ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

STL SYSTEMTECHNIKLABOR

On the Security of the TLS Protocol

Pascal Jung

Technical Report – STL-TR-2014-01 – ISSN 2364-7167

stl.htwsaar.de

Technische Berichte des Systemtechniklabors (STL) der htw saar
Technical Reports of the System Technology Lab (STL) at htw saar
ISSN 2364-7167

Pascal Jung: On the Security of the TLS Protocol
Technical report id: STL-TR-2014-01

First published: July, 2014
Last revision: July, 2015 (after IEEE Student Conference)
Internal review: André Miede

For the most recent version of this report see: http://stl.htwsaar.de/

Title image source: Flavio Takemoto (flaivoloka), http://www.freeimages.com/photo/1160561

htw saar – Hochschule für Technik und Wirtschaft des Saarlandes (University of Applied Sciences)
Fakultät für Ingenieurwissenschaften (School of Engineering)
STL – Systemtechniklabor (System Technology Lab)
Prof. Dr.-Ing. André Miede (andre.miede@htwsaar.de)
Goebenstraße 40
66117 Saarbrücken, Germany
http://stl.htwsaar.de

On the Security of the TLS Protocol

Pascal Jung
htw saar – Hochschule für Technik und Wirtschaft des Saarlandes

Abstract—**Most applications communicating over the Internet are nowadays secured by the Transport Layer Security (TLS) protocol. This paper takes a look at the protocol, its security goals and currently common attacks to identify gaps and problems in todays secure communication. To accomplish this, three main attack categories are identified and several attacks are analyzed using a four part pattern to describe them in detail and provide some countermeasures to defend against them. In addition, the attack frequency against each category is measured to see which parts are most vulnerable.**

I. Introduction

TLS or in former versions known as Secure Socket Layer (SSL) describes a protocol that is designed to provide a secure communication over a network connection. It is located in Layer five of the ISO/OSI model. Protocols of a higher layer (e.g. HTTP) pass their data to the TLS protocol, which then secures the whole data and passes it to the next layer (e.g. TCP). If a specific application protocol is secured by TLS, it is often referred to *Protocolname* concatenated with an "S", e.g. HTTP*S*.

TLS and its predecessor SSL have a quite long history. The SSL protocol was a proprietary protocol developed by netscape in order to secure ecommerce transactions in the Internet. After version 3.0, the Internet Engineering Task Force (IETF) standardized the protocol and renamed it to TLS [1]. So SSL 1.0 is the very first version of the protocol and TLS 1.2 is the currently latest.

This paper gives an introduction to the TLS protocol and existing threats by explaining different kinds of attacks. Thereby, some general problems of securing the Internet, especially through TLS, are shown. In order to achieve this, first the Protocol and its security goals are described. This foundations are needed to understand the structure of the following attacks and why these are threats at all. After each attack, some countermeasures to defend against them are explained.

II. Security goals

This sections explains the fundamentals to understand the described attacks. Therefore, the security goals that TLS fulfills are explained in order to understand where an attack should aim at.

There are predefined security goals that each secure application should fulfill, such as the Confidentiality, Integrity and Availability (CIA) triad [2]. Furthermore, TLS defines its own goals in its specification [3]. The goals defined by these two specifications and how TLS implements them are discussed in this section.

The primary goal of TLS, defined in its specification, is "Cryptographic security" [3]. This goal already refers to the *Confidentiality* corner of the CIA triad. The CIA triad describes the most necessary security goals a secure application has to fulfill in order to provide data security. The following three enumeration items describe the corners of the CIA triad [2].

1) *Confidentiality* refers to the secrecy of the data, which means the data must not be disclosed to unauthorized persons. This is achieved by encrypting the application data before sending it.
2) *Integrity* guarantees, that the data is not altered by unauthorized parties, especially not by a third party during transport. This goal is reached by the use of a message authentication code (MAC).
3) *Availability* means that the system should always work properly, so that the data can always be accessed by authorized persons.

In addition to the security goals of the CIA triad, TLS also provides us with [4]:

4) *Authentication*, i.e., either party can prove their identity to the other one. This is achieved by the use of digitally signed certificates.
5) *Non-repudiation* so the sender can not repudiate that the data was sent by him [5]. This is also achieved by the use of digitally signed certificates.
6) *Replay protection*, i.e., an attacker can not record the sent messages and send them later again to the server to fake the victims identity. This is achieved by the use implicit sequence numbers.

