Bachelor thesis
Computer Science

Radboud University

Persistent effects of man-in-the-middle attacks

Author:
Ronnie Swanink
s4382838

First supervisor/assessor:
dr.ir. Erik Poll
E.Poll@cs.ru.nl

Second assessor:
dr. Peter Schwabe
P.Schwabe@cs.ru.nl

10th January 2016

Abstract

Does a man-in-the-middle attack have long term consequences? In man-in-the-middle attacks, an attacker is able to read and alter internet communications. Given these possibilities, a lot of rewriting and sniffing attacks can be executed on popular protocols. HTTP can be tampered with in order to for example inject malware into pages, steal credentials and force victims to cache tampered pages. DNS is easily spoofed, as there are no real countermeasures for MITM attacks. This allows an attacker to store false records in the DNS cache. We also take a look at SMTP, IMAP, POP3, FTP, SSH and TLS/SSL. TLS/SSL is in most cases the standard countermeasure against MITM attacks. However, TLS/SSL has weaknesses as well. We discuss a few methods to circumvent the protection offered by TLS/SSL. In Chapter 9 we explain how an ordinary internet user can protect himself from these kind of attacks in laymans terms.

Contents

Chapter 1

Introduction

The internet as we know it today is huge. Not only in terms of amount of devices connected, but also how many different kinds of information flows around. Complex languages, or protocols, are specified to make sure everyone on the internet understands each other. A web browser for example uses HTTP and HTTPS to request webpages from a server, and email clients use IMAP or POP3 to retreive messages from the mailbox.

However, when two devices talk to each other, their traffic is not only visible by themselves. The internet relies on a massive web of servers that pass information to other servers, just like a physical mail service. But what if the mailman 'attacks' the letters he is meant to deliver? If the mailman reads a letter, he is performing a passive man-in-the-middle attack. If he alters the content, he is performing an active man-in-the-middle attack. Not delivering the letter at all is a denial of service (DOS) attack.

In [6] researchers demonstrated a way to inject malicious javascript code into webpages using a proxy server. But this example raises some questions. Given that an attacker is able to read and alter any traffic flowing from/to a client to/from the internet, what kind of attacks are possible? And is it possible to create some sort of persisting effect?

At the time of writing, a lot of attacks are known. However, most attacks are not documented in academic papers. Most attacks are described in articles on the internet or presented at a security conference such as Blackhat of DEFCON. Because of this, it is difficult to get a complete list of attack vectors.

In this paper we document MITM attacks against a few popular protocols. This documentation consists of a technical specification of the attack and possible mitigations. In Chapter 9 we explain without jargon what an ordinary internet user can do to protect him- or herself from the attacks described in this paper.

The idea for this paper originates from an interest in WiFi-hacking and MITM attacks, and was inspired by [6]. From personal experience while

working at a computer store I believe a lot of people are unaware of any dangers while browsing the internet.

Attacker model All the attacks described in this paper assume the attacker has already devised a way to be able to read and alter all traffic of his victim. Possible ways to archieve this are for instance evil access points (See [20]) or ARP-spoofing (See [1]). Attackers can also be internet service providers or network administrators at internet exchanges.

The goal of his attack is to leave some persisting effect on the victim by tampering with the victims' connection. A result might be for example a poisioned DNS cache.

Scope and structure In Chapter 2 we give a broad overview of the type of effects a MITM attack might have.

In the following chapters, we discuss attacks against some selected protocols. The protocols we will discuss are:

- HTTP and HTTPS for web browsing in Chapter 3.
- DNS and DNSSEC for name resolution in Chapter 4.
- SMTP, IMAP and POP3 for e-mail in Chapter 5.
- FTP, FTPS and SFTP for file tranfers in Chapter 6.
- SSH for remote terminals in Chapter 7.
- TLS/SSL for secure tunnels in Chapter 8.

We have listed all documented attacks against these protocols, and a few attacks that exist but do not seem to have been documented, most likely because of a small usability. The reader should note that these lists of attacks try to be exhaustive, but there is a possibility an attack was not noticed and therefore not included. We give at least one mitigation against every attack, but not an exhaustive list.

The protocols are chosen because they are widely used and are responsible for the functioning of a large part of the internet. TLS/SSL does not serve a direct use, but is the standard solution to secure otherwise plaintext protocols. Some attacks can have effects on higher level protocols. The attacks discussed are all MITM attacks, and should enable the attacker to leave some persistent or semi-persistent effect on his victim.

All the attacks we discuss work by abusing a mechanism in these protocols, for instance caching commands. We will not discuss attacks against specific vulnerabilities for example in parsing code.

Additionally, we discuss a few attacks that allow the attacker to bypass the security offered by TLS/SSL and SSH.

Alongside these attacks, we discuss methods to protect a system against these attacks, and who is responsible for executing that method (for instance the end user or hosting provider). In Chapter 9 we discuss how a non-technical end user is able to protect him- or herself against the discussed attacks.

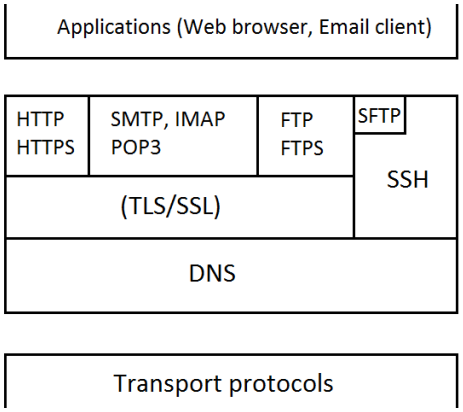

Figure 1.1: Relations between protocols

We will focus on attacks executed at protocol level. More specifically protocols in the upper layers of the ISO/OSI model. The relation between the protocols can be seen in Figure 1.1, this means that an attack on a lower protocol might also have an impact on a higher one.

Topics we will not discuss in this paper are:

- Malware.
- How to attack specific applications.
- How to set up a MITM attack.
- How to use the tools mentioned in attack descriptions.
- Cracking a TLS/SSL or SSH connection with cryptanalysis.

Readers with knowledge about networking should be able to understand every section. Readers that are only interested in the protection measures should read all the "Protection" sections and Chapter 9 at the end. Readers without a computer background should only read Chapter 9.

Chapter 2

Effects and Impacts

If an attack is successfully executed, this will have some kind of effect on the victim. This effect is also called the impact. Some attacks might have a direct impact on the user, for instance by injecting code into a webpage. Some attacks do only have an impact some time after the attack. Finally, some attacks do not have an impact on the victim user but on the integrity of higher protocols. Some impacts do not fall into this model, which causes classifying impacts into categories to be difficult.

In this chapter we give a broad overview of the impacts posed by the attacks presented in this paper.

2.1 Sniffing

Sniffing, or eavesdropping, is the act of reading traffic and collecting information. Mostly, sniffing is used to steal credentials that are sent in plain text. The impact of sniffing depends on what kind of information is stolen. Passwords to bank accounts for instance are more valuable to an attacker than passwords to some forum.

In some cases the attacker is only able to sniff encrypted credentials. Depending on the system used to encrypt these credentials, the attacker might launch a password cracking attack.

2.2 Malware

Malware is software that runs on the device of a victim. The damage this malware might cause differs greatly as new malware keeps popping up. Malware can be installed in multiple ways, for instance:

- Using misleading advertisements to trick a user into installing the malware himself.

- Exploiting some browser vulnerability with for example a buffer overflow.

It is possible for an attacker to add malware to existing files. For instance by injection code while a file is being transfered.

Further discussion of malware is however outside of the scope of this paper.

