The state of StartTLS

Sean Rijs & Magiel van der Meer

June 1, 2014

Abstract

The state of StartTLS is a project conducted to inventorise the implementations of StartTLS in 116 Dutch organisations and cloud services used by Dutch citizens. For the project a tool was developed to automate the detection of StartTLS characteristics of Simple Mail Transfer Protocol (SMTP) servers belonging to the defined domains. The tool detects the presence of StartTLS, extracts public certificates, enumerates Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocols and cipher suites available and validates found certificates to the chain of trust. Additionally the tool checks for weak certificates due to the Random Number Generator flaw found in Debian and the Heartbleed vulnerability. As a result of the scan with by the tool, a list of 249 unique servers were found. Of these 249 servers, 136 servers support StartTLS. For the domains with StartTLS available on at least one server, 102 unique Mail eXchange (MX) records have been found. Of these 102 MX records 54 MX records present a valid certificate. No servers vulnerable to Heartbleed have been found. One case has been found where two parties share the same certificate. Recommendations to improve the global implementation of StartTLS are to at least present a valid certificate and to proper use Domain Name System (DNS) so shared SMTP services will match the hostname of the MX record.

Contents

1 Introduction

Since the introduction of e-mail in 1982, it has been an important part of the Internet. It is used to communicate between entities, in which an entity can be described as a system or a human end-user. E-mail is as wide-spread and important as the addressing of houses; everybody has an address and can be reached by contacting that address. Over time, the upcoming of Instant Messaging and Social Media has decreased the use of personal e-mail systems. Yet e-mail remains important for official communications between governments and their citizens, companies and their customers and between (employers of) companies. Again, simply due to the fact that everybody has an e-email address and can be reached using that address.

The conceptual working is quite easy to comprehend. In a simplified schematic there are four entities to be defined: the sender, the sending e-mail server, the receiving e-mail server and the receiver.

All these entities might be located on the same system or spread over up multiple systems. When a sender sends an e-mail, the sending e-mail server does a Domain Name System (DNS) lookup for the Mail eXchange (MX) records of the receiving mailserver based on the domain-part of the receiving e-mail address. The host name returned is resolved to an Internet Protocol (IP) address and a Simple Mail Transfer Protocol (SMTP) connection to this server is initiated. The receiving mail server has several options to accept this, which will be further described in section 7. The receiving e-mail server generally places the incoming e-mail in the receivers INBOX.

In the first specification of SMTP no authentication, authorisation, encryption or authenticity checks were in place. These were later added in RFCs [13] [10]. But not all e-mail servers on the Internet are using the most optimal settings. While it is quite common the connection between the sender or receiver and the sending or receiving e-mail server is authenticated and encrypted, communications between mail servers are often unencrypted and no authenticity checks are in place.

This projects goal is to define a minimum default an e-mail server should support to ensure authentication, authorisation, encryption and authenticity when communicating with other mail servers and develop a framework which tests a given mail server based on the defined minimum default.

2 Research

The goal of this project is to scan a set of domains to look at the state of the **StartTLS** extension for SMTP defined in RFC 2487 [11]. The domain set is taken from different type of sources such as current research [7], known Dutch Internet Service Provider (ISP)s, banks and e-mail providers. The set of domains does not represent the whole Internet, or all Dutch organisations. However it can give a indication of the current state of StartTLS implementations.

For this project a Proof of Concept (PoC) is developed that can test the StartTLS capabilities of the a Mail Transfer Agent (MTA). The output of the test framework should be able to show if e-mails sent from a domain to an other domain are protected from eavesdropping.

2.1 Approach

The research follows the following steps.

- Define requirements for SMTP communication.
- Define a list of configuration errors or vulnerabilities to match against the requirements. For example: the Debian RNG flaw, OpenSSL Heartbleed, self-signed keys, duplicate keys, hostname mismatching, expired certificates, improper configured (Start)TLS, unverifiable chain of trust, untrusted signing authorities in use and the presence of StartTLS altogether.
- Develop a PoC to test these requirements based on the list of requirements and tests. The PoC will be a framework written in Python which takes a domain name as input, resolves the MX records and adds the IP(v6) addresses found to the scan engine.
- Scan the set of e-mail providers given the domain name.
- Draw conclusions from results.

Note that seen from the test environment its perspective, the tests will only check outgoing and not incoming e-mail, as this would require user access at the target MTA.

2.2 Scope

The research focuses on inter-MTA server communication, specifically StartTLS. The DNS-Based Authentication of Named Entities (DANE) is excluded in this research. Also end-to-end e-mail traffic is out of scope, as technologies as Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME) are designed for that.

2.3 Related Work

There are a lot of Transport Layer Security (TLS) assessment tools [2] that test TLS for web services. However these assessment tools do not provide SMTP TLS assessment.

