernel documentation: *"Transparent proxying often involves "intercepting" traffic on a router. This is usually done with the iptables REDIRECT target; however, there are serious limitations of that method."*, [20] This means, that the proxy has to be run on a router between source and destination of the traffic. The interception of the packets is done by the packet filter called Iptables which is built into Linux kernel inside the PREROUTING chain. The command for this is typically the following:

```
iptables -t nat -A PREROUTING -p tcp --dport 143 -j REDIRECT --to-port <port>
```

This firewall rule matches all packets which are routed by the system and where the destination port is 143 which is the standardized port of IMAP. Such packets are redirected to localhost to a port where the used transparent proxy listens. Note that the original destination address is still being stored by the kernel, so the transparent proxy can use this information to send the packet to it's intended destination after being processed. Packets, which are sent from localhost are matched by the OUTPUT chain of iptables and are therefor ignored by the redirection command above. This is true also for packets whose destination is the host where the transparent proxy is being run, as these are matched by the INPUT chain. The consequence is that a transparent proxy under Linux must not run neither on the source host, nor on the destination host of traffic in order to work as intended. Also it must be run on the same host where the traffic is intercepted because the original destination can be recovered only from the kernel which is inaccessible for other hosts. An exception are reverse proxies in which the destination is known in advance because it is static. In that case instead of redirect, a destination network address translation (DNAT) can be applied to forward the traffic to another host which the reverse proxy runs on. An alternative design is to run at least the asynchronous scanner on another host with a separate public IP address. In this way by analyzing the log files on the IMAP server it can be determined whether a real user has

established a connection with a MUA from behind the firewall which runs the transparent proxy, or the log entry has been caused by the asynchronous scanner without user intervention. This feature needs two public IP addresses. In many real world scenarios just one IP address is available which was the reason why this idea was abandoned.

Also a convenient single-host solution is needed that the whole environment could be stored on the laptop, which I use for development. These requirements are fulfilled with the following setup: The host which is used for development runs also the proxy and provides the redirect rule in the firewall. This is rather convenient as after saving and compiling the source code the software is run on the same host where it is located and does not have to be copied to a separate router. This host provides resources for two virtual machines: The IMAP server and the IMAP client (MUA):

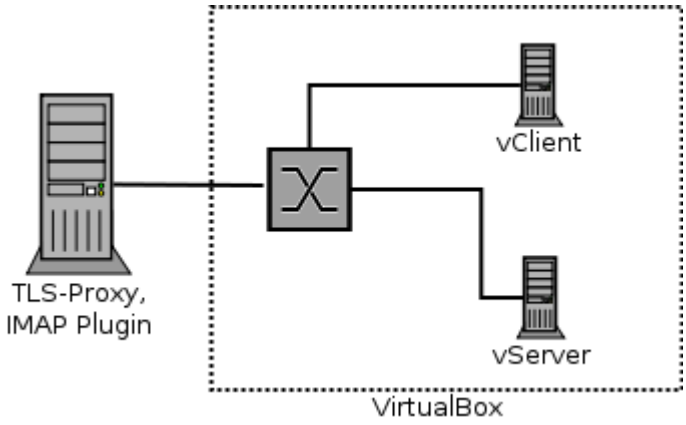


Illustration 21: environment during development

These virtual machines are completely separated from each other and from the development host's real network despite using the same physical interface of the host computer. This is done by adding separate VLANs [46] to the network interface of the host computer. Through this the development host has two additional VLAN tagged interfaces. For virtualization the open source software “VirtualBox” [21] is used. A useful feature of it is, that virtual network interfaces can be bridged not only to real physical ones, but also separately to a specified VLAN on the physical interface. This way the virtual machine gets an untagged port which is connected to a VLAN interface of the host computer. This is convenient as it is not necessary to deal with VLANs on virtual computers. The network setup on ISO OSI layers 2 and 3 looked like this:

Host	*L2 network*	*Address*
Tproxy / development	11, tagged	192.168.11.1
Tproxy / development	12, tagged	192.168.12.1
ImapServer	11, untagged	192.168.11.2
ImapClient	12, untagged	192.168.12.2

The script, which sets up the virtual test network according to these parameters can be found in appendix B.

In this design the ImapClient, which is in the network 192.168.12.0/24, has to use the development host in order to connect to the ImapServer in the network 192.168.11.0/24. This setup is suitable to check for basic functionality and for experiments with the IMAP protocol. An additional feature is that the development host has an interface on both sides of the proxy. The interface “eth0.11” is in VLAN 11 and “eth0.12” in VLAN 12. This makes it possible to sniff the traffic which is sent from the client or the server and also the traffic which was manipulated by the TLS-Proxy. Comparing the two streams is a good way to analyze the functionality of the developed software on a low level. Doing so made sense only when the communication was done without TLS. Testing and verifying the IMAP stream while using a layer of TLS is just possible when enabling the dumping of traffic within the IMAP plugin. The result is that both the client and the server were using encryption, but the whole traffic is dumped to stdout by the TLS-Proxy. For an admin with malicious intentions this is an easy way to perform a person-in-the-middle attack on encrypted IMAP.

This way of testing the software is reasonable until basic functionality works without errors, the performance tests are done and the software seems to be ready to use. However as in case of every software there can be still hidden bugs which are hard to find and could stay hidden while testing in a lab environment. An additional way of testing was needed. Because this software is not intended to be used directly by users or system administrators and will rather be installed on security appliances by professionals, monkey testing has not been performed. However a long term test under real conditions is mandatory. There was a host with an MTA, an IMAP server and several users in an existing network environment which seemed to be the right equipment for the long term test. However integrating the TLS-Proxy into this environment is far from trivial. As already mentioned, a transparent proxy running on Linux needs to be placed on a router. In the existing infrastructure the only router is an appliance from Linksys with NAT, port forwarding and a rather limited firmware. This router forwards

the ports 25 and 143 to the server which was in the LAN with a privat IP address. The TLS-Proxy cannot be placed between the router and the server as the router performs DNAT during port forwarding and if IMAP traffic is forwarded to the address of the TLS-Proxy instead, the packets will never reach the IMAP server.

The solution to this issue was to use again two virtual machines: One was a router with TLS-Proxy installed with two virtual network interfaces, the other one contained just an IMAP server and had two network interfaces as well:

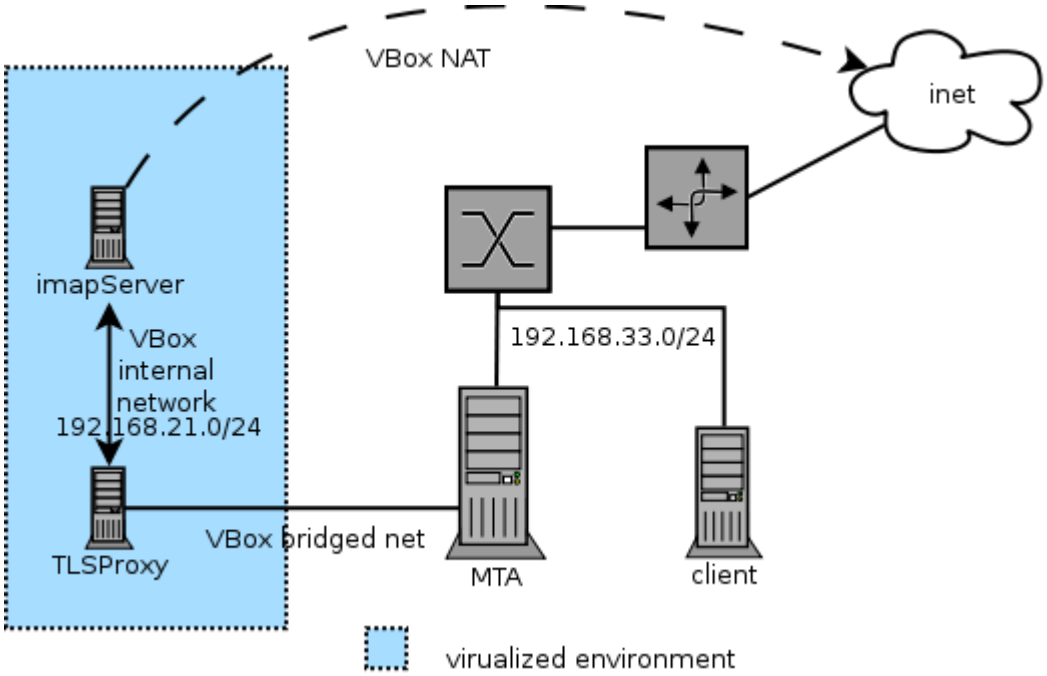


Illustration 22: environment during long term test: network architecture

The trick to direct traffic over the TLS-Proxy was to use an internal network between the virtual IMAP server and the virtual router. This way clients had to connect to the virtual server over the virtual router. The virtual server however needed to get mails from the real server in order to act as an IMAP server. To avoid double checking this was not done over the virtual router, but rather over the second network interface. That virtual NIC was connected over the NAT feature of VirtualBox. This way the virtual server could fetch mail from the real server with the IP address of the host computer's network interface. The fetching has been done over the IMAP protocol using a mail retrieval tool called “fetchmail” [22]. On client side the address of the IMAP server had to be changed to the address of the virtual server. The following figure shows an overview of the flow of information in the test environment. Red

lines symbolize unscanned SMTP / IMAP traffic, while the green one stands for traffic, which was scanned by the IMAP plugin of the TLS-Proxy:

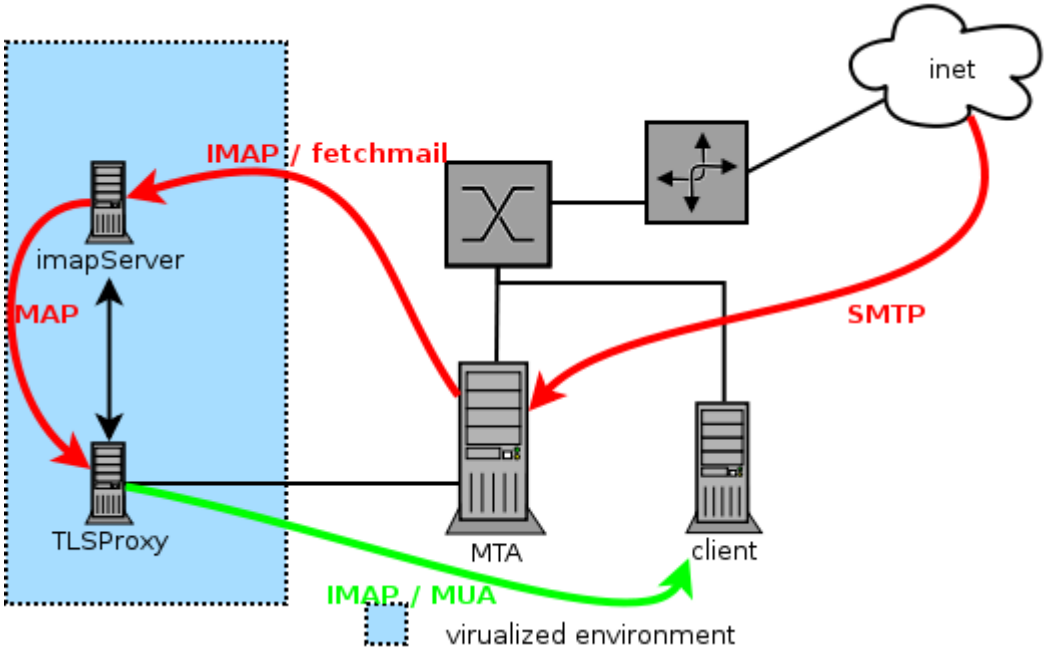


Illustration 23: environment during long term test: flow of information

Until now the IMAP plugin has been working as expected. Apart from the issues described in section 4.2. of this document additional bugs have not been found yet.

3.10. MUA test results

Using the architectures explained above tests have been performed with several mail user agents. Important requirements were that the developed software would not damage messages during retrieval, that usable performance would be provided and that all the developed features must work flawlessly. Also the software may not crash as in it's natural environment uptimes of several months or even years are expected. The performed long term tests show that the proxy with the IMAP plugin is still working after more than one week of usage. Errors have not been logged. Much longer explicit testing is not possible within this thesis. However the developed software will continue to run in the virtualized production environment and will be used in future. This way testing will be done simply by reading lots

of e-mails every day. If issues occur they will be noticed because of the production environment around it.

Some parts of the testing process were documented using screen shots. These were used to show the functionality on the following pages.

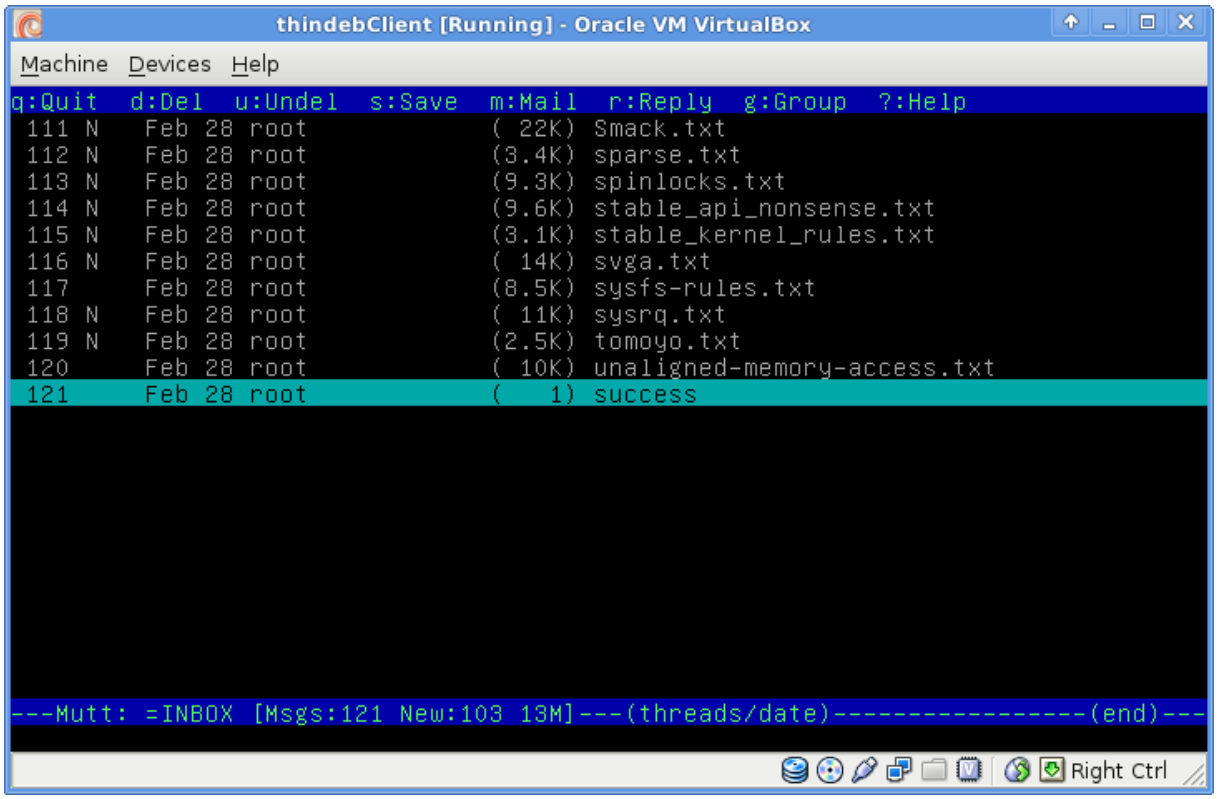


Illustration 24: Mutt

This picture shows the mail user agent Mutt after retrieving headers of 121 test messages Mutt is the most tolerant mail client and showed already successful results during development where other mail clients failed due to incorrect IMAP responses.