2.3 Binary patching

Downloaded files can be altered in-transfer by an attacker. With executable files, an attacker is able to rewrite part of the code in the executable file to install malware, or perform other malicious actions. There are multiple ways to achieve this. A tutorial on how to add code to executables can be found at [13].

2.4 Cookie insertion/stealing

Cookies are a part of the HTTP and HTTPS protocols. Cookies allow a web server to temporarily store some data on the clients' device. This data might consist of an session identifier in order to remember is the user was logged in or not. By stealing cookies, the attacker might be able to copy the session of the user and thus be logged in as the user. By inserting cookies, the attacker places some forged data into the cookie storage of the user. This forged data might for instance contain a stored XSS attack.

2.5 Cache poisoning

A cache is some local storage where a device can temporarily store some data in order to speed up future requests. A DNS cache for example stores the resolved IP-addresses for a while. Cache poisoning happens when an attacker manages to place some forged data into a cache. In the case of DNS, a poisoned cache might lead to a browser connecting to the wrong IP-address when visiting a site.

2.6 Fake certificates

Fake certificates on their own can not cause any damage, but they are really dangerous when these certificates end up being trusted as root. If an attacker gets his root certificate on a device, the attacker is able to pretend to be every site he wants to be and listen in on normally encrypted connections.

2.7 Session hijacking

Some protocols work with sessions. In these sessions, the user authenticates himself and proceeds to do whatever needs to be done. An attacker might observe connections from these protocols, and wait until the user has authenticated himself. When the user has successfully authenticated himself, the attacker can hijack the session. By closing the connection between the user and the attacker, the attacker is able to claim the session for himself.

Another way for the attacker to hijack a session is by injecting commands into a session. This way commands can be executed in name of the user without the user knowing. However, if the user decides to terminate the connection, the attacker is also disconnected. This attack is also more difficult for the attacker to execute, as he has to make sure the victim does not receive any of the attacker-injected material.

2.8 Downgrade attacks

A downgrade attack is an attack in which the attacker interferes in the communication to restrict the usage of newer (and safer) protocols or capabilities. By downgrading a protocol, the attacker might for instance force two parties to use weaker encryption, or no encryption at all. The attacker might also strip security-related parts of packets.

Chapter 3

HTTP

3.1 Description

HTTP, of HyperText Transfer Procotol, is a protocol designed for 'distributed, collaborative, hypermedia information systems'. A more simple description is that HTTP is the protocol responsible for retreiving webpages from a server. This retreiving of webpages is done by a program called a web browser. Some well-known web browsers are Firefox and Internet Explorer.

HTTP is old. HTTP/1.0 was specified in 1996 [RFC1945], followed by HTTP/1.1 in 1999 [RFC2616]. Recently, a research project from Google called SPDY was proposed as HTTP/2 [RFC7540]. We will primarily focus on HTTP/1.1 as this is the current working standard.

The problem with old protocols is that they are not designed for security. HTTP does not use any encryption or authentication. Anyone watching traffic on a router is able to see what pages are accessed, and tamper with the connection if desired. To prevent this, a standard called HTTPS was introduced. In HTTPS, the plaintext traffic is tunneled through a SSL or TLS connection. See Chapter 8 for more information about SSL/TLS.

In the past few years, more and more software is offered via a web page. Mobile apps also tend to include web pages as part of their user interface. Because of this, hacking a browser is almost as valuable as hacking a computer itself. As browsers are becoming increasingly complex (See [2]), more attack vectors are introduced.

The attacks described below read and tamper with HTTP traffic in order to abuse webpages and the browser itself. How these attacks can be used in practice, for instance with the framework described in [6], will not be discussed.

3.2 Attacks

3.2.1 Sniffing

The most simple way to abuse HTTP is to watch connections go by and sniff POST requests. If a user tries to log in on a website his username and password are sent in plaintext. By simply listening what is sent, an attacker is capable of collecting all kind of credentials. This attack can be combined with SSL stripping or splitting (3.2.6, 3.2.7) in order to capture more data. A tool such as mitmproxy [12] is able to display HTTP traffic.

Mitigation By upgrading the webserver to use HTTPS, this attack can be easily mitigated.

3.2.2 Cookie insertion

As described in [23], in some browser implementations it is possible to override or set cookies whose HTTPS-only flag is set by defining a cookie with the same name in a HTTP response.

With a standard HTTP connection, an attacker is also able to inject cookies. To accomplish this, the attacker has to add `Cookie` headers to a HTTP response. This kind of attack, against standard cookies or HTTPSOnly ones, allows an attacker to inject cookie material into the session of a user.

Mitigation The attack described in [23] is possible due to a bug in webbrowsers. Browser users should keep their browser up-to-date, and the browser vendors should fix their implementations. Using HTTPS should prevent an attacker from injecting cookie material into connections. Browser users are also able to disable cookies altogether, but this breaks a lot of websites.

3.2.3 Cache poisoning

When browsing the web, a lot of content has to be frequently downloaded over and over again. This is a waste of bandwidth and time. To prevent this waste, browsers implement a content cache. HTTP provides headers with which a server can control caching. Larger assets such as images will be cached most of the time to improve load times.

But there is a problem. HTTP is a plaintext protocol, including the cache-control headers. As demonstrated in [22], it is possible to inject malicious javascript code into requested script files, which will then be cached by the browser. Having attacker-controlled javascript on a webpage is a security disaster. With javascript, the attacker is for instance able to read all form inputs on a page, show whatever the attacker wants the user to see and leak session information. Another serious problem is the possibility to

load a BeEF hook (See [30]). BeEF can be used together with Metasploit to exploit browser vulnerabilities by injecting all kinds of exploits with the ultimate goal of breaking into the system. However, these exploits are beyond the scope of this paper.

Mitigation Upgrading the webserver to HTTPS solves this problem. Wiping the browser cache is a simple way to remove any poisoned files.

3.2.4 Binary patching

Downloading an executable can be a risky thing. A lot of sites distribute all kinds of software installers bloated with crapware. But even if you manage to find the official page of the program you are trying to find, your download is still at risk. A possible attack is to alter binary files with a proxy. By wrapping the original executable code in malware, an attacker is able to abuse downloads offered via HTTP. There is evidence that malicious TOR nodes use this method (See [34]). This attack is now implemented in BDFProxy [42], a utility which automatically wraps executable files.

By wrapping an executable with malware, an attacker is able to create a persistant backdoor using this attack. How this works exactly is beyond the scope of this paper.

Mitigation Again, upgrading the webserver to HTTPS solves this problem altogether. The end user should also use an antivirus scanner to detect and remove known malware. Publishes binaries should be digitally signed, as this allows users to verify the downloaded contents.

3.2.5 Mixed content abuse

When building a HTTPS-enabled webpage, developers should make sure that every asset is loaded with HTTPS. When an asset is loaded with HTTP, the attacker is able to tamper with it. By tampering with one asset, the integrity of the whole page can be jeopardized. A breakdown of mixed content errors is given in [39], along with a table of browsers differences regarding this attack.

Mitigation To migitate this attack, two HTTP extensions have been introduced:

- CSP (Content Security Policy [48]): Allows a developer to specify from which domains content may be loaded in order to prevent cross site scripting.

- HSTS (HTTP Strict Transport Security [RFC6797]): With this header, a developer can specify that every asset on a page, including those that would normally be loaded with HTTP, must be loaded using HTTPS.

These extensions should be enabled and implemented by server administrators and browser vendors. Only having support for these on one side of the connection does not yield any benefits.

Current popular browsers (Firefox, Google Chrome, Internet Explorer) alert their users when mixed content is loaded. However, which requests are considered harmful and how this is shown to the user is implemented differently. Google Chrome for instance changed its policy about mixed content, see [7].