A web service called CheckTLS [1] can assess an MTA StartTLS configurations on demand, however it only shows results per domain and it is not possible to compare results of multiple domains. CheckTLS also does not differentiate in TLS protocol versions, or checks for the ciphers in use or checks the Heartbleed vulnerability.

The online social networking service Facebook did an assessment of its users MTA StartTLS availability and configuration. [4] However government and corporate e-mail systems were not represented according to the authors. Therefor it does not represent a general view of e-mail MTAs, nor does it shows interesting details about the successful TLS connections. (e.g. keysize used, Heartbleed, Debian RNG and etc.)

2.4 Target groups

The target hosts can be categorised in the following categories. The entire list of uncategorised domains that were tested can be found in appendix A.

- Major Dutch banks
- Some Dutch government sites
- Dutch ISPs
- Top 10 Dutch universities
- Top 10 news papers
- AEX listed companies
- List of domains provided by Cosmin Dumitru

The tests were not conducted based on the categories thus the results cannot be shown separately per category. This decision was made due to the limit amount of time available for this project.

2.5 Expected results

Expected is that StartTLS adoption would be low. Only Gmail, domains hosted by Google and private hosted domains are expected to have proper StartTLS configured. This expectation is based on the fact that StartTLS is not enabled by default in most MTA's, knowledge is needed to install a proper certificate and this cannot be automated in a reliable and secure manner.

3 Work done

As no clear solution appeared to be available for scanning StartTLS enabled hosts and gathering the data needed for this research, the decision was made to write a Python program. This program should be capable of resolving DNS records, initiating SMTP sessions, detecting and if applicable, starting StartTLS session. From a StarTLS session the following information should be extracted:

- Available SSL/TLS protocols

  TLS 1.0 - 1.2

  SSLv3

- Available cipher suite
- Public certificate

  Moduli

  Certificate bit size

  Hostname

  Susceptible to the Debian Random Number Generator

- Certificate chain validation
- Heartbleed vulnerable

Subsequently the program also gathers the Pointer Record (PTR) and Sender Policy Framework (SPF) records for the domain to get a more clear view of the scanned domain. DomainKeys Identified Mail (DKIM) is not tested since the DNS record containing the DKIM record has a so called 'selector' property which is unknown unless one receives an e-mail containing DKIM headers and this 'selector'.

3.1 SSL/TLS protocols

The protocol used by a Secure Sockets Layer (SSL) or TLS session is negotiated by the server and the client. The highest common supported protocol is usually chosen. By initiating and destroying sessions with only one supported protocol (as the client) one can detect all the protocols supported by the server. The protocols tested are SSLv3, TLSv1.0, TLSv1.1 and TLSv1.2. The Python library 'ssl' also has a compatiblity version called SSLv23 which supports SSLv2, SSLv3 and TLSv1. But since this library doesn't support SSLv2 any more, effectively only SSLv3 and TLSv1 are used in this mode. The 'ssl' library recommends using the SSLv23 protocol for maximum compatibility. But SSLv3 and TLSv1 are tested seperatly so SSLv23 is not used by the program.

This requires the client to start at least 4 connections with the SMTP server to detect all available protocols. These connections are started sequentially.

3.2 Ciphers

The cipher used in an encrypted session is negotiated in the same way as the underlying protocol. The client sends a list and the server usually picks the highest common supported cipher. For this

test only the ciphers supported by OpenSSL on the test-system are tested. Usable ciphers can be categorised by protocol. Only the ciphers applicable to a given protocol should be tested on that protocol. This would reduce the amount of connections which are initiated to a server. Due to the nature of SSL and TLS it is not possible to enumerate all the available server ciphers within the same connection. This means that also for each cipher tested a new SMTP session needs to be initialised.

In the current version of the developed application only the clients top 30 available ciphers are tested for server-side support, regardless of the protocol support. This is to lower the amount of servers blocking consecutive requests. The concept outlined in the paragraph above would have been too time consuming to implement error-free in the application developed. The list of ciphers tested is found in appendix B.

3.3 Certificate & chain

The certificate is extracted from the first connection which is initiated to the server. The chain is validated by the built-in 'ssl' library of Python. This function unfortunately returns 'None' if the certificate is valid and raises a Python exception when invalid but gives no clear reason why the certificate is not valid. For instance, the library raises a distinct exception for unmatching hostnames but does not distinct between an invalid Certificate Authority (CA) and a self signed certificated. Given the working of a Public Key Infrastructure (PKI) this actually makes sense. A client-side certificate validator cannot see if a certificate is signed by an unknown authority or not signed at all.