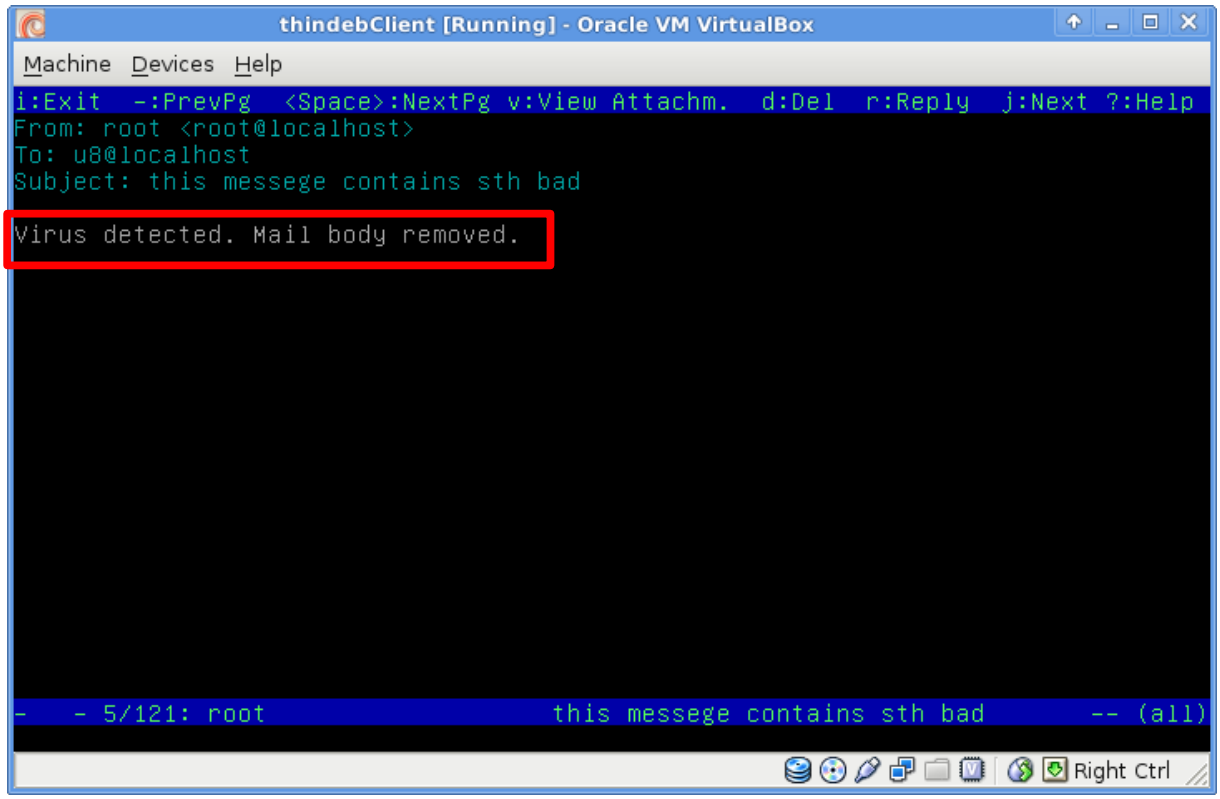


Illustration 25: Infected Message

This instance of mutt wanted to fetch a message which was infected by a virus. The malware has been scanned, identified and removed from the message. A warning has been sent instead of the original message body. The warning message is marked with red color. This scenario shows how the IMAP plugin of the TLS-Proxy protects the network behind the firewall.

Testing of ZIP archives is not part of this thesis, because the result depends only on the used antivirus software. The IMAP plugin does not unpack ZIP archives. Apart from that, password protected ZIP files cannot be unpacked for analysis.

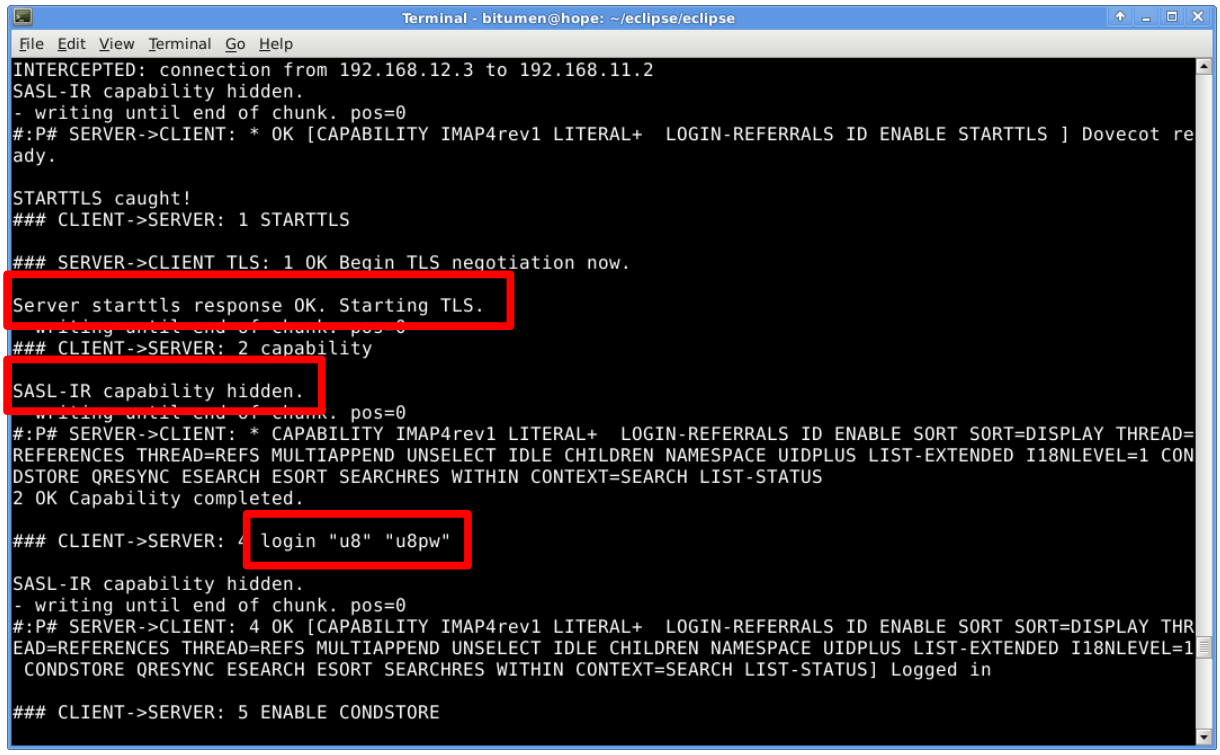


Illustration 26: IMAP plugin, debug output

This terminal window was captured in debug mode which is very verbose and prints the whole network traffic on application layer as well as debug messages of the IMAP plugin. It is visible that the client issued a STARTTLS command. This triggered the proxy to start TLS on server- and client side in order to encrypt traffic (red). After this logging in via challenge-response protocols has been disabled. As a consequence the MUA has to use login credentials (“u8” and “u8pw”) which is sniffed and parsed by the IMAP plugin (red).