3.2.6 HTTPS stripping

HTTPS stripping is an attack that downgrades a HTTPS connection to HTTP with a proxy. With some clever rewriting done by the proxy, the victim is presented with a HTTP version of any site he is visiting. This attack does not do anything on its own, but it allows the attacker to defeat HTTPS and regain tampering capabilities. This attack was popularized by the tool sslstrip [28], built by Moxie Marlinspike.

This attack differs from HTTPS stripping. With this attack, the attacker maintains a TLS/SSL secured connection to the destination server, but the connection to the victim user is not secured. This way, the attacker does not have to forge certificates.

Mitigation Website administrators should use HSTS (See [RFC6797] as not all implementations of sslstrip are able to handle this header. When browsing, using a plugin such as HTTPS Everywhere (See [15]) can help, as this switches the browser to HTTPS automatically. Finally, the user should pay attention to the URL bar, and leave a page when HTTPS seems to be missing.

3.2.7 HTTPS splitting

Just like HTTPS stripping, HTTPS splitting is a method of regaining tampering capabilities. HTTPS splitting works with a proxy. The victim connects to the proxy, and the proxy presents the user with a fake certificate for any site the user requests. The proxy, being the CA of these certificates, is able to decrypt any traffic from the victim. The proxy establishes a normal HTTPS connection with any server the victim wants to visit. Sslsplit [40] is a popular tool for this.

This attack differs from HTTPS stripping. With this attack, the attacker maintains TLS/SSL secured connections to both the victim and the

destination server. The attacker does have to forge certificates for every secure connection the client initiates.

In some cases, vulnerabilities in certificate checking allow the attacker to produce valid certificates without actually being a certificate authority. See Chapter 8 for more information about TLS/SSL and certificates.

Mitigation This attack is possible due to the way TLS/SSL is organized on the web. See chapter 8 for Mitigations.

3.2.8 General SSL/TLS attacks

As HTTPS is just HTTP tunneled using SSL or TLS, all attacks on SSL and TLS might give an attacker the oppertunity to attack the underlying HTTP data. See chapter 8 for a breakdown on these attacks.

3.3 Protection

A website administrator should use HTTPS whenever possible, as this prevents a lot of attacks. On top of that, HSTS should be used to enforce the use of HTTPS. If a site contains a lot of sentitive data, the administrator might consider pinning key information (See chapter 8 for more information). This provides protection against HTTPS splitting.

Browser vendors should strive to separate secure and insecure parts of a website. This is already done with the HTTPS Only flag on cookies, but these implementations sometimes do not work completely as expected (See [23]).

HTTPS stripping can be partially prevented by implementing HSTS on the server side. Browser developers should implement these features as well. The client should consider using HTTPS Everywhere or a comparable tool to automatically switch to HTTPS.

When downloads are published, it is recommended to publish the hash digests of these files as well. It should be noted that these hashes have to be transferred by HTTPS, to prevent the attacker from editing the hash. Binary files can also be signed, this allows verification as well.

Chapter 4

DNS and DNSSEC

4.1 Description

When browsing the internet, a computer does not actually connect to "google.com". Instead, it connects to for instance `74.125.136.113`. This conversion is done by DNS, as specified in [RFC1034] and [RFC1035]. Because of this, DNS is the so-called phonebook of the internet. This conversion is done by sending small questions to a hierarchy of DNS-servers. All these questions are deliberately small, as servers receive massive amounts of requests (See [11]).

DNS does not use any encryption, so tampering with requests has been a common problem. In order to solve this, DNSSEC was introduced. DNSSEC implements cryptography, allowing the receiver of a DNS-response to verify that no one has altered it.

DANE, or DNS-based Authentication of Named Entities, is another addition to the DNS security toolkit. DANE is a possible alternative for the currently used PKI system (See 8). We will not discuss DANE, as the adoption of this protocol is much lower than DNSSEC (See [43]).

DNS is riddled with problems. Server outtages can cause problems for large groups of people (See [41]). Poorly configured servers can also be abused to increase DDOS attack power (See [35]).

DNS, being a plaintext protocol, does not provide any protection against MITM attacks. DNSSEC on the other hand does implement signing, and does provide protection against MITM attacks. However, unless the usage of DNSSEC is forced by a client, it is possible for an attacker to strip all DNSSEC parts.

A lot of research has been done on DNS and its security. However, this research is mostly pointed at attacking large DNS-caches. Here we focus on attacks happening close to the victim, where the attacker might for instance run a malicious DHCP server.

4.2 Attacks

4.2.1 DNS cache poisoning

When a DNS request has been made, the response packet is sent in plain text. This allows an attacker to substitute the real IP with an IP that is controlled by the attacker. This allows the attacker for instance to run all attacks described in Chapter 3.

To speed up loading times, DNS implements a caching mechanism. Without caching, a DNS request would have to be made for almost every connection. A response is allowed to specify how many seconds it should be cached with its TTL field. [RFC1035] paragraph 2.3.4 specifies that the size of this field is 32 bits. This allows up to 4,294,967,295 seconds of caching, or a bit more than 49710 days.

By setting the TTL to the maximum value, an attacker is able to make his attack persist for all poisoned domains. After disconnecting, the cached records live on, routing the domains to the wrong IP addresses.

Mitigation Preventing DNS spoofing is rather problematic. Neither DNS nor DNSSEC do prevent a MITM attack. DNSSEC does allow a receiver to verify the correctness. Domain owners should use DNSSEC to prevent DNS spoofing attacks. However, with a MITM attack it is trivial to strip the DNSSEC parts. Clients should therefore enforce DNSSEC verification whenever possible.

4.3 DNSSEC downgrade attack

DNSSEC is gaining traction (See [8]), but a lot of domains still use standard DNS to resolve their domains. The attacker has limited attack power because of DNSSEC.

Any domain that is not using DNSSEC can be spoofed, even if the clients' resolver validates DNSSEC records. This is because the chain of trust ends at the last nameserver in the chain that does support DNSSEC. Every record after this cannot be validated.

Domains that do use DNSSEC can be spoofed under some conditions. Not all DNS resolver servers support DNSSEC, and these will not provide their clients with all DNSSEC functions. Clients that are using these servers can have their DNSSEC responses downgraded and spoofed.

Mitigation Clients can opt to only accept DNSSEC responses, but this might lead to a large portion of the internet to be unreachable. As DNSSEC adoption grows, this becomes less of a problem.

4.4 Protection

DNS does not provide any MITM protections. A user is able to connect with a VPN server in order to increase the difficulty of spoofing any DNS records. This does not protect the user against attacks aimed at the resolver server, such as the Kaminsky attack (See [32]). These attacks are however not possible if DNSSEC is used.

Enabling DNSSEC provides the verification most of the time. Some resolver servers do not support DNSSEC, leaving their users vulnerable to spoofing attacks. Users behind resolvers that do support DNSSEC are able to verify answers, and are protected against MITM attacks.

If DNSSEC adoption keeps growing, resolvers might consider to only accept DNSSEC signed records. This will however break a part of the internet.

Chapter 5

SMTP, IMAP, POP3

5.1 Description

SMTP, IMAP and POP3 are protocols used to send and receive mail. The email infrastructure consists of mailservers and mail readers such as Outlook. Mail is sent using the SMTP protocol. If a server receives a message for one of its users, the message is stored. Reading mail is done by receiving the messages from the server by POP3 and IMAP. POP3 used to be the standard for collecting messages, but it has mostly been replaced with IMAP. Just as HTTP and DNS, these three protocols do not use encryption by default. Instead, TLS/SSL tunnels should provide security.

5.2 Attacks

5.2.1 Sniffing

By default, user information is sent in plaintext. Because of this an attacker is able to simply collect credentials by listening on the wire. SMTP and IMAP implement different kinds of authentication which do not leak the password to the attacker.