3.3.1 Moduli

The certificate is imported in X.509 format and further parsed to extract the moduli of the public key. The moduli could be analysed for weak numbers. Due to time constraints, this research could not properly research such weaknesses. Additionally only the Rivest Shamir Adleman (RSA) was supported in our scan, and the DSA and ECDSA are omitted. Therefore it was decided not to use this information, as it would only represent RSA keys.

3.3.2 Bit size

The bit size of the public key is extracted to asses what bit sizes are used. As it would be a weakness to use for example 1024 bit size keys [8].

3.3.3 Hostname

The hostname is extracted from the certificate its Common Name (CN) and Subject Alternative Name (SAN) field. This is not further used in the code, the certificate chain is validated otherwise, but might be interesting for future use.

3.3.4 Debian RNG

In the past a vulnerability on Debian distributions which generated weak keys [5]. OpenSSL released the tool openssl-blacklist that checks public keys against a database containing moduli generated by this vulnerability. This research also checks if they are vulnerable using the openssl-blacklist tool.

3.4 Heartbleed

Quite recently the OpenSSL vulnerability named Heartbleed was published. [6] This can be detected sending heartbleed request after the TLS handshake is completed. A tool was developed [12] in which these checks are made. This research uses this tool to check if targets are still vulnerable to the Heartbleed attack.

4 Test environment

The host performing the scans needs to be configured to resemble a valid SMTP host under a valid DNS domain. The reason for this is to prevent being marked as a malicious host. As this could imply our test environment to be blacklisted on lists such as *The Spamhaus Block List* [3], consequently making it unable to scan any hosts that actively apply these blacklists.

Additionally the following extra steps are taken to prevent blacklisting:

- Run a SMTP daemon running which listens on the expected default Transmission Control Protocol (TCP) port 25
- Configure the PTR to the testing host name
- Include valid DKIM records the test domain
- Include valid SPF records in the test domain

During this project the testing domain name is **gotls.info** and the Fully Qualified Domain Name (FQDN) is **smtp.gotls.info**. To further increase trust of a legitimate SMTP server, an informational web server runs on the same host serving a web page. The web page informs visitors that this is used for research purposes by showing a readable version of our initial research proposal. The web page is running on `https://smtp.gotls.info/`.

5 Results

5.1 StartTLS usage

A total of 116 domains are listed in appendix A. Resolving MX and A(AAA) records for these domains results in a total of 598 servers of which 242 unique servers. This huge difference is caused by shared infrastructure between domains, domains using the same set of A(AAA) records under multiple MX records (i.e. mx1.domain.com and mx2.domain.com both resolving to 123.123.123.123) and domains pointing their MX records to e-mail providers as Google or Yahoo.

The results are based on the unique amount of servers. Yet all the servers are tested per domain to get the validity of the certificate based on the domain. A server can have a valid certificate for domain A yet presenting this same certificate for domain B, which makes it invalid for domain B.

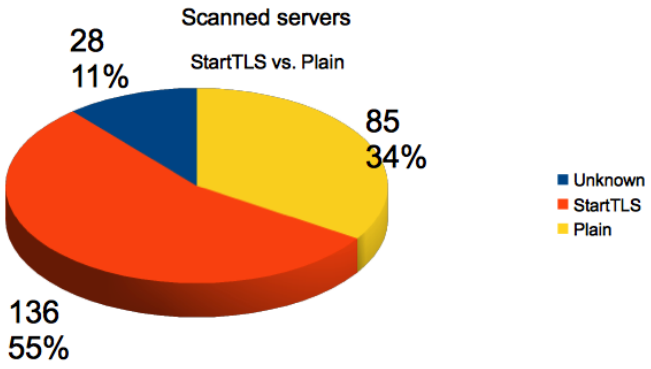

Figure 1: StartTLS versus plain

The number of servers supported StartTLS is shown in figure 1. Of the tested domains 55% supports StartTLS. 34% does not support any kind of StartTLS over SMTP. For a 11% the state of StartTLS on the remote server could not be detected. This can be caused by the remote server rate limiting the requests, supporting only an untested SSL/TLS protocol like SSLv2 or using Digital Signature Algorithm (DSA)/Elliptic Curve Digital Signature Algorithm (ECDSA) certificates.[1]

The expectation was that only Google would be supporting StartTLS in a proper way. To show that the support of StartTLS is actually more spread out over the tested domains, figure 2 shows the domains with 6 servers or more supporting StartTLS. This figure shows that it is not only the big domains supporting StartTLS but that the spreading is equally divided between the tested domains.

[1] These numbers slightly differ from what was shown in the presentation due to a bug in the calculations used. The numbers mentioned here are correct.