III. Foundations

This section is a short introduction on how the protocol works in order to understand the described attacks. The whole specification of TLS 1.2 can be found in [3].

The protocol itself is split into two phases, the handshake and the application data phase. The handshake phase negotiates the cryptographic methods – also known as a cipher suite – and the related key material used to encrypt the application data. Furthermore, the certificates are exchanged and checked for validity. After the handshake phase successfully completed, the application data phase enters, which secures the data received from the overlying protocol.

A cipher suite simply contains the cryptographic methods used to 1) exchange the key, 2) encrypt the data, 3) generate the MAC. So the security strongly depends on the chosen cipher suite. If a broken encryption method is negotiated, an attacker might easily be able to decrypt the whole traffic. Every implementation therefore has to implement the following cipher suite [3], that is considered secure (unless explicitly otherwise specified by the applications secured through TLS): `TLS_RSA_WITH_AES_128_CBC_SHA`, what simply means for key exchanging Rivest, Shamir and Adleman cryptographic system (RSA) is used, the application data is secured by Advanced Encryption Standard (AES) with a 128-bit key in Cipher Block Chaining (CBC) mode, and the MAC is generated by the Secure Hash Algorithm (SHA).

The MAC provides integrity by generating a unique code for a particular message. The other party can now check for the validity of the MAC and conclude if the message is valid or if it has been compromised [6].

IV. Attacks

During the last 20 years, several attacks on SSL/TLS were found and fixed [4], [7]. Meyer et al sorted them by looking at the attacked TLS layer. In this paper, they are categorized another way by assigning one of three attack categories. These categories outline three general channels that can be attacked:

The **User** is the target of the attack, so the attack does not try to use a security vulnerability of TLS itself, it merely tricks a person involved with TLS. This might be the end user or any other party involved with TLS (e.g. the certificate authority (CA)). The **protocol** is attacked directly, that means a security vulnerability in the protocol or the used cipher suite is exploited. Or the **implementation** is the target of the attack, that means a bug that leads to a malfunction of the software is used to overcome the security mechanisms.

To portray each category, a currently applicable attack for each category is described in detail. Thus, a better understanding on the attack channels (categories) is gained. Each attack is explained by the following four part pattern:

1) Naming the author of the attack, the category and the threatened security goals (*Founder & Category*).
2) Analyzing the prerequisites to mount the attack (*Prerequisites*).
3) Taking a detailed look on how the attack works in theory and how it can be applied in practice (*Theory & Practice*).
4) Outlining some countermeasures to defend against the attack (*Countermeasures*).

By assigning categories to attacks, we can afterwards analyze which attack types are most common, identify gaps that should be improved and forecast what kind of attacks are most likely to be found next.

A. Heartbleed

1) Founder & Category: The Heartbleed vulnerability was found by Neel Metha from Google's security team [8]. It is an attack on the OpenSSL implementation and breaks the Confidentiality security goal, because an attacker is able to read sensitive data out of the victims memory and may be able to decrypt traffic.

2) Prerequisites: In order to successfully mount the attack, some prerequisites need to be accomplished. The attacked application needs to run a vulnerable OpenSSL version (the attack was fixed within version 1.0.1g). As update cycles are long sometimes, there might be a lot of applications running an old vulnerable OpenSSL version. Some systems might never get an update, e.g. if the application development is discontinued, the user is a computer layman and does not care about updates or the system is not updateable at all.

3) Theory & Practice: The attack exploits a bug in the Heartbeat extension. The extension is used to ask the other party if it is still available. Therefore, the client sends a request containing an arbitrary string and its length. The server then answers with the received string. This is used to keep the connection alive when no data is send in a specific amount of time. The attack is also reversible, that means the attacker could also asks the client if it is still there. As both sides work the same way, we only consider the client attacks server side.

OpenSSL missed in its implementation to check for the validity of the length of the string. It copies the amount of bytes, given in the length field, out of the memory and sends it back to the client. As the length field is 2 bytes long, the attacker could read up to 64 kb of data, stored in the server's memory. This memory area could basically contain anything – server private keys, private user data, server logs, passwords, but also useless binary data [9]. If the attacker was able to steal the private keys of the server, he could decrypt all upcoming and past traffic of that server (as long as perfect forward secrecy is not used [10]). Thus, the server operators that own a signed certificate for their domain, need to request new certificates what might be a time and money consuming process.