The authentication mechanisms of these protocols do however support ways of authentication without revealing the credentials. The default authentication method is PLAIN. There are some other methods, for example CRAM-MD5, that use hashing in order to keep the credentials a secret.

To sniff on SMTP, IMAP and POP3 sessions, generic packet sniffing tools can be used. The penetration testing framework metasploit ([37]) does include tools to capture email credentials.

Mitigation The server should be configured to accept TLS/SSL connections. The user should upgrade his connection to use SSL/TLS. As a server

administrator, leaking passwords can be prevented by allowing more secure authentication methods.

5.2.2 Method downgrading

Setting up a secure tunnel is possible in two different ways in case of the mail protocols. Servers can provide connections that enable TLS/SSL from the start, but there is a second option allowing a client to choose to use TLS/SSL if so desired. In the second case, the client connects to the server, and is (in plaintext) presented with a list of possible actions, including STARTTLS. An attacker is able to strip this line, and force the user to continue on in plaintext. This attack is not only used for malicious reasons. Some corporate routers disable encryption inside corporate networks in order to be able to scan email messages for viruses. Some discussion about STARTTLS can be found at [9]. Besides the downgrade attack, there are more arguments against using STARTTLS. For instance, STARTTLS breaks TLS termination proxies.

Another way for an attacker to downgrade a connection is by removing login mechanics from the server EHLO. When a connection is set up between a server and the client, the client should authenticate himself. The most simple way is for the client to send his username and password directly to the server, but this might leak the credentials to a listening attacker. To prevent this, more authetication mechanisms were introduced, and server administrators are allowed to choose which ones they want to support on their servers. The server will then tell any connecting client which mechanisms are supported. An attacker is able to remove methods from this list, and might for instance leave the client to believe the only way to authenticate is PLAIN.

Mitigation Most of the email clients configured to use STARTTLS will refuse to connect when the STARTTLS flag is missing. A bigger problem arises when the user is setting up his email client. In this case, the user should be informed that his provider does allow STARTTLS. If the option is provided, users should connect to the mail server with TLS/SSL enabled from the start.

Server administrators should disable the PLAIN login method, as this way of authenticating is extremely vulnerable to sniffing. Users connecting to email servers should exclude the PLAIN mechanism from being used.

5.2.3 Email sniffing

Even if an email sent from a client to his/her mail server was transmitted securely, there is a potential vulnerability in the communication between SMTP servers. Not all SMTP servers support the transmission of email via

a secure connection, forcing the transmission of email messages to plaintext. If an attacker is able to find a way of placing himself between these two servers, he is able to read all mail sent between these servers and potentially modify them.

Mitigation As a mail server administrator, mail servers should be configured to at least try using an encrypted channel. By forcing encryption, some mail might be silently discarded as the other party does not allow encryption. As a user, it is best to stay away from mail servers not using encryption. Larger providers such as Gmail do employ secure connections most of the time. The end user is also able to use some kind of signing/encryption tools such as PGP to protect his messages.

A different method to add confidentiality to email is S/MIME, defined in [RFC2045]. S/MIME has comparable functionality to PGP. The content of the message is encrypted, and the headers are left untouched to aid in routing. S/MIME is not well known, and like PGP suffers from the problem that it is not simple to set up (See [31]).

5.2.4 Malware insertion

Email messages are made up of some headers with text data. Because of this the attacker is able to inject attachments or text into messages with relative ease. Healthy attachments can also be infected. Email servers tend to run virus scans on incoming messages, but injecting malware when a message is retreived bypasses that. Some email clients sort mail based on antivirus headers added by the scan done on the mail server, so this can be abused to gain the users' trust.

Mitigation Upgrading to SSL/TLS stops the attacker from tampering with the connection. Running a local antivirus scanner should prevent the user from opening appended malware. Antivirus scanners at server level can detect any sent or injected malware.

5.2.5 Fake certificates

Email clients seem to be notorious for having bad SSL/TLS implementations and/or bad certificate checking when reading about this on the internet. If this is actually true is hard to tell, as there does not seem to be an actual statistic about this. The goal of using fake certificates is to generate an invalid certificate, and trick a client into connecting with a proxy server. This proxy server relays the email traffic to the real server, decoding and encoding traffic on the fly.

In [46], problems regaring domain validation are shown. When email servers connect with each other, the presented certificates have to be checked

against a domain name. This domain name will be the MX record associated with the destination email address. With standard DNS, this domain can be spoofed. This article might also explain why people think email encryption is broken. It is not caused by the TLS implementation itself, but the way the client has to determine what certificate to use.

Mitigation See Chapter 8 for Mitigations. In order to prevent an attacker from spoofing MX records, server administrators should implement DNSSEC or DANE in order to secure their certificates.

5.3 Protection

Mail server administrators should configure their mailservers to at least support encryption. Wether the administrator should avoid using STARTTLS is arguable, because this might leave the client vulnerable to the method downgrading attack in Section 5.2.2. This does however break some legitimate uses of stripping STARTTLS capabilities. Enforcing encryption secures email transfer between servers, but this causes messages destined to servers without encryption support to be dropped. A pleasant side effect of enforcing encryption is that a lot of spam is dropped as well, as spambots usually do not support encryption.

Server administrators should disable the PLAIN mechanism for authentication. Email users should disable PLAIN authentication as well.

Domain holders should register their domains with DNSSEC. By using DNSSEC, an attacker is unable to spoof MX records to another server, preventing MITM attacks against the TLS/SSL connection. DANE is a system to allow administators to attach their certificate to the DNS records of a domain. This solves the current certificate validation problems.

To improve awareness about security, email providers should be transparent about their security. At the time of writing, it is difficult to find out wether a email provider uses encryption. Google did announce their support, but this was meant as a incentive to encrypt email globally. If a provider does not support encryption when sending email, the user might consider using PGP or S/MIME. These tools are not easy to set up, so this option will probably not work for ordinary internet users.

Email users should run an antivirus scanner, but antivirus is also possible on the server level. Mail servers are able to detect and remove malware before it reaches the user.

Chapter 6

FTP

FTP, or File Transfer Protocol, is a protocol used to tranfer files over the internet. Currently, FTP is mostly used to upload files to websites and to provide public downloads. FTP is defined in [RFC959]. There are multiple ways to connect to FTP servers:

- The first and obvious way is to directly connect to the ftp server.
- To provide integrity and confidentiality, FTPS (FTP over SSL) can be used.
- Another way to connect to FTP is by tunneling all FTP traffic through SSH. This is called SFTP. By using SFTP, the actual FTP server doesn't need to be directly connected to the internet.

FTP on its own does not use any encryption or verification, which allows for a man-in-the-middle attack. FTPS and SFTP were introduced to provide the neccesary security.

At the time of writing, a lot of public facing FTP servers do still use FTP instead of a secure variant such as SFTP or FTPS (See [3]). When downloads are offered via an insecure connection, some sites publish hashes to verify if the download was not tampered with.

6.1 Attacks

6.1.1 Sniffing

As with any protocol that does not utilize encryption, FTP is vulnerable to sniffing attacks. Passwords can be captured using standard packet sniffing tools or with tools such as the sniffers offered by the Metasploit framework (See [37]).

Mitigation Sniffing can be prevented by upgrading the connection to FTPS, or by using SFTP. Both of these protocols might however be vulnerable to splitting attacks. See Section 6.1.5 for more information on splitting attacks.

6.1.2 Binary patching

Because FTP does not utilize encryption, an attacker is able to modify files sent from server to client and vice-versa. The client can be attacked by infecting downloaded executables with malware. The server can be attacked by adding malicious code to php scripts, or by infecting uploaded executables.