These extracted usernames and passwords are kept in memory only. Unless an attacker has access to the memory or to the page files (virtual memory) of a running system they are safe. Of course debug mode has to be disabled as it does not make sense to dump login credentials to stdout which is a serious security risk. If the online scanner was enabled, it would be called directly by the IMAP plugin with the necessary login parameters.

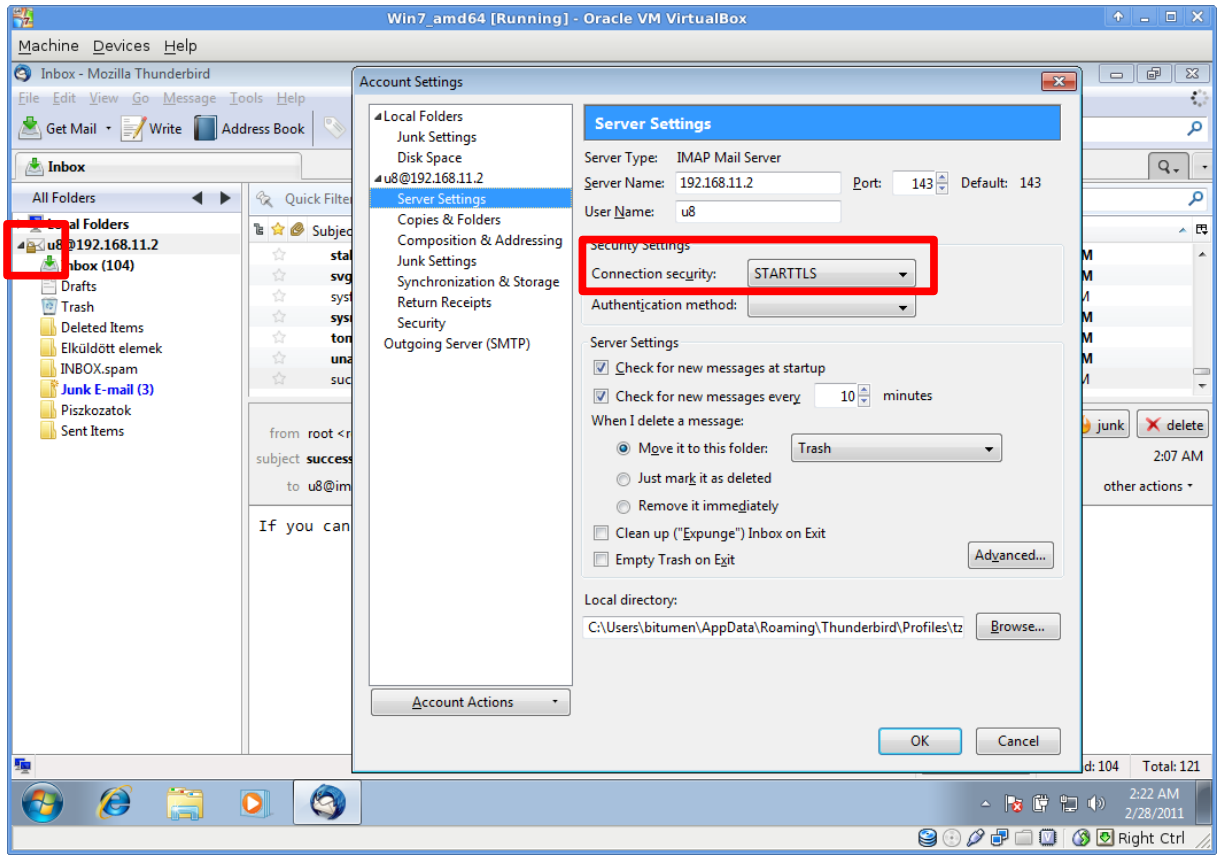


Illustration 27: Testing STARTTLS

The screenshot above shows Mozilla Thunderbird operating with STARTTLS as security setting. Messages are retrieved without problems. Thunderbird is compatible with the developed IMAP plugin and the TLS-Proxy. Note the small lock symbol within the red square. It shows that the connection to the IMAP server is secure.

The same mailbox has been used to test all mail clients. It contained various documents from the documentation of the Linux kernel, pictures as attachments and large files (>10mb tar file) to search for possible issues with the developed software. It was decided to use kernel documentation because it includes code examples with many non letter characters which could have been potentially difficult for the IMAP plugin to handle. With this payload several bugs have been found and also fixed. Large files were needed for testing as they are for sure fragmented and are sent in chunks. This test case was designed for the algorithm in the method “handleClientSide()” which was already mentioned in section 3.3. of this document. Pictures were very convenient test subjects as it is visible immediately after the retrieval of the mail whether all parts were downloaded correctly or not. This is shown on the following picture. Note that in order to see the whole picture scrolling would be necessary which was

not an option, because this way it can be shown that a multi-part message was retrieved. Also scrolling within a screenshot is not possible.