4) Countermeasures: To avoid the attack, OpenSSL has to be updated, including all applications that use OpenSSL. In addition, all certificates and password should be renewed. Furthermore, more frequent security audits of TLS implementations could lead to less frequent security vulnerability findings.

B. Man in the Middle & sslstrip

1) Founder & Category: The Man in the Middle (MitM) attack is more likely a principle than a specific attack against TLS. Therefore, there is no direct founder, respectively there is no source stating who ran this kind of attack the very first time. A more advanced variation of the MitM is the sslstrip attack, that was found by Moxie Marlinspike. He presented it at the Black Hat DC in 2009 [11]. Although the attack is several years old, it still can be applied to inattentive users. Both attacks are against the user, because they try to trick him to break the security. Once mounted, they break Confidentiality, Integrity and Non-repudiation. The sslstrip attack additionally breaks Authentication.

2) Prerequisites: The attacker needs to intercept the network traffic of the victim. That is, receiving all data and sending own data pretending it comes from the victim. To accomplish this, an attacker could run an ARP spoof attack on a wireless local area network. Such an attack tricks the address resolution protocol (ARP) of the victims machine to poison the ARP Table.

To avoid a security warning during a straight MitM attack, stating that there is a problem with the certificate, the attacker could inject a compromised root certificate into the victims application [12]. Although, this would be a great advantage, nevertheless it is not necessarily required. The more advanced way of solving this problem is to mount the sslstrip attack, that strips of the SSL (or TLS) protection, so the connection to the client is established unsecured. In order to mount this attack, the victim must try to open the connection without

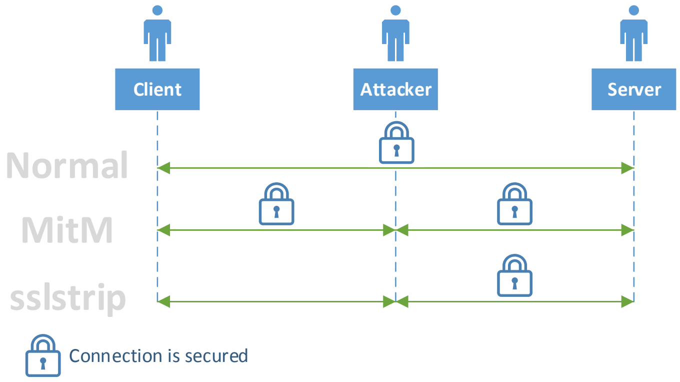

Figure 1. Comparison of normal, MitM and sslstrip connections

TLS. As most applications do not try to establish an unsecured connection, they are not affected. But the attack is applicable to any Browser, as the user decides in first place if he wants to connect though HTTP or HTTPS (most websites redirect their users for security relevant tasks to the HTTPS protocol) and this circumstance makes the sslstrip a really serious threat.

3) Theory & Practice: A normal TLS connection provides end to end security, so an attacker is not able to break in and read the plaintext network traffic or send own requests. In order for the attacker to break in anyway, he needs to tell the client he is the server and tell the server he is the client. So he establishes secure connections to client and server. A comparison of a normal secured connection to a server, a MitM compromised and a sslstrip infected connection can be found in Figure 1.

The problem with this is, that it does not break the protection gained from signed certificates (assuming a secure cipher suite is negotiated [3]). So the application notices, while checking the certificate for validity, that it can not be trusted. As a result, most applications will show a warning or dismiss the communication channel.

As long as the user does not press “Continue anyway”, he is secure. But it is likely for a layman or naive user to ignore this warning and trust the connection. When the user continues the process, the MitM can control the whole network traffic of the victim and is able to do a variety of things, ranging from website defacements to stealing credentials for bank accounts or any other confidential data.

If the attacker is able to inject a compromised root certificate, this warning would be omitted. As the root certificate is responsible for proving that a particular certificate is valid, the attacker would be able to generate valid certificates on the fly for any domain [12]. So the victim would not realize that he is being attacked at all. But to inject a root certificate into the victims machine, an attacker would need to have direct access to the victims machine, which is hard to gain in most cases.