Mitigation Again, upgrading the connection defeats simple MITM attacks. If a server is solely used for providing people with files, publishing the hash codes of these files allows people to detect if their files have been modified. Be sure that these hash codes are sent securely, as the attacker might try to alter these as well.

6.1.3 Injections in NAT-FTPS servers

This attack does only apply to FTPS servers positioned behind an active NAT. As described in [45], it is possible for an attacker to inject data into a secure FTPS session. Because of NAT, the server needs to downgrade the FTPS connection to plaintext in order to allow the NAT to rewrite the PASV and PORT commands. When this happens, the attacker can intervene and inject data into an upload.

Mitigation By statically assigning ports to the server the NAT doesn't have to modify any packets. This allows the connection to be secure all the time. Using an FTPS terminating proxy on the NAT device might also solve this problem.

6.1.4 Injections with TCP termination

As described in [45], by injecting a TCP termination (RST/FIN) it is possible to inject data into a FTPS session. By terminating the connection, the attacker forces the victim to try and restart the connection. This causes the connection to be downgraded to plaintext for a moment. The attacker is now able to inject some prepared data into the upload.

This attack is possible due to the handling of TCP terminations. A client should send a TLSShutdown message before terminating the connection, but most do not. FTP servers implement this as a corner case and allow this behaviour.

Mitigation FTP servers should not allow resuming after a non-graceful termination of the connection. This might break compatibility with some clients, but these clients do not follow the standards.

6.1.5 Splitting attacks against SFTP and FTPS

In the case of FTPS, attacks against SSL/TLS (See Chapter 8 have an impact on transport security. Attacks against SSH (See Chapter 7) have an impact on the transport security of SFTP.

6.2 Protection

As FTP does not provide any MITM protection, connections should be made securely. A FTP connection can be secured by using FTPS or SFTP. A lot of FTP usage originates from webhosting, but these providers do not always advertise alternative/secure ways to connect.

In general, downloads should be published together with hashes. This does not prevent MITM attacks, but this allows users to verify if a file was transferred without any interference.

Uploads should also be done via a secure channel, such as SFTP or FTPS. This is in the best interest of any hosting provider, as this prevents malicious code from being injected into an upload. This does not protect the hosting provider against malicious users. Server administrators should therefore use an antivirus utility.

FTP uses a second connection for actual file transfers. When a FTP server uses TLS/SSL and is positioned behind a NAT, encryption needs to be dropped in order to correctly forward connection ports. Ports can be statically assigned by a network administrator to allow the connection to stay encrypted all the time. Another way to prevent connection downgrades is to use SFTP. Because SSH supports multiple channels, it is possible to dynamically create extra connections without having to downgrade the connection.

Chapter 7

SSH

SSH is a relatively new protocol. The first formal specification is done in [49]. SSH was created in order to prevent password sniffing attacks on a university network. The goal was to replace plaintext protocols such as telnet and rlogin. SSH has become more than just a login protocol. Besides the standard remote shell, SSH can be used for file transers and even VPN connections.

SSH consists of a few layers:

- The first layer is the transport layer. This transport layer provides SSH with connection security and compression, this layer provides a secure tunnel.
- A user authentication layer is responsible for client authentication. Authentication can be done in multiple ways, including the well-known password method and authentication by public key.
- The last layer is the connection layer. This layer is responsible for channels and SSH services. A single SSH connection is able to run multiple channels at once. Channels can be of different types, and can be used for traffic forwarding.

SSH is divided into two different specifications, named SSH1 and SSH2. SSH1 is known to be riddled with vulnerabilities, and was replaced with SSH2 in 2006. Support for SSH1 is dropping, but servers supporting this protocol still exist at the time of writing (See [4]).

The attacks discussed here are mostly mitigated in the same way. Because of this we discuss the general mitigations in Section 7.2.

7.1 Attacks

7.1.1 Sniffing

Sniffing passwords is possible on SSH. As described in [44], by simply rewriting the version string a connection can be downgraded to the SSH1 protocol. The SSH1 protocol allows the attacker to easily sniff the credentials. There are a lot of tools available to sniff SSH1 passwords, for instance: ettercap and dsniff.

However, downgrading the connection requires the SSH server to still support the SSH1 protocol version. Support for this insecure version is dropping, but some hosts are still vulnerable.

In the case of SSH2 the attacker has fewer possibilities. The only publicly available tool to perform MITM attacks on SSH2 is jmitm2 as decribed in [18]. This tool is able to sniff SSH2 passwords, but this does alert the victim of a change in server fingerprint. If public key authentication is used on the SSH2 connection, it is not possible to sniff passwords.

A problem with SSH server impersonation is that the host signature will differ from the original server. This causes SSH clients to display a warning, or refuse connecting altogether.

7.1.2 Session hijacking

If an attacker manages to obtain MITM capabilities, he is able to alter the commands sent from client to server. The attacker might for example hijack the session by disconnecting the client after authentication was successful. The attacker is also able to inject a channel for himself, allowing the client to continue without notice.

A demonstration for this attack against SSH1 is described in [10]. At the time of writing, there are no publicly available tools to hijack SSH2 sessions.

7.1.3 Attacks on VPN and other tunneled connections

SSH is capable to tunnel traffic between two endpoints. A tool called sshuttle (See [33]) is capable of using SSH as a 'poor man's VPN'. SFTP is a FTP connection in which the FTP traffic is first tunneled through SSH to the server. If an attacker is able to obtain MITM capabilities on the main SSH connection, he is able to see and attack every channel of the SSH session as well.

Mitigation Traffic that is tunneled through SSH is visible to the SSH server, and any attacker that manages to get access to the connection. It is possible to apply encryption to any traffic tunneled through SSH, for instance by using HTTPS, to prevent an attacker from undoing the protection offered by SSH.

7.2 Protection

Server administrators and users should disable SSH1 support as the newer SSH2 is more secure. Users should not simply accept a new host signature if they connect to their SSH server.

To fully prevent MITM attacks against SSH2, users and server administrators should work together to set up public key authentication. As described in [5], using public keys prevents MITM attacks because of a change in how keys are negotiated.

Chapter 8

TLS/SSL

As seen in the previous chapters, a lot of protocols have not been designed to be secure when used on the open internet. In 1993, Secure Network Programming was introduced to add transport security to otherwise unsecured protocols. This was followed by SSL 1.0, 2.0 and 3.0, all developed by Netscape. TLS was introduced in 1999 as the successor of SSL 3.0. SSL 3.0 was deprecated in June 2015. TLS 1.2, as defined in [RFC5246], is the current version of TLS. The naming of these protocols is somewhat confusing, as a lot of references to SSL actually refer to TLS.

The TLS/SSL suite of protocols provide a secure tunnel. These tunnels provide a point-to-point connection that provides confidentiality as well as integrity by using encryption and certificates. The goal of TLS and SSL is to secure otherwise insecure protocols. A plain HTTP connection can be for instance tunneled through TLS, preventing MITM attacks.

However, TLS and SSL do have their flaws. Attacks against these protocols have been published, the POODLE attack for instance driving the final nail in SSL 3.0s' coffin (See [RFC7568] section 4.1). A lot of attacks are targeting the cryptographic aspect of these protocols, and are mostly of academic interest.

TLS and SSL rely on certificates to exchange key information when connections are made. These certificates contain information about the public key used, and what domain it can be used for. Some certificates called root certificates are pre-installed in a system, and are trusted by default. These root certificates can be used to sign intermediary certificates. Intermediary certificates are not trusted on their own. If the intermediary certificate is signed by a trusted higher level certificate, such as a root or another intermediary certificate, the intermediate certificate is trusted as well. Leaf certificates are the certificates on the end of the tree, and do actually provide key material to secure a connection. Typically, a certificate used to secure the connection to a website is signed by an intermediary certificate. These trees of certificates are also called certificate chains.