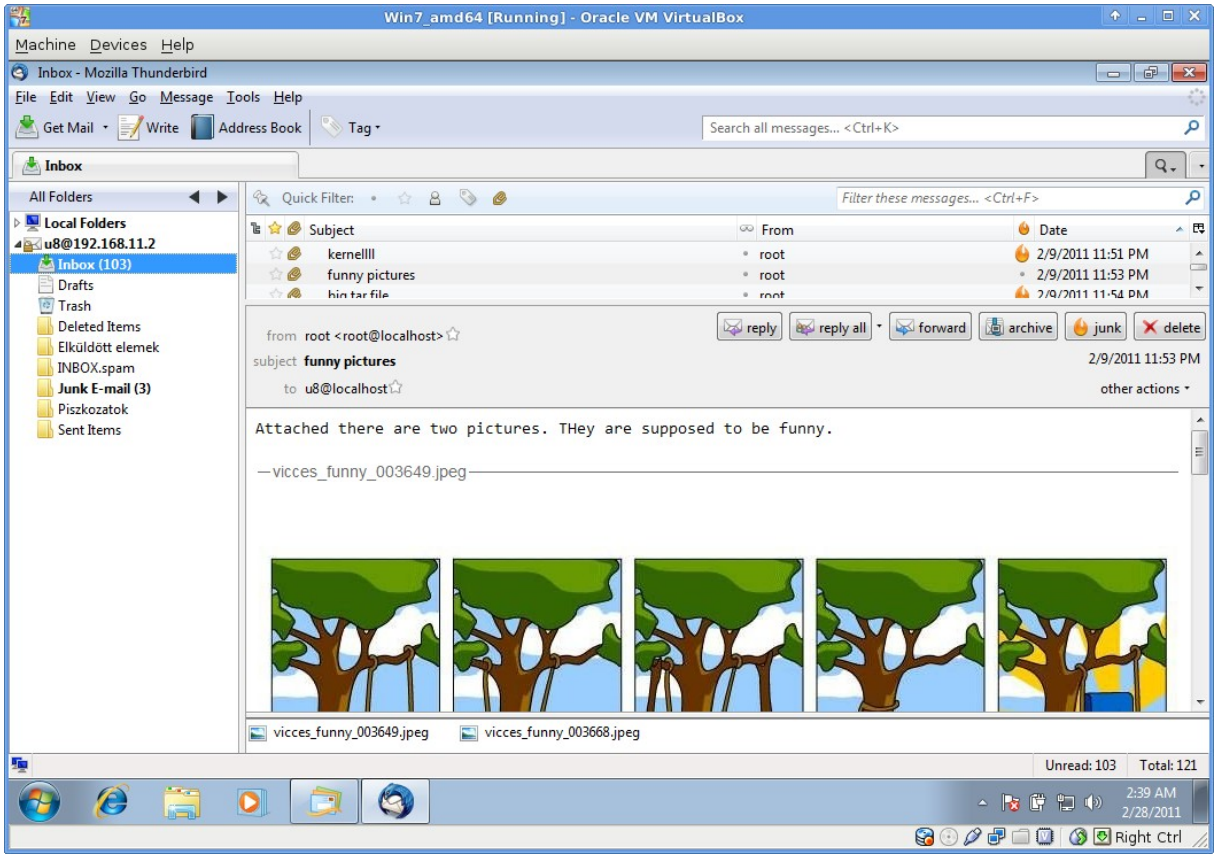


Illustration 28: Thunderbird, attachments

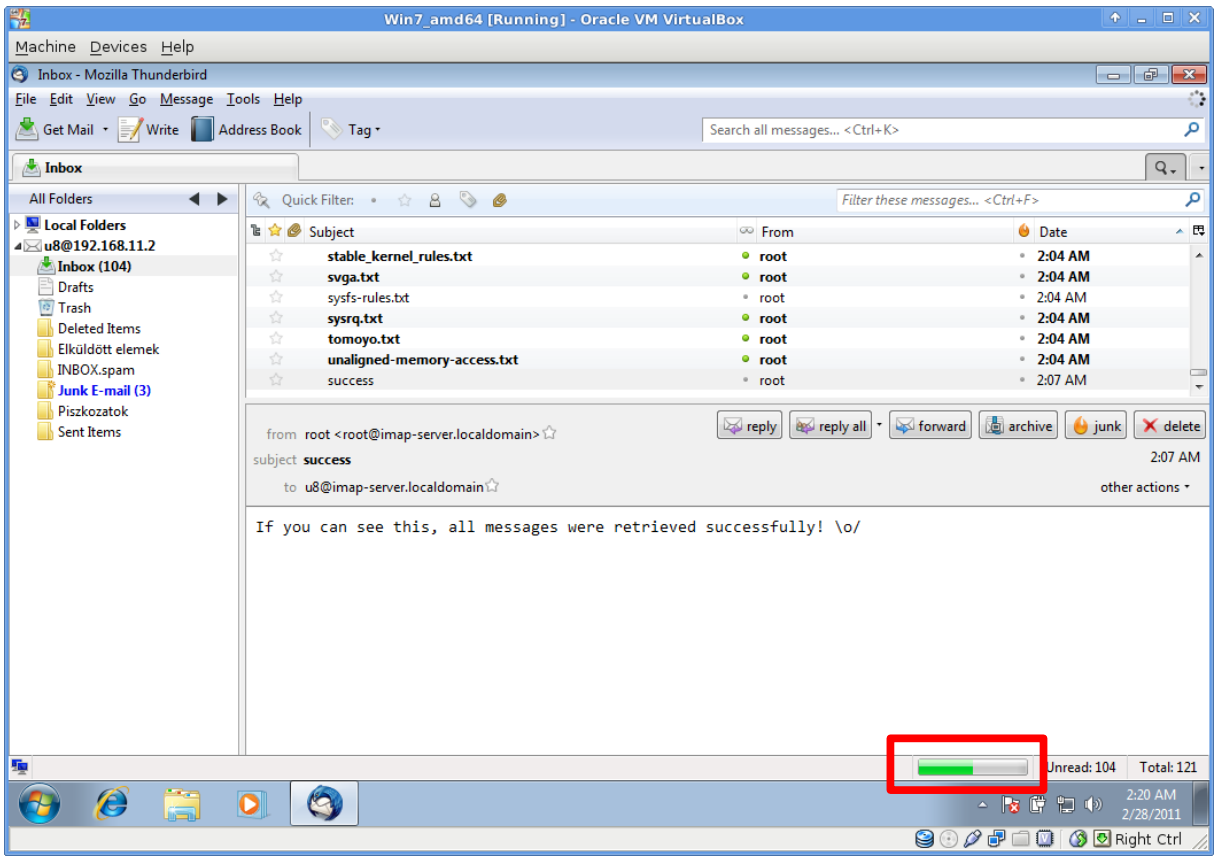


Illustration 29: Thunderbird, success

The picture above shows the retrieval of plain text messages with Mozilla Thunderbird. Note that all headers already had been retrieved, but the status indicator shows only about 50% because the message bodies were still being scanned and downloaded in the background. The bodies of those messages which were clicked on, had been given a higher priority while downloading.