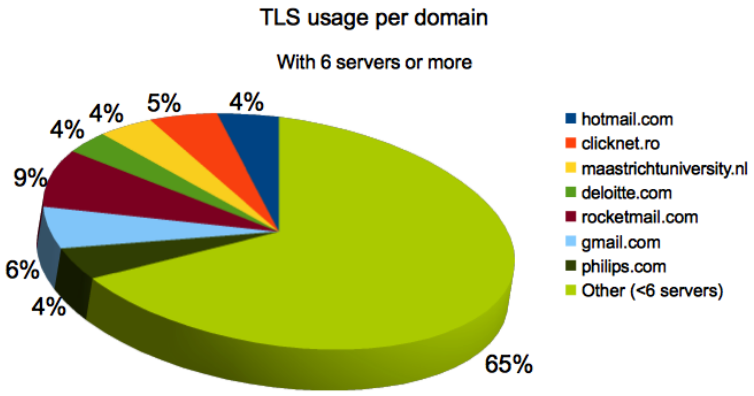

Figure 2: StartTLS usage per domain

5.2 Protocol & Cipher support

A server which supports StartTLS can support multiple protocols. In figure 3 is shown how much servers support what protocol. Any given server can support multiple protocols so the total does not add up to 100%. For SSLv3 127 servers of the total of 136 servers supporting StartTLS are found. For TLSv1 this is 125 and TLSv1.1 and 1.2 this is both 30.

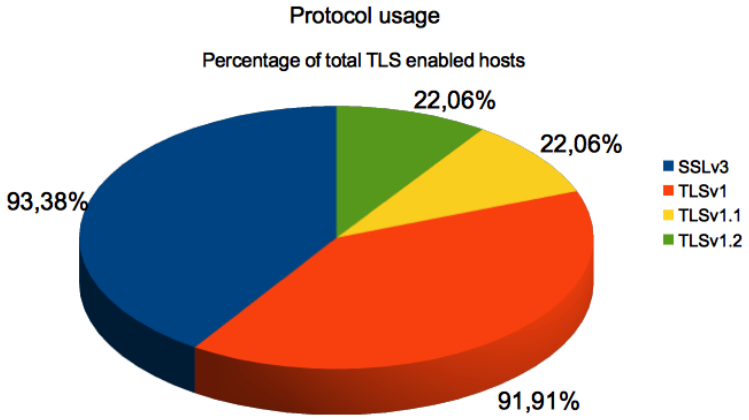

Figure 3: Supported protocols

The same goes for the cipher usage. A total of six ciphers has been found. This low diversity is caused by the approach of testing this as described in section 3.2. Because only the top 30 of the test environments ciphers is tested, the diversity of supported ciphers is quite low. The top 30 contains the more stronger ciphers and apparently this is not widely supported.

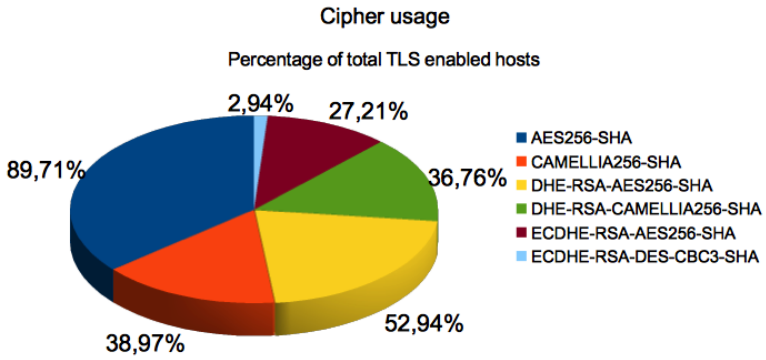

Figure 4: Supported ciphers

5.3 Keysizes

For all servers supporting StartTLS the keysize of the certificate was extracted from the certificate. 86% of the servers is using a 2048 bits key, while 7% is still using 1024 bits keys. Only 3% is using 4096 bits keys and 4% is unknown. This is likely caused by rate limiting or the server presenting a broken certificate while it supports StartTLS.

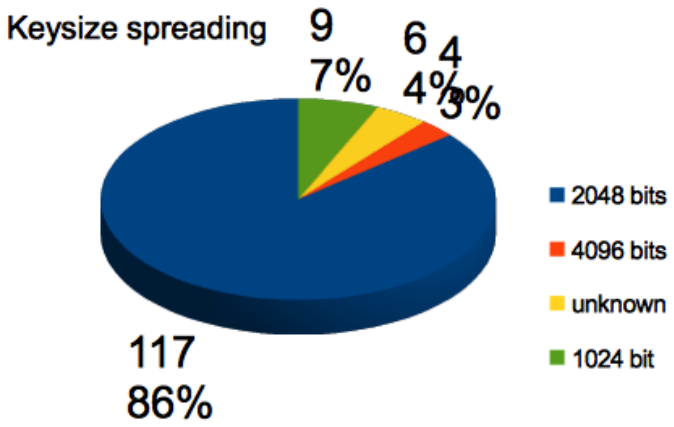

Figure 5: Keysize spreading

5.4 Certificates

The CN and SAN properties of a certificates match with a hostname and not with an IP address. Therefore, the numbers here have no obvious connection with the amount of servers supporting

StartTLS. The 136 servers supporting StartTLS are providing SMTP services for 70 different domains which resolve to 102 unique MX records.