Root certificates are owned by certificate authorities, or CAs. Well-known CAs include VeriSign, or the hacked DigiNotar. The hack at DigiNotar shows the amount of trust that is put into these authorities, as any certificate that is signed by a trusted CA is trusted as well.

Thorough checking of these certificates is critical, because otherwise an attacker would be able to create false certificates. A problem with certificate checking was demonstrated by Moxie Marlinspike in [25], where anyone with a valid certificate could sign new certificates. In 2009, Marlinspike demonstrated another attack in [26], defeating ownership tests.

The problem with having the ability to obtain valid but false certificates is that they allow an attacker to pretent to be the server. A client connecting to a server receives a valid certificate from the attacker, triggering no alarms. The attacker can simply forward all traffic to the real server with a second TLS/SSL connection.

8.1 Attacks

8.1.1 Downgrade attacks

As described in [29], TLS implements some sort of downgrade dance. With every connection attempt that fails, an older version of the protocol is tried instead. An attacker is able to interfere with this downgrade dance, and force both parties to use SSL 3.0.

When the connection is downgraded, the attacker might try to steal information with for instance the POODLE attack. Another option is to save the traffic and try decrypting it later, this strategy is applied by the NSA. If the attacker is able to find the keys, he is able to recreate the server certificate and pose as the server. Cracking keys is still a tough problem however.

Mitigation Downgrading attacks can be mitigated by disabling support for older and broken, in particular SSL 3.0 and RC4, should be disabled. This can be done on the client and server side. Client software might also warn the user if an older cipher suite is used, this is already the case with some internet browsers.

8.1.2 False certificates

If an attacker is able to create valid certificates for any domain, it's trivial to terminate the TLS/SSL connection and perform a MITM attack. There is a tool called sslsplit (See [40]) that is capable of creating forged certificates given an authority. A tool called sslsniff (See [27]) is able to forge certificates and exploit some checking vulnerabilities. Sslsniff is also capable of substituting specific certificates.

But how does the attacker obtain these certificates? The most ideal case for an attacker is having access to a root or intermediary certificate from a CA. In this case, he is able to create certificates for any domain he wants. With this, he is able to break up almost all TLS/SSL connections without alarms ringing at the user. Another way would be to obtain a certificate for a specific domain. Most CAs use automated systems to verify domains, and this can sometimes be easily abused. As described in [17], a researcher was able to obtain a SSL certificate for live.com by registering the address `sslcertificates@live.com`.

Sometimes, these fake certificates are used for good. Larger companies for example use their own CA to be able to inspect in- and outgoing traffic for viruses. This does raise questions about privacy, but in some cases network security is crucial. An example of how false certificates can be abused is demonstrated by the Kazachstan government. Starting in 2016, all Kazachstan citizens will have to install a CA certificate issued by the government, to allow traffic interception. Some discussion about this can be found at [24].

Mitigation False but valid certificates can be blocked with the use of certificate pinning on the client side. When connecting to a server, the server transfers its' certificate to the client. The client then checks the fingerprint of this certificate, and matches that to the fingerprint the client is told to trust. If the fingerprints do not match, the client abort the connection. Even if an attacker is able to produce valid certificates, he would have to get the fingerprint right in order to fool the client.

OWASP maintains a detailed explaination of the workings of certificate pinning at [47].

8.2 Protection

In the general sense, TLS and SSL are the protection. These protocols do have to be used in a correct way to actually offer any protection however.

Clients and servers should disable old and broken protocols and cipher suites. The linux distribution Fedora for instance has implemented this since version 23 (See [38]). App developers are able to pin their certificates, and should use this functionality to prevent their app from getting attacked.

Setting up certificate pinning on HTTPS connections is a more involved process. Some browsers implement pinning for large sites, and allow site operators to add headers to add their own pinning. The problem with these headers is that they only work on a clean load.

Certificate pinning can be setup with SMTP servers, but it is an involved process. Postfix for example requires manual specification of fingerprints and policies regarding each domain.

A solution to these complicated setups might be to add certificate fingerprints to DNSSEC responses. This way a clean load is not required, and no manual specification is required.

Chapter 9

Personal protection

In this chapter we discuss tips and guidelines for protection against the attacks described in this paper. This chapter is free of technical jargon and is aimed at internet users without a computer science or security background. All guidelines are grouped based on the type of internet activity.

This chapter covers attacks described in this thesis. For more information on how to safely use the internet, see for instance [14], a website dedicated to educating dutch people about online safety.

Regardless of internet activity, its strongly recommended to keep the operating system and any programs up-to-date. If a system did not receive recent updates, it might be unnecessarily vulnerable. Running antivirus software on the side is also recommended, as this blocks a large portion of malware and hacking attempts.

9.1 Web browsing

While browsing the web, some sites are accessed in a secure way, and others are not. To determine wether the connection to a website is secured, check the URL-bar. If the URL of the site starts with http://, the site was loaded insecurely. If the URL starts with https://, the site is using encryption.

Most browsers display a green lock icon left of the website address when the connection is secure. If the lock is missing, or if it is colored yellow[1], something is not completely right and therefore the connection might not be completely secure. Some browsers display a pop-up message asking the user if the browser should load 'mixed content', Internet Explorer is an example of such a browser. These questions should be answered with no, as mixed content might leave you vulnerable to be attacked.

[1]The meaning of a yellow lock depends on the browser, ranging from weak hashes to mixed content. It is best to have people distrust the yellow icon, this way website administrators will try to have their certificates set up correctly

Some websites do not just have a green lock, but a complete green bar, including some name. This is called extended validation. If a website has this green bar, it should be considered safe to use as well.

If the connection is not secured, an attacker is capable of eavesdropping the connection and/or modifying it. In this case, it is not recommended to:

- Log into important services such as bank accounts.
- Log in with usernames and passwords that you also use with important services.
- Download executable files.

Browser plugins exist to help users regain their privacy and also secure their connections:

- HTTPS Everywhere (See [15]) is a plugin that forces the browser to use secure connections whenever possible.
- NoScript (See [21]) is a plugin to disable scripts on all webpages. If an attacker manages to add malicious code into a website, the browser will ignore it when this plugin is installed.

9.2 Email

The security of email starts at configuring the email reader. Most internet service providers publish simple step-by-step instructions for setting up your email reader, but these settings might not be the most secure. To check wether the offered settings are secure, look at the connection settings of the email reader. Look for the following:

- Receiving email:
  - If using POP3, port 995 should be used. TLS/SSL should be enabled.
  - If using IMAP, port 993 should be used. TLS/SSL should also be enabled.
- Sending:
  - The best protection is offered when using port 465, and have TLS/SSL enabled.
  - A second option is to use port 587, and require the use of TLS/SSL.

If this is too complicated, a local computer store is most likely able to help.

With a secure connection to the mail server set up, it is important to know not all communications can be deemed safe. Some destinations might not have a secure connection when transfering mails and/or while delivering them. Determining what providers do and don't use these secure connections is complicated. Google has enabled this for the Gmail service, and other providers most likely have too.

A final but somewhat complicated way to secure an email reader is by using plugins such as Mailvelope (See [16]) or Enigmail (See [36]). Using extra encryption requires the receiver to support this as well.

With the email client set up for security, it's still important to be alert. The general rules of email security still apply here. It is not recommended to open email attachments you don't trust. Also, clicking on links might lead you to websites trying to infect you or steal your password.