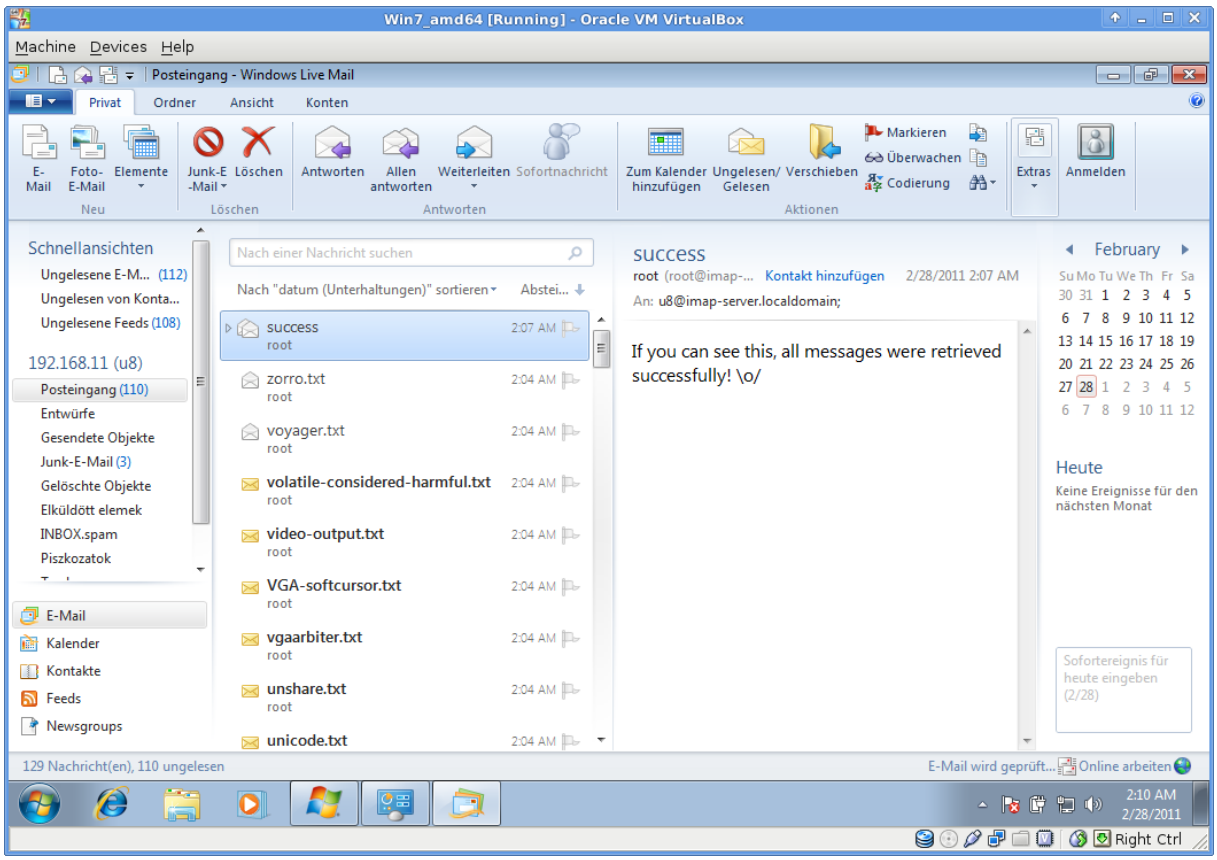


Illustration 30: Windows Live Mail, success

This screenshot indicates that the developed software is also compatible with the other officially supported mail used agent: Windows Live Mail. It is visible that messages can be downloaded successfully.

The window which was captured below contains dumped network traffic from the application Wireshark which was executed on the router on the interface of the IMAP client. It shows that the STARTTLS capability is advertised and that the client also issues this command. After the response “1 OK Beginning TLS negotiation now.” traffic is encrypted and the connection is secure. By sniffing the encrypted network traffic it is not possible to eavesdrop on the unencrypted IMAP traffic.

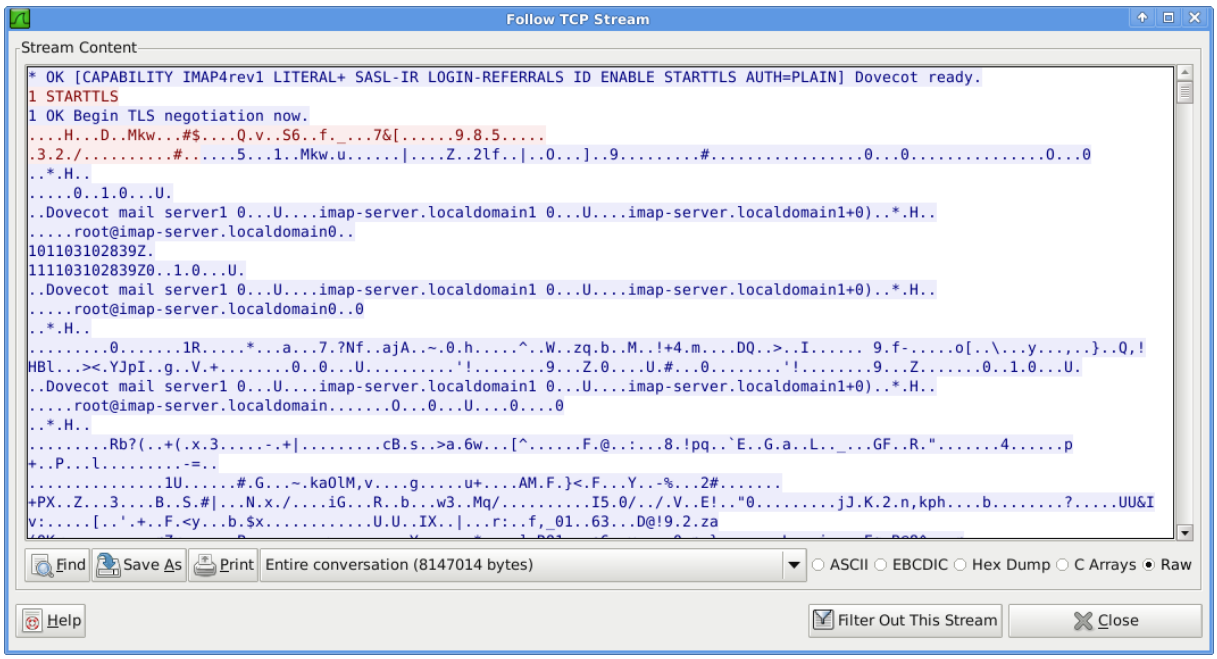


Illustration 31: Wireshark, proof of TLS

4. Summary

Beside all the success and usable results the extent of a master thesis is limited. Therefor there are still topics which could be investigated around IMAP. This is the reason why there has to be a section dedicated to shortcomings right after the next section which summarizes what was achieved in this thesis.

4.1. Results

During the creation of this thesis significant investigation was made into the inner working of the IMAP protocol. It was researched why IMAP is an issue for proxy software and how such proxies can be developed anyway. Available software solutions were reviewed. It was found out, that currently available free software which makes IMAP connections safer does not exist. Shortcomings of RFC 3501 such as the partial fetching feature or the missing timing requirements have been found. It was investigated how these limitations can be overcome by workarounds. For this also underlying layers were inspected as parts of the system. Data throughput and timing were analyzed as critical aspects of communication. After reviewing the timeouts specified by RFCs features such as keep alive bytes were implemented. Further ideas include setting of the PSH bit in TCP headers.

After identifying the attack vectors appropriate measures where created to keep the network safe from threats due to the usage of IMAP. The most important attack vector was malware which could enter the network by being packed into attachments of multipart e-mails. A solution was described how IMAP traffic can be scanned and malware can be blocked.

The introduced measures have been realized by implementing a working, usable IMAP plugin for the existing TLS-Proxy, which fulfills the previously set requirements. This software consists of a virus scanning transparent proxy which filters IMAP traffic and an asynchronous online scanner which can be used to handle unwanted spam messages. Also INIT scripts were written and a Debian package was built for comfortable installation and integration into an existing system. Informative documentation was written within this thesis including descriptions of configuration file options.

Another feature is the safe logging of login credentials without writing them to the harddrive even tough the connection is encrypted. These credentials can be passed on to additional

scanners. Therefor the developed software solution provides a tool for network administrators to handle IMAP appropriately. The result was tested on a variety of systems and these tests were documented. The gist of this thesis is that using the developed IMAP plugin communication over the IMAP protocol can be scanned for viruses or spam thereby adding security to the system. This is a big step towards security for organizations, which use IMAP servers which are outside of their network.

4.2. Problems and shortcomings

As this project was developed as a master thesis resources were limited. Additional ideas exist which could be investigated and implemented. These include moving messages with viruses in them into a separate quarantine folder or appending more detailed virus warning messages. A shortcoming is definitely that spam is not filtered in real time. A solution for this issue has not been found yet. Also the code of the online scanner is has not received too much attention after virus filtering was considered as the main field of investigation. It supports only plain text IMAP and IMAP over SSL using the TCP port 993, but not the STARTTLS feature of IMAP. This is a limitation of Isbg[2] and affects spam scanning only. Virus scanning is not affected by this limitation and works well also with STARTTLS. The topic of extrusion prevention has not been analyzed yet. For that feature a statistical filter for keywords could be used.
Due to the lack of resources, such as time or a realistic test environment for stress testing and reproducing errors during the end of development, currently there are still two known bugs which have not been fixed yet:

Bug No. 1

Before the transfer of actual messages over IMAP, there is a field in the header indicating the RFC822.SIZE attribute of the message. This is important information as it is further processed during the process which determines the end of a message. In case of Microsoft Exchange being used as an IMAP server, this value may not be parsed correctly which renders the input validation of the IMAP plugin useless. In the worst case if the message contains a virus and in the beginning of the message the character sequence “)\r\n\r\n” is present the infected message reaches the IMAP client. To determine the source of this bug it

would be necessary to print debug information about the processed data directly in that part of the source code, where this attribute is extracted.