Not all domains have StartTLS deployed consistently, so of these 102 MX records, 54 have a certificate which validates against the chain, has a CN or SAN which matches the MX record and is valid in time. 38 MX records present an invalid certifacte. Mostly this is because the hostname does not match the certificates CN or SAN properties. For example, Yahoo has a wildcard certificate deployed over all their servers but a wildcard certificate does not allow for nesting. So 'mta5.am0.yahoodns.net.' does not match the CN '*.yahoodns.net'.

5.4.1 Duplicate certificates

Another interesting result is duplicate certificates in use between different domains. This can be caused by multiple reasons, like separate companies having their SMTP outsourced to the same supplier or because they use the same default certificate shipped with their SMTP server software.

Looping over all the certificates and appending unique domains to a list per unique certificate gets the following list (omitting the results with only one domain).

```
19de957e ['trouw.nl', 'volkskrant.nl', 'parool.nl', 'ad.nl']
348078ce ['uu.nl', 'uva.nl', 'rug.nl', 'leidenuniv.nl', 'ru.nl', 'tue.nl', 'vu.nl',
    'utwente.nl', 'leiden.edu', 'maastrichtuniversity.nl']
5a4d8c52 ['live.com', 'msn.com', 'outlook.com', 'hotmail.com']
5e52fdc2 ['clicknet.ro', 'gmail.com']
6700c9ab ['telegraaf.nl', 'dsm.com', 'aegon.nl']
7d27a9c0 ['arcelormittal.com', 'fugro.com']
9c23d58d ['asml.com', 'shell.com', 'wageningenur.nl', 'boskalis.com']
b0d61365 ['yahoo.es', 'yahoo.de', 'ymail.com', 'yahoo.ro', 'yahoo.it',
    'yahoo.co.uk', 'yahoo.ca', 'rocketmail.com', 'yahoo.fr', 'yahoo.com']
c8e86c96 ['unilever.com', 'elsevier.nl']
e69dbb96 ['randstad.nl', 'ahold.nl']
ead672ee ['snsbank.nl', 'asnbank.nl']
```

One can immediately see several related domains sharing infrastructure. The news papers are part of the same parent company, the universities host their SMTP services at Surfnet, Microsoft shares servers for related services, clicknet.ro is hosted at Gmail, Yahoo uses a lot of domains (and hosts rocketmail.com) and the SNS and ASN bank are also part of the same parent company.

What is left is a list of companies not directly related to each other.

```
6700c9ab ['telegraaf.nl', 'dsm.com', 'aegon.nl']
7d27a9c0 ['arcelormittal.com', 'fugro.com']
9c23d58d ['asml.com', 'shell.com', 'wageningenur.nl', 'boskalis.com']
c8e86c96 ['unilever.com', 'elsevier.nl']
e69dbb96 ['randstad.nl', 'ahold.nl']
```

For the first entry, these companies host their SMTP services at Outlook.com:

```
$ for i in $(
        dig +short mx telegraaf.nl | awk '{print $2}';
        dig +short mx dsm.com | awk '{print $2}';
        dig +short mx aegon.nl| awk '{print $2}'
        ); do
    echo $i; dig A +short $i;
done
telegraaf-nl.mail.eo.outlook.com.
```

```
213.199.154.87
213.199.154.23
dsm-com.mail.protection.outlook.com.
213.199.154.23
213.199.154.87
aegon-nl.mail.protection.outlook.com.
213.199.154.87
213.199.154.23
```

The same goes for asml.com, shell.com, wageningenur.nl and boskalis.com. These domains have their SMTP hosted at Frontbridge.com. Elsevier.nl and Unilever.com host their e-mail at MXLogic.net and Ahold.com and Randstad.nl have their SMTP hosted by Psmtp.com. So far so good.