9.3 File transfers

In this section we focus on file transfers done with FTP. FTP is the standard way to upload files to web hosting space. Most hosting providers publish a connecting with FTP manual on their site, but this is not a secure way to connect.

Some hosting providers have started to support a more secure way to access the files of a website by supporting SSH and SFTP. This support is still scarce. Out of the ten best rated hosting packages on `www.hostingwijzer.nl`, none support connecting by SFTP of FTPS. When looking for a hosting provider, it is recommended to look for SSH support.

One.com for instance offers SFTP support with most packages, and they provide instructions on how to set up your file transfer client to use SFTP. Yourhosting.nl advertises SFTP as an 'advanced' feature, and doesn't provide instructions. Mijndomein.nl doesn't offer SFTP at all.

If you are using a provider without SFTP, it is recommended to only connect to your provider from a home network.

9.4 Shell access

A connection to a remote shell, for instance the shell access provided by hosting providers, is secured by default. On Linux, the `ssh` program should be used with `-2` as an argument to force SSH2 usage. On Windows, the PuTTY program can be launched with the `-2` argument to force SSH2 as well.

In order to maximize shell security, public key authentication should be used. This feature does require configuration on the server as well. To set this up, contact the server administrator.

If public key authentication can not be used, be sure to check the fingerprint of the server you are connecting with. If the fingerprint has changed, the software will alert you of this. Contact the server administrator if the fingerprint has changed without notice. Do not connect if an unknown fingerprint is shown to you.

9.5 VPN

To further improve security and to regain some privacy, consider using a VPN. Using a VPN adds extra security to a connection, and hides the true origin of your internet traffic. In general, VPN providers offer software to simplify configuration. Using a VPN is recommended when connecting with a public network, such as free WiFi services.

Chapter 10

Future Work

In this thesis we discussed MITM attacks against a few protocols. There is however a lot of material outside of the scope that we did not cover. There are a lot of protocols in use today that we did not cover and might be vulnerable to MITM attacks as well, for instance OpenVPN and IPSec. We also did not cover different types of attacks such as DoS and social engineering attacks. These attacks are a field of reasearch on their own. In short, there is still much to learn about.

We discussed a few protocols to give a higher level overview of the current state of security. However, by focussing on one specific protocol in the future, more elaborate attack possibilies might be found.

As the DANE extension of DNS is not adopted as well, we opted to skip this protocol. Research into the possibilies of DANE might yield to major advancements in the PKI.

Malware was mentioned throughout this thesis, but we did not further discuss the capabilities malware can provide to an attacker. Research might be done in how to infect a system, how malware can stay undetected, or what kinds of malware exist today.

In this thesis we did not focus on a specific platform. Different platforms contain their own specific security challenges. Apps for instance are a somewhat new concept, and problems that have already been solved in traditional software resurface again here. See for instance [19]. By researching vulnerabilities in apps, these newer platforms can mature.

Chapter 11

Conclusions

Before writing this thesis, I expected security researchers to document their findings in papers. This was not exactly the case. MITM attacks against TLS and SSL get a lot of interest from the academic community, but these attacks do mostly focus on the cryptographic side. Other attacks try to break into network- or data link protocols. A lot of attacks were found on the open web, where people documented their findings in blog posts. It is interesting to note that the non-cryptographic aspect of security is a more open topic, not restricted to the academic community.

The research itself was very interesting. While the state of internet security is looking somewhat grim, there are a lot of technologies to improve this. Reading into all kinds of attacks and mitigations left me fascinated by the systems built to secure our communications. What did surprise me was the limited availability of tools to attack SSH2. The only tool to effectively attack SSH2 seems to be jmitm2.

As shown in this thesis, the internet is built upon protocols that were designed before security became an issue. Since then, new protocols have been introduced to secure the expanding web. In this thesis we collected some of the most prominent attacks and discussed how to protect a system. These protections might not last forever. Internet security is a cat-and-mouse game, and it is just a matter of time before new vulnerabilities are discovered.

Bibliography

[1] URL: http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html.

[2] URL: https://www.openhub.net/p/chrome.

[3] URL: https://www.shodan.io/search?query=ftp+port%3A21 (visited on 03/01/2016).

[4] URL: https://www.shodan.io/search?query=ssh\%2B1.99 (visited on 10/01/2016).

[5] abb. *SSH Man-in-the-Middle Attack and Public-Key Authentication Method*. Dec. 2010. URL: http://www.gremwell.com/ssh-mitm-public-key-authentication (visited on 27/11/2015).

[6] Chema Alonso and Manu "The Sur". *Owning Bad Guys {& Mafia} with JavaScript Botnets*. 2012. URL: https://media.blackhat.com/bh-us-12/Briefings/Alonso/BH_US_12_Alonso_Owning_Bad_Guys_WP.pdf (visited on 01/10/2015).

[7] Sebastian Anthony. *Chrome finally kills off the HTTP-HTTPS "mixed content" warning*. Oct. 2015. URL: http://arstechnica.com/information-technology/2015/10/chrome-finally-kills-off-the-http-https-mixed-content-warning/ (visited on 19/10/2015).

[8] APNIC. URL: http://stats.labs.apnic.net/dnssec/XA?c=XA&x=1&g=1&r=1&w=7&g=0 (visited on 18/12/2015).

[9] Andrew Ayer. *STARTTLS Considered Harmful*. Aug. 2014. URL: https://www.agwa.name/blog/post/starttls_considered_harmful (visited on 07/12/2015).

[RFC7568] R. Barnes, M. Thomson et al. *Deprecating Secure Sockets Layer Version 3.0*. June 2015. URL: https://tools.ietf.org/html/rfc7568 (visited on 02/12/2015).

[10] Julian Beling. *Conducting SSH Man in the Middle attacks with sshmitm*. URL: http://www.giac.org/paper/gsec/2034/conducting-ssh-man-middle-attacks-sshmitm/103515 (visited on 27/11/2015).

[RFC7540] M. Belshe, BitGo et al. *Hypertext Transfer Protocol – HTTP/1.0*. May 2015. URL: https://tools.ietf.org/html/rfc7540 (visited on 05/10/2015).

[RFC1945] T. Berners-Lee, MIT/LCS et al. *Hypertext Transfer Protocol – HTTP/1.0*. May 1999. URL: http://tools.ietf.org/html/rfc1945 (visited on 05/10/2015).

[11] Jeremy K. Chen. *Google Public DNS: 70 billion requests a day and counting*. Feb. 2012. URL: https://googleblog.blogspot.nl/2012/02/google-public-dns-70-billion-requests.html (visited on 12/10/2015).

[12] Aldo Cortesi. *mitmproxy*. URL: https://mitmproxy.org/ (visited on 07/01/2016).

[13] A. Danehkar. *Inject your code to a Portable Executable file*. Dec. 2005. URL: http://www.codeproject.com/Articles/12532/Inject-your-code-to-a-Portable-Executable-file (visited on 03/01/2016).

[RFC5246] T. Dierks, E. Rescorla et al. *The Transport Layer Security (TLS) Protocol Version 1.2*. Aug. 2008. URL: https://tools.ietf.org/html/rfc5246 (visited on 02/12/2015).

[14] Stichting ECP-EPN. *Veilig internetten*. URL: https://veiliginternetten.nl/ (visited on 03/01/2016).

[15] EFF. *HTTPS Everywhere*. URL: https://www.eff.org/HTTPS-everywhere (visited on 19/10/2015).

[RFC2616] R. Fielding, UC Irvine et al. *Hypertext Transfer Protocol – HTTP/1.1*. June 1996. URL: http://tools.ietf.org/html/rfc2616 (visited on 05/10/2015).

[RFC2045] N. Freed, Innosoft et al. *The Transport Layer Security (TLS) Protocol Version 1.2*. Nov. 1996. URL: https://tools.ietf.org/html/rfc2045 (visited on 07/01/2016).