Another way to bypass the certificate issue is to mount the mentioned sslstrip attack. Unlike the pure MitM, it does not open a secured communication channel to the client. Let us assume the attacker tries to eavesdrop the victims HTTPS traffic. At first the attacker needs to listen to unsecure (HTTP) connection attempts. As soon as one is captured, the attacker takes over the whole session by mounting a MitM, the sslstrip way, as seen in Figure 1. This is done by analyzing the HTTP request and sending it to the server. During this process, the MitM might establish a secured connection to the server, depending on the servers TLS policy. After the result is obtained from the server, the attacker can analyze the message and replace all HTTPS links with HTTP links and sends it back to the victim, so he is able to read any subsequent requests and responses. The only difference the user could see, compared to a secured communication, is the missing “S” of HTTPS in the address bar and for some browsers the lock, indicating a secured communication, would be missing. [11]

Marlinspike was able to take over 117 email accounts, 16 credit card numbers, 7 paypal logins and over 300 other secure credentials in 24 hours by mounting the attack via the TOR[1] network [11].

4) Countermeasures: The straight MitM attack can be prohibited by being really critical with certificates and not trusting any suspicious ones. If the user is not careless and does not click on “Continue anyway”, security against MitM attacks is really good. So the feasibility of this attack only relies on the user. Training courses like described in [12] are a good method to teach users, how an ongoing attack would look like.

The more difficult thing is to detect and defend against a sslstrip attack. As only a small difference in the protocolname and a missing lock icon indicate that there might be an sslstrip attack ongoing, it is very difficult to defend against them. The only currently applicable way is to always check the symbols of a secure connection and dismiss it if at least one is missing. Symbols of a secure connection are:

1) Is the protocol HTTPS?
2) Does the domain match the one I want to visit?
3) Is the certificate issued to the domain I want to visit?
4) Does my browser show a lock icon to indicate a secure connection?

C. BEAST

1) Founder & Category: The Browser Exploit Against SSL/TLS (BEAST) attack was developed by Thai Duong and Juliano Rizzo. They presented their work at the ekoparty conference in 2011. Their exploit is based on work by Bard, Möller and Dai [4]. Duong and Rizzo extended their ideas and implemented a working example. Before they implemented it, the attack was mostly meant to be theoretical and there was no practical use for it. It is an attack on the protocol as it directly addresses a flaw in the protocol. When the attack executes successfully, it breaks Confidentiality because the whole traffic can be deciphered.

2) Prerequisites: For the attack to execute successfully, the following prerequisites need to be met: The attacker needs to be able to eavesdrop on the traffic of the victim and send own requests through the current active TLS session. The connection must not use a TLS version greater than 1.0 as the flaw was fixed within TLS 1.1. But this is not a big problem because most servers still support TLS 1.0 [13]. And last but not least, a block cipher encryption method in CBC

[1] https://www.torproject.org

mode has to be negotiated. To eavesdrop the traffic, we could again use an ARP spoof attack or behave as a proxy server. But to gain the possibility to send own requests through the current session, in most cases a Same Origin Protection bypass is needed [14] and we need to execute custom code on the victim's machine or at least his browser.

3) Theory & Practice: To understand the attack, some basics need to be explained. It is crucial to know that exclusive or'ing (xor = $\oplus$) a value with another value cancels the xor'ing out, so $x \oplus y \oplus y = x$. Furthermore, the CBC mode of a block cipher must be known. As a block cipher symmetric encrypts byte blocks, two plaintext blocks that are the same end up to the same ciphertext (encrypted plaintext). This is bad because an attacker could see which parts of a message are the same and stay the same during several requests. To undermine this issue the CBC mode can be used. In this mode each plaintext block gets xor'ed with an Initialization Vector (IV), that is a random value, so every block becomes different. The first IV is generated randomly and for each subsequent block the ciphertext of the block before is used. So the algorithm to generate the cipher text of a message looks the following way (C = ciphertext, P = plaintext):

$$\begin{aligned} C_0 &= Enc_{key}(IV \oplus P_0) \\ C_1 &= Enc_{key}(C_0 \oplus P_1) \\ C_2 &= Enc_{key}(C_1 \oplus P_2) \\ C_n &= Enc_{key}(C_{n-1} \oplus P_n) \end{aligned}$$

TLS 1.0 determined that the last block of a message is used as IV for the next message, so C_n would be used as IV for the first block of the next message. This makes the IV predictable and therefore the encryption vulnerable. TLS 1.1 fixed this issue by adding a unique IV field to every message.