However there are two domains that resolve to different IP addresses, yet they present the same certificate. After extracting the certificates from the data repository, verifying them with OpenSSL gave the result shown below. The domains, defined as companyA.tld and company.B for responsible disclosure purposes, use a 1024 bit self-signed Cisco Appliance Demo certificate:

```
$ openssl verify companyA.tld
companyA.tld: C = US, ST = California, L = San Jose, O = "Cisco Systems, Inc",
    CN = Cisco Appliance Demo Certificate
error 18 at 0 depth lookup:self signed certificate
OK
$ openssl verify companyB.tld
companyB.tld: C = US, ST = California, L = San Jose, O = "Cisco Systems, Inc",
    CN = Cisco Appliance Demo Certificate
error 18 at 0 depth lookup:self signed certificate
OK
```

6 Lessons learnt

During our scans the testing environment often got rate limited and even blacklisted, effectively breaking our scan and making the results inconsistent. It is unclear what the reason is as our sessions will get disconnected. It could be that it does not allow multiple session simultaneously which is caused by our threaded approach of our PoC, or it could have been that the scanner got marked as dangerous as it did not send any e-mail in the SMTP sessions.

To mitigate this threading can be disabled when scanning an SMTP host. However due to time constraints a rewrite of the code was not possible. Additionally the scans would have taken up to hours instead of minutes, which would have cost even more time.

7 Ideal world

7.1 Current situation

In the current situation, the use of StartTLS is very opportunistic. A sending e-mail server can optionally initiate the StartTLS command when it receives a $250-STARTTLS$' extension. What happens next is pure overhead; it validates the received certificate and encrypts the connection. But independent of the certificates validity, it always continues with StartTLS over SMTP. In this situation, Eve the eavesdropper can still intervene on the connection by presenting an another invalid certificate because the sending host will accept any invalid certificate. Eve now has succesfully performand an Man-in-the-Middle (MitM) attack. The encryption added to the connection by StartTLS is pure obfuscation but adds no real security.

7.2 Ideal situation

In the ideal situation, every server, sending and receiving, must present a valid certificate which matches the server its hostname. To exchange e-mail between e-mail servers in a truly secure manner, both sides should adhere to the rules described in sections 7.2 and 7.2. The terminology is based on Request for Comment (RFC) [9].

With these rules implemented, SPF and DKIM would not be necessary anymore. The PKI infrastructure used by StartTLS authenticates the sending servers identity and eliminates the use of SPF. The encrypted SSL/TLS tunnel ensures data authenticity and effectively obsoletes DKIM. Since the proper use of certificates most likely will involve financial costs, it would be less attractive for spam sending groups to continue their operations. These costs will also

Sending e-mail server

- Must only send e-mail when receiving server supports StartTLS
- Must require a valid certificate

  Either via DANE (requires Domain Name System Security Extensions (DNSSec))

  Or via the traditional certificate chain
- Must force the receiving server to validate the sending e-mail server
- Should force the use of Perfect Forwarding Secrecy (PFS)
- The PTR record of the IP address connecting to must match the hostname
- Must refrain from sending e-mail when one of the above musts fails

Receiving e-mail server

- Must only receive e-mail when sending server initiates StartTLS

- Must validate the sending e-mail server certificate based on the PTR record

  Either via DANE (requires DNSSec)

  Or via the traditional certificate chain

- Should force the use of PFS
- Must refrain from receiving e-mail when one of the above musts fails

7.3 Compromise

Unfortunately, this ideal situation is unprocurable to become implemented world wide. The biggest and inevitable disadvantage is that every SMTP server on the Internet should adhere to these rules. This will probably never happen. A compromise could be to mitigate the defined rules. In this situation, a sending SMTP server should at least try to initiate a StartTLS session over SMTP when StartTLS is reported available by the receiving SMTP server. The receiving e-mail server should at least try to validate the client certificate, but PTR records are often not configured correctly to allow for this.

7.4 Recommendations

Some recommendations to improve e-mail handling for SMTP hosting parties are:

- Enable StartTLS on all servers
- Point multiple domains to the same MX records

  I.e. MX records for example.com point to example.net

  I.e. MX records for example.org point to example.net

- Present a valid certificate on example.net
- Configure PTR records for the IP addresses hosting SMTP
- At least configure opportunistic StartTLS

8 Conclusion

After conducting this research, one can conclude the state of StartTLS is better than expected beforehand but still needs a lot of work. There are still a lot of SMTP servers badly configured. For the 102 unique MX records scanned, 54 validated and 38 failed. The remainder has no StartTLS on all MX records available.

Recommendations for the community are to solve the mismatching hostnames. This is the most common reason why most validations fail. A sending SMTP host should at least try to do opportunistic encryption and validate the remote certificate.

Note that end-users never really know if their e-mail is sent over TLS beforehand. They must trust their MTA to have TLS implemented in a strict configuration as described in section 7. However doing this implies not being able to exchange e-mails with the bigger part of the Internet. An alternative would be to implement TLS in a opportunistic configuration as described in section 7.3.