[16] Mailvelope GmbH. *Mailvelope*. URL: https://www.mailvelope.com/ (visited on 02/12/2015).

[17] Dan Goodin. *How is SSL hopelessly broken? Let us count the ways*. Apr. 2011. URL: http://www.theregister.co.uk/2011/04/11/state_of_ssl_analysis/ (visited on 04/12/2015).

[18] David Guembel. *jmitm2*. URL: http://www.david-guembel.de/index.php?id=6 (visited on 27/11/2015).

[19] Priyank Gupta. *Validating SSL certificates in mobile apps*. Mar. 2005. URL: http://priyaaank.tumblr.com/post/81172916565/validating-ssl-certificates-in-mobile-apps (visited on 03/01/2016).

[20] Ethical Hacking. *Evil Twin and Fake Wireless Access Point Hacks: What They Are, How To Defends*. Apr. 2014. URL: http://breakthesecurity.cysecurity.org/2014/04/evil-twin-attack-fake-wifi-hack.html (visited on 10/01/2016).

[RFC6797] J. Hodges, PayPal et al. *HTTP Strict Transport Security (HSTS)*. Nov. 2012. URL: https://tools.ietf.org/html/rfc6797 (visited on 05/10/2015).

[21] InformAction. *NoScript*. URL: https://noscript.net/ (visited on 01/12/2015).

[22] Matt Johansen Jeremiah Grossman. *Million Browser Botnet*. 2013. URL: https://media.blackhat.com/us-13/us-13-Grossman-Million-Browser-Botnet.pdf.

[23] Jian Jiang, Nicholas Weaver et al. *Vulnerability Note VU#804060*. 2015. URL: http://www.kb.cert.org/vuls/id/804060 (visited on 24/09/2015).

[24] Hugo Landau. *TLS and the Policy MitM Armageddon*. Dec. 2015. URL: https://www.devever.net/~hl/policymitm (visited on 04/12/2015).

[25] Moxie Marlinspike. *Internet Explorer SSL Vulnerability 08/05/02*. Aug. 2002. URL: http://www.thoughtcrime.org/ie-ssl-chain.txt (visited on 02/12/2015).

[26] Moxie Marlinspike. *Null Prefix Attacks Against SSL/TLS*. July 2009. URL: http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf (visited on 02/12/2015).

[27] Moxie Marlinspike. *sslsniff v0.8*. Apr. 2011. URL: https://github.com/moxie0/sslsniff (visited on 02/12/2015).

[28] Moxie Marlinspike. *sslstrip*. 2009. URL: http://www.thoughtcrime.org/software/sslstrip/ (visited on 21/02/2009).

[RFC1034] P. Mockapetris and ISI. *DOMAIN NAMES - CONCEPTS AND FACILITIES*. Nov. 1987. URL: https://tools.ietf.org/html/rfc1034 (visited on 19/10/2015).

[RFC1035] P. Mockapetris and ISI. *DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION*. Nov. 1987. URL: https://tools.ietf.org/html/rfc1035 (visited on 19/10/2015).

[29] Bodo Möller, Thai Duong and Krzysztof Kotowicz. *This POODLE Bites: Exploiting The SSL 3.0 Fallback*. Sept. 2014. URL: https://www.openssl.org/~bodo/ssl-poodle.pdf (visited on 02/12/2015).

[30] Nbblrr. *The browser exploitation framework project*. URL: http://beefproject.com/ (visited on 12/10/2015).

[31] Mark Noble. *S/MIME Secure Email - A Beginners Guide*. URL: http://www.marknoble.com/tutorial/smime/smime.aspx (visited on 07/01/2016).

[32] Matthew Olney, Patrick Mullen and Kevin Miklavcic. *Dan Kaminsky's 2008 DNS Vulnerability*. July 2008. URL: https://www.ietf.org/mail-archive/web/dnsop/current/pdf2jgx6rzxN4.pdf (visited on 04/12/2015).

[33] Avery Pennarun. *sshuttle: where transparent proxy meets VPN meets ssh*. Apr. 2010. URL: https://github.com/apenwarr/sshuttle (visited on 27/11/2015).

[34] Josh Pitts. *THE CASE OF THE MODIFIED BINARIES*. Oct. 2014. URL: http://www.leviathansecurity.com/blog/the-case-of-the-modified-binaries/ (visited on 12/10/2015).

[RFC959] J. Postel, J. Reynolds and ISI. *FILE TRANSFER PROTOCOL (FTP)*. Oct. 1985. URL: https://www.ietf.org/rfc/rfc959.txt (visited on 16/11/2015).

[35] Matthew Prince. *Deep Inside a DNS Amplification DDoS Attack*. Oct. 2012. URL: https://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack/ (visited on 19/10/2015).

[36] The Enigmail Project. *Enigmail: A simple interface for OpenPGP email security*. URL: https://www.enigmail.net/home/index.php (visited on 02/12/2015).

[37] Rapid7. *metasploit*. URL: http://www.metasploit.com/ (visited on 07/01/2016).

[38] Inc. Red Hat et al. *Disable SSL3 and RC4 by default*. URL: https://fedoraproject.org/wiki/Releases/23/ChangeSet#Disable_SSL3_and_RC4_by_default (visited on 04/12/2015).

[39] Ivan Ristic. *HTTPS Mixed Content: Still the Easiest Way to Break SSL*. Mar. 2014. URL: https://community.qualys.com/blogs/securitylabs/2014/03/19/https-mixed-content-still-the-easiest-way-to-break-ssl (visited on 05/10/2015).

[40] Daniel Roethlisberger. *SSLsplit - transparent and scalable SSL/TLS interception*. 2015. URL: https://www.roe.ch/SSLsplit (visited on 29/07/2015).

[41] Joost Schellevis. *Ziggo lost tweede storing op door ddos-aanval op dns af te weren*. Aug. 2012. URL: http://tweakers.net/nieuws/104853/ziggo-lost-tweede-storing-op-door-ddos-aanval-op-dns-af-te-weren.html (visited on 12/10/2015).

[42] secretsquirrel. *BDFProxy*. URL: https://github.com/secretsquirrel/BDFProxy (visited on 12/10/2015).

[43] Verisign Labs Shumon Huque. *Next Steps in DANE Adoption*. Oct. 2015. URL: https://indico.dns-oarc.net/event/24/session/6/contribution/23/material/slides/0.pdf (visited on 02/01/2016).

[44] Click Death Squad. *Performing an SSH Man-in-the-Middle downgrade attack*. URL: https://sites.google.com/site/clickdeathsquad/Home/cds-ssh-mitmdowngrade (visited on 27/11/2015).

[45] Milan Singh Thakur. *CRACKING FTPS - NOT SFTP*. Sept. 2015. URL: http://www.sectivenet.com/index.php/blog/76 (visited on 16/11/2015).

[46] Filippo Valsorda. *THE SAD STATE OF SMTP ENCRYPTION*. Apr. 2015. URL: https://blog.filippo.io/the-sad-state-of-smtp-encryption/ (visited on 07/12/2015).

[47] Jeffrey Walton et al. *Certificate and Public Key Pinning*. URL: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning (visited on 04/12/2015).

[48] Mike West, Adam Barth and Dan Veditz. *Content Security Policy Level 2*. June 2015. URL: http://www.w3.org/TR/CSP2/ (visited on 19/10/2015).

[49] T. Ylonen and Helsinki University of Technology. *The SSH (Secure Shell) Remote Login Protocol*. Nov. 1995. URL: http://www.snailbook.com/docs/protocol-1.5.txt (visited on 17/11/2015).