Bug No. 2

If a large amount of mail is retrieved on one connection (about >200 messages) it may occur that the proxy only caches and does not write data to the client. A probable cause is that the start tag of a message is split into two separate buffer chunks and is not recognized by the algorithm. This bug is not caused by the special structure of a message as the affected message can be downloaded after being given a second try. The cause of this bug may lay also in the the end tag detection of the IMAP proxy, or in non RFC 3511 conform traffic sent by the IMAP server. The real cause of this bug is hard to find as hundreds of messages have to be retrieved to encounter this mentioned issue. Also additional debug output would be necessary which could falsify results in case of a race condition.
Apart from these two bugs no other malfunction is known.

4.3. Ideas for further work

Further work includes fixing the problems and shortcomings above. The spam filtering which has been developed in this thesis is asynchronous. In most cases this is fine, but it would be elegant to offer also a synchronous version. A missing piece of the puzzle is a web interface for administrators which can be used to activate spam protection for IMAP accounts. Another idea would be a quarantine where suspicious messages are moved into via IMAP.

Literature

[1] RFC 3501, page 39, http://www.networksorcery.com/enp/rfc/rfc3501.txt (2011.12.28)

[2] Isbg, http://redmine.ookook.fr/projects/isbg/wiki (2011.12.28)

[3] Trusted Persion in the Middle: TLS-Proxy, Aspetsberger R.

[4] Perdition, Horman S., http://horms.net/projects/perdition/ (2011.12.28)

[5] Vanessa Socket, Horman S., http://hg.vergenet.net/vanessa/vanessa_socket/ (2011.12.28)

[6] RFC 3501, page 12, http://tools.ietf.org/html/rfc3501 (2011.12.28)

[7] ClamAV, http://www.clamav.net/lang/en/ (2011.12.28)

[8] Mutt, http://www.mutt.org/ (2011.12.28)

[9] RFC 822, http://www.ietf.org/rfc/rfc0822.txt (2011.12.28)

[10] Linux 2.6.32 source code: Documentation/networking/tproxy.txt

[11] Untersuchung von verschlüsselter E-Mail Kommunikation nach Spam und Viren, Grundmann M.

[12] Rfc2595, http://www.faqs.org/rfcs/rfc2595.html (2011.12.28)

[13] Lintian, http://lintian.debian.org (2011.12.28)

[14] Litmus, http://litmus.com/resources/email-client-stats (2011.12.28)

[15] Exim 4 default configuration in Debian 6.0: /etc/exim4/exim4.conf.template

[16] Verification and improvement of the sliding window protocol, D. Chkliaev, J. Hooman, E. de Vink, www.cs.ru.nl/ita/publications/papers/hooman/SWP.pdf (2011.12.28)

[17] Ripmime, P. L. Daniels, http://www.pldaniels.com/ripmime/ (2011.12.28)

[18] RFC 2822, Internet Message Format, http://www.ietf.org/rfc/rfc2822.txt (2011.12.28)

[19] http://www.underground8.com/de/products/mf_security_gateway.html (2011.12.28)

[20] Linux kernel documentation: linux/Documentation/networking/tproxy.txt

[21] Oracle VirtualBox, http://www.virtualbox.org/ (2011.12.28)

[22] Fetchmail, http://fetchmail.berlios.de/ (2011.12.28)

[23] Underground_8, http://www.underground8.com/de/ (2011.12.28)

[24] GNU GPL version 2, http://www.gnu.org/licenses/gpl-2.0.html (2011.12.28)

[25] Debian, http://www.debian.org/intro/about (2011.12.28)

[26] Imapfilter, L. Chatzimparmpas, https://github.com/lefcha/imapfilter (2011.12.28)

[27] Juniper IMAP Scanning, http://www.juniper.net/techpubs/software/junos-security/junos-security10.0/junos-security-swconfig-security/jd0e63294.html (2011.12.28)

[28] Linux Programmer's Manual, Signal, http://www.kernel.org/doc/man-pages/online/pages/man7/signal.7.html (2011.12.28)

[29] POSIX, http://standards.ieee.org/develop/wg/POSIX.html (2011.12.28)

[30] RFC1122 p. 81, PSH-flag, http://tools.ietf.org/html/rfc1122 (2011.12.28)

[31] ISO-OSI Model, Application layer, p. 32, http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items (2011.12.28)

[32] RFC 793 p. 32, http://tools.ietf.org/html/rfc793#section-2.8 (2011.12.28)

[33] Zimmermann, Philip (1995). PGP Source Code and Internals. MIT Press. ISBN 0-262-24039-4

[34] National Research Council (U.S.). Committee on the Future of the Global Positioning System; The global positioning system: a shared national asset, ISBN 0-309-05283-1

[35] EICAR test file, http://www.eicar.org/anti_virus_test_file.htm (2011.12.28)

[36] Unix Programmer's Manual, http://cm.bell-labs.com/cm/cs/who/dmr/1stEdman.html (2011.12.28)

[37] TCP/IP Illustrated, Volume 1, Stevens, W. R., p. 282, ISBN 0-201-63346-9

[38] Proceeding of the Seminar Future Internet (FI), DoS, p. 40, http://www.net.in.tum.de/fileadmin/TUM/NET/NET-2009-04-1.pdf#page=40 (2011.12.28)

[39] Courier IMAP server, http://www.courier-mta.org/imap/features.html (2011.12.28)

[40] Iptables, Prerouting http://dev.medozas.de/files/xtables/iptables.html (2011.12.28)

[41] MF series security appliances, http://www.underground8.com/de/products/mf_security_gateway.html (2011.12.28)

[42], Linux Programmer's Manual, Dynamic linking, http://www.kernel.org/doc/man-pages/online/pages/man8/ld-linux.so.8.html (2011.12.28)

[43], “Man in the Midde Attach”, Network Security, Kaufman C., Perlman R., Speciner M., p167, ISBN 0-13-046019-2

[44] “Sqlite 3”, http://www.sqlite.org/docs.html (2011.12.28)

[45] “DPKG – Package manager for Debian”, http://manpages.debian.net/cgi-bin/man.cgi?query=dpkg&apropos=0&sektion=0&manpath=Debian+Sid&format=html&locale=en (2011.12.28)

[46] IEEE standard: Virtual Bridged Local Area Networks, http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf (2011.12.28)

Abbreviations

- Scan connector: layer between the proxy and the spam handling solution
- NAT: Network Address Translation
- DNAT: Destination Network Address Translation
- RFC: Request For Comment
- ISBG: IMAP Spam Begone
- MTA: Mail Transfer Agent
- SMTP: Simple Mail Transfer Protocol
- DoS: Denial of Service
- TCP: Transmission Control Protocol
- CR: Carriage Return
- LF: Line Feed
- FD: File Descriptor
- NTLM: NT LAN Manager
- FQDN: Fully Qualified Domain Name
- MUA: Mail User Agent
- MIME: Multipurpose Internet Mail Extensions
- ASCII: American Standard Code for Information Interchange
- VLAN: Virtual Local Area Network
- DPKG: Debian Packaging system
- MDA Mail Delivery Agent
- GPS Global Positioning System
- TLV Type Length Value