In the table a summary is shown of all the results gathered from the research.

(a)

Servers	
Servers scanned	249
Supporting StartTLS	136
No StartTLS available	85
Unknown StartTLS	28
Domains	
Unique domains scanned	116
Domains with StartTLS	70
Unique MX records	102
Validating certificates	54
Failing certificates	38
Unintended duplicate keys	1
Vulnerabilities	
Heartbleed	0
Debian Random Number Flaw	0

(b)

Keysizes	
Unknown	6
1024 bit	9
2048 bit	117
4096 bit	4
Protocols	
SSLv3	127
TLSv1	125
TLSv1.1	30
TLSv1.2	30
Cipher suites	
AES256-SHA	122
CAMELLIA256-SHA	53
DHE-RSA-AES256-SHA	72
DHE-RSA-CAMELLIA256-SHA	50
ECDHE-RSA-AES256-SHA	37
ECDHE-RSA-DES-CBC3-SHA	4

Table 1: Summary

Future research

Future research can be to extend this research including DANE and PFS.

Acronyms

CA Certificate Authority

CN Common Name

DANE The DNS-Based Authentication of Named Entities

DKIM DomainKeys Identified Mail

DNS Domain Name System

DNSSec Domain Name System Security Extensions

DSA Digital Signature Algorithm

ECDSA Elliptic Curve Digital Signature Algorithm

FQDN Fully Qualified Domain Name

IP Internet Protocol

ISP Internet Service Provider

MitM Man-in-the-Middle

MTA Mail Transfer Agent

MX Mail eXchange

PFS Perfect Forwarding Secrecy

PGP Pretty Good Privacy

PKI Public Key Infrastructure

PoC Proof of Concept

PTR Pointer Record

RFC Request for Comment

RSA Rivest Shamir Adleman

SAN Subject Alternative Name

S/MIME Secure/Multipurpose Internet Mail Extensions

SMTP Simple Mail Transfer Protocol

SPF Sender Policy Framework

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

References

[1] CheckTLS Service. https://www.checktls.com/.

[2] Qualys SSL Labs. https://www.ssllabs.com.

[3] Spamhaus. http://www.spamhaus.org.

[4] The Current State of SMTP STARTTLS Deployment. https://www.facebook.com/notes/protect-the-graph/the-current-state-of-smtp-starttls-deployment/1453015901605223.

[5] CVE-2008-0166, May 2008. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166.

[6] CVE-2014-0160, April 2014. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[7] Fahimeh Alizadeh and Razvan C. Oprea. Discovery and Mapping of the Dutch National Critical IP Infrastructure. Master's thesis, August 2013.

[8] Elaine Barker and Allen Roginsky. NIST Special Publication 800-131A, January 2011. http://csrc.nist.gov/publications/nistpubs/800-131A/sp800-131A.pdf.

[9] S. Bradner. RFC 2119: Key words for use in RFCs to Indicate Requirement Levels, 1997. http://www.ietf.org/rfc/rfc2119.

[10] M. Delany. RFC 4870: Domain-Based Email Authenticationi (DKIM), 2007. http://www.ietf.org/rfc/rfc4408.

[11] P. Hoffman. RFC 2487: SMTP Service Extension for Secure SMTP over TLS, 1999. http://www.ietf.org/rfc/rfc2487.

[12] Jared Stafford. Tool to check for Heartbleed. https://gist.github.com/dyatlov/10192468.

[13] M. Wong and W. Schlitt. RFC 4408: Sender Policy Framework (SPF), 2006. http://www.ietf.org/rfc/rfc4408.

Appendices

A List of domains

- kpnmail.com
- telfort.nl
- upc.nl
- ziggo.nl
- tele2.nl
- scarlet.nl
- caiway.nl
- vodafone.nl
- xs4all.nl
- T-mobile.nl
- abnamro.nl
- asnbank.nl
- bankofscotland.nl
- binck.com
- dbn.nl
- ing.nl
- triodos.nl
- snsbank.nl
- vanlanschot.nl
- rabobank.nl
- equens.com
- gmail.com
- hotmail.com
- outlook.com
- me.com
- yahoo.com
- mail.ru
- icloud.com
- uva.nl
- rug.nl
- vu.nl
- leidenuniv.nl
- leiden.edu
- maastrichtuniversity.nl
- ru.nl
- eur.nl
- tilburguniversity.edu
- uu.nl
- tudelft.nl
- tue.nl
- wageningenur.nl
- utwente.nl
- deloitte.com
- deloitte.nl
- tomtom.nl
- ams-ix.net
- aex.nl
- ahold.nl
- aegon.nl
- klm.com
- akzonobel.com
- arcelormittal.com
- asml.com
- boskalis.com
- corio.com
- dsm.com
- fugro.com
- heineken.com
- philips.com
- postnl.nl
- randstad.nl
- elsevier.nl
- shell.com
- sbmoffshore.com
- tnt.com
- unilever.com
- wolterskluwer.com
- nrc.nl
- ad.nl
- volkskrant.nl
- parool.nl
- telegraaf.nl
- trouw.nl
- fd.nl
- nu.nl
- tweakers.net
- campusdiemenzuid.nl
- transip.nl
- protonmail.ch
- greenhost.nl
- xnet.ro

- me.com
- msn.com
- clicknet.ro
- mail.com
- home.ro
- web.de
- email.ro
- thebest4ever.com
- personal.ro
- yahoo.es
- mailxtr.eu
- yahoo.fr
- freemail.hu
- yahoo.it
- yahoo.ca
- yahoo.de
- live.com
- rdslink.ro
- k.ro
- rocketmail.com
- yahoo.ro
- ymail.com
- yahoo.co.uk
- overheid.nl
- belastingdienst.nl
- aivd.nl
- mivd.nl
- sidn.nl
- cwi.nl
- rijksoverheid.nl
- e-overheid.nl
- hackdeoverheid.nl
- e-overheid
- digid.nl
- svb.nl

B List of tested ciphers

- ECDHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AESGCM(256) Mac=AEAD
- ECDHE-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AESGCM(256) Mac=AEAD
- ECDHE-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA384
- ECDHE-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA384
- ECDHE-RSA-AES256-SHA SSLv3 Kx=ECDH Au=RSA Enc=AES(256) Mac=SHA1
- ECDHE-ECDSA-AES256-SHA SSLv3 Kx=ECDH Au=ECDSA Enc=AES(256) Mac=SHA1
- SRP-DSS-AES-256-CBC-SHA SSLv3 Kx=SRP Au=DSS Enc=AES(256) Mac=SHA1
- SRP-RSA-AES-256-CBC-SHA SSLv3 Kx=SRP Au=RSA Enc=AES(256) Mac=SHA1
- DHE-DSS-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=DSS Enc=AESGCM(256) Mac=AEAD
- DHE-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=DH Au=RSA Enc=AESGCM(256) Mac=AEAD
- DHE-RSA-AES256-SHA256 TLSv1.2 Kx=DH Au=RSA Enc=AES(256) Mac=SHA256
- DHE-DSS-AES256-SHA256 TLSv1.2 Kx=DH Au=DSS Enc=AES(256) Mac=SHA256
- DHE-RSA-AES256-SHA SSLv3 Kx=DH Au=RSA Enc=AES(256) Mac=SHA1
- DHE-DSS-AES256-SHA SSLv3 Kx=DH Au=DSS Enc=AES(256) Mac=SHA1
- DHE-RSA-CAMELLIA256-SHA SSLv3 Kx=DH Au=RSA Enc=Camellia(256) Mac=SHA1

- DHE-DSS-CAMELLIA256-SHA SSLv3 Kx=DH Au=DSS Enc=Camellia(256) Mac=SHA1
- ECDH-RSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
- ECDH-ECDSA-AES256-GCM-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AESGCM(256) Mac=AEAD
- ECDH-RSA-AES256-SHA384 TLSv1.2 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA384
- ECDH-ECDSA-AES256-SHA384 TLSv1.2 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA384
- ECDH-RSA-AES256-SHA SSLv3 Kx=ECDH/RSA Au=ECDH Enc=AES(256) Mac=SHA1
- ECDH-ECDSA-AES256-SHA SSLv3 Kx=ECDH/ECDSA Au=ECDH Enc=AES(256) Mac=SHA1
- AES256-GCM-SHA384 TLSv1.2 Kx=RSA Au=RSA Enc=AESGCM(256) Mac=AEAD
- AES256-SHA256 TLSv1.2 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA256
- AES256-SHA SSLv3 Kx=RSA Au=RSA Enc=AES(256) Mac=SHA1
- CAMELLIA256-SHA SSLv3 Kx=RSA Au=RSA Enc=Camellia(256) Mac=SHA1
- PSK-AES256-CBC-SHA SSLv3 Kx=PSK Au=PSK Enc=AES(256) Mac=SHA1
- ECDHE-RSA-DES-CBC3-SHA SSLv3 Kx=ECDH Au=RSA Enc=3DES(168) Mac=SHA1
- ECDHE-ECDSA-DES-CBC3-SHA SSLv3 Kx=ECDH Au=ECDSA Enc=3DES(168) Mac=SHA1
- SRP-DSS-3DES-EDE-CBC-SHA SSLv3 Kx=SRP Au=DSS Enc=3DES(168) Mac=SHA1