Appendix

Appendix A: Sourcecode of an early attempt:

```
#!/bin/bash


# author: Macskasi Csaba, 2010

# This demo script starts a transparent imap proxy on port 143 (prerouting)
# and a non transparent one on port 144 (output)
# WARNING! All iptables rules are deleted!!!

# how long to wait between scan (seconds). Note, that this value should be
at least a few minutes:
DELAY="15"

# flushing all iptables rules
iptables -t nat -F

# kill running perdition instances
killall perdition.imap4

# using iptables for destination nat
# transparent:
iptables -t nat -A prerouting -p tcp --dport 143 -j DNAT --to 127.0.0.1:143
# non transparent, for local testing only. This is needed to prevent loops.
iptables -t nat -A OUTPUT -p tcp --dport 144 -j DNAT --to 127.0.0.1:143

# starting perdition, log pw of successful logins
perdition.imap4 -c --log_passwd ok

while /bin/true
do
      cat /var/log/syslog | grep Auth | sed 's/ //g' | awk 'BEGIN { FS =
"\""  }; { print $6 "," $2 "," $4}' | sort -u > /tmp/imap_logins
      clear
      echo -e "### Transparent IMAP proxy with spam-scanning capability ###
\n"


      # The logins from /tmp/imap_logons can be used to generate a config
file for isbg
      # now using sniffed login data to call isbg and spamassassin

      for i in `cat /tmp/imap_logins`;
      do
            iHost=`echo $i | awk 'BEGIN { FS = "," }; { print $1 }'`
            iUser=`echo $i | awk 'BEGIN { FS = "," }; { print $2 }'`
            iPasswd=`echo $i | awk 'BEGIN { FS = "," }; { print $3 }'`

            # DEBUG:
            #echo "i: $i"
```

```
            #echo "parsed login data: host $iHost, user $iUser, pw
$iPasswd"

            # scan for spam using isbg, irgnore bad logins/unavailable
hosts...
            ./isbg.py --imaphost $iHost --imapuser $iUser --imappassword
$iPasswd 2>/dev/null
            if [ $? -eq 0 ]
            then
                  echo "$iUser scanned successfully. (`date`)"
            fi
      done;

      # delay...
      sleep $DELAY
done
```

Appendix B: Prepare-network.sh

```
#!/bin/bash

# This script sets up the local network for test purposes.
# Connection to it should be made via virtual computers
# and bridged interfaces to the given vlans.

# --- BEGIN CONFIG AREA ---

VTAG1="11";
VTAG2="12";
VADD1="192.168.11.1";
VADD2="192.168.12.1";
HWNIC="eth1";
REDRPRT="4430";

# --- END CONFIG AREA ---

# enable ipv4 forwarding to play gateway
echo 1 > /proc/sys/net/ipv4/ip_forward
if [ $? -eq 0 ]
then
      echo "forward enabled successfully."
fi

# configure virtual interfaces for VMs
# we'll "route" between these networks:
modprobe 8021q 2>/dev/null;
vconfig add $HWNIC $VTAG1 2>/dev/null
vconfig add $HWNIC $VTAG2 2>/dev/null

# ugly hack to check interface creation...
ifconfig $HWNIC.$VTAG2 1>/dev/null 2>/dev/null
if [ $? -ne 0 ]
then
      echo "ERROR while adding vlan tags to $HWNIC";
fi

# adding ip address to virtual nics
```

```
ifconfig $HWNIC.$VTAG1 $VADD1;
ifconfig $HWNIC.$VTAG2 $VADD2;

if [ $? -eq 0 ]
then
        echo "virtual NICs ($HWNIC.$VTAG1, $HWNIC.$VTAG2) set up
successfully."
fi

# netfilter rule for transparent proxy
# redirect destination is localhost only!
iptables -t nat -F
iptables -t nat -A PREROUTING  -p tcp --dport 143 -j REDIRECT --to-port
$REDRPRT;
if [ $? -eq 0 ]
then
        echo "Netfilter redirect successful."
fi
```

Appendix C: Init script

```
#!/bin/bash
# init script for tlsproxy-imap

# author: Macskasi Csaba

# In order to adapt this script to other protocols of TLSProxy,
# just change the variable $PROTOCOL in the config section.
# After that insert protocol specific commands.

### BEGIN CONFIG ###
BINARY_PATH="/usr/bin/TLSProxy";
CONFIG_PATH="/etc/tlsproxy/imap.conf";
PROTOCOL="IMAPS";
### END CONFIG ###

case $1 in
      start)
            tmp=`ps ax | grep TLSProxy | grep IMAPS`;
            if [ $? -eq 0 ]
            then
                  echo "TLSProxy with $PROTOCOL support is already
running.";
                  exit 255;
            fi

            $BINARY_PATH -P $PROTOCOL -c $CONFIG_PATH &

            tmp=`ps ax | grep TLSProxy | grep IMAPS`;
            if [ $? -eq 0 ]
            then
                  echo "TLSProxy started successfully with $PROTOCOL
support";
            else
```

```
                echo "Error while trying to start TLSProxy";
            fi
            exit 0;;

      stop)
            tmp=`ps ax | grep TLSProxy | grep $PROTOCOL`;
            if [ $? -ne 0 ]
            then
                  echo "TLSProxy with $PROTOCOL support is not running.";
                  exit 255;
            fi;
            pid=`echo $tmp | awk '{ print $1 }'`;
            kill $pid;

            if [ $? -eq 0 ]
            then
                  echo "TLSProxy with $PROTOCOL support was stopped
successfully.";
                  exit 0;
            else
                  kill -9 $pid; # Die already!!
                  if [ $? -ne 0 ]
                  then
                        echo "Error stopping TLSProxy.";
                        exit 255;
                  fi;
            fi;;
      restart)
            $0 stop;
            $0 start;;
      status)
            tmp=`ps ax | grep TLSProxy | grep $PROTOCOL`;
            if [ $? -eq 0 ]
            then
                  echo "TLSProxy with $PROTOCOL support is running.";
            else
                  echo "TLSProxy is NOT running.";
            fi;
            exit 0;;
      *)
            echo "USAGE: $0 (start / stop / restart / status)";
            exit 255;;
esac
```

Europass
curriculum vitae

Personal information

Surname(s) / First name(s)	Macskási Csaba
Address(es)	M.-Hainischstr. 11, 4040 Linz, Austria 1012 Budapest, Attila út 101, Hungary
Telephone(s)	(43-680) 315 25 75
E-mail(s)	macskasi.csaba@gmail.com
Nationality(-ies)	Hungarian
Date of birth	1987
Gender	male

Work experience

Dates	2008 February – 2011 February
Occupation or position held	tutor
Name and address of employer	Pervasive Computing, Johannes Kepler University, Linz Austria

Dates	2010 October – 2011 February
Occupation or position held	Software developer
Name and address of employer	Underground_8 Secure Computing GmbH., Linz Austria

Dates	2011 May – present
Occupation or position held	Software engineer, developer
Name and address of employer	JM-Data GmbH., Linz Austria

Education and training

Dates	2010-present: JKU Linz, Networks and Security

2010: VGTU Vilnius, Erasmus
2006-2011: JKU Linz, IT
2000-2006: highschool graduation

Title of qualification	Bachelor of Science
awarded	
Name and type of organisation providing education and training	JKU, Johannes Kepler University, Linz VGTU, Vilnius Gediminas Technical University, Lithuania Johannes Kepler Gymnasium, Linz

Eidesstattliche Erklärung:

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.