Based on this, Gregory Bard invented the chosen plaintext attack [15]. With it, he stated that he was able to guess the value of a plaintext block and verify if this guess is correct. If the attacker wants to guess block i, he xor's P_{i_guess} (plaintext guess of block i) with C_{i-1} (which has been the IV for block i) and C_n that is the IV for this block. So the attacker would end up with the following equation to build his block that gets injected to the encryption (M = modified plaintext block):

$$M_{i\_guess} = P_{i\_guess} \oplus C_{i-1} \oplus C_n$$

During the next step M_{i_guess} is encrypted by the encryption of TLS to create the cipher block.

$$\begin{aligned} C_{i\_guess} &= Enc_{key}(M_{i\_guess} \oplus C_n) \\ C_{i\_guess} &= Enc_{key}(P_{i\_guess} \oplus C_{i-1} \oplus C_n \oplus C_n) \\ C_{i\_guess} &= Enc_{key}(P_{i\_guess} \oplus C_{i-1}) \end{aligned}$$

$$C_i = Enc_{key}(P_i \oplus C_{i-1})$$

The attacker can now check, if his guess is correct by checking whether $C_{i\_guess} = C_i$. When both cipher blocks are the same, his guess was correct. [4], [15]

This attack was mostly theoretical, because the attacker needed to guess a whole block of about 8 bytes resulting in $255^8 \approx 1,7 \cdot 10^{19}$ possibilities.

Duong and Rizzo extended this idea by inventing the blockwise chosen plaintext attack. Their idea was to know the first seven bytes (as they are predictable under some circumstances) and only have to guess the last byte. Therefore there are only 255 possibilities left, out of the former $1,7 \cdot 10^{19}$, making the attack much more practical than before. In most applications the header data is predefined and therefore predictable and for HTTP the block boundaries can also be moved by controlling the resource that is requested. So the attacker can construct his needed block with just one unknown byte [16], see Figure 2. Using this method an attacker can easily decrypt whole blocks by adjusting the block boundary as needed and guessing a maximum of 255 times (128 on average [16]). Duong and Rizzo used it to decrypt secure HTTP cookies. They were able to decrypt a paypal[2] cookie and steal the session, enabling them to takeover control of the account [17].

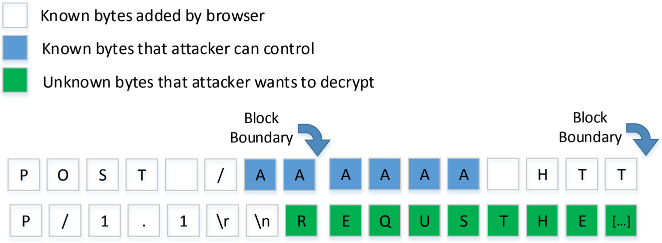
Known bytes added by browser
Known bytes that attacker can control
Unknown bytes that attacker wants to decrypt
Block Boundary
Block Boundary
P
O
S
T
/
A
A
A
A
A
A
H
T
T
P
/
1
.
1
\r
\n
R
E
Q
U
S
T
H
E
[...]

Figure 2. BEAST attack based on [16]

To initiate the attack, the attacker needs to run some JavaScript or Java code in the victims browser. Therefore the victim needs to visit a prepared website, that injects the malicious code.

4) Countermeasures: To defend against the attack, server administrators should forbid the use of TLS version 1.0 and less. But this could lead to problems as not every client supports TLS 1.1 and above.

The user could also disable vulnerable technologies, but that would be a big problem as most websites nowadays use technologies, that might be vulnerable as JavaScript, Java or WebSockets. A more practical way would be to disable TLS version 1.0 or less on the clientside (e.g. browsers).

D. Even more attacks

The previous sections described attacks for each category in detail. Some more attacks were analyzed and categorized into the earlier described categories to get an overview over the attacks frequency on the different categories. The result can be seen in table I. The analysis of the attacks is based on [4], unless otherwise stated. Some improvements of attacks published under a new name, were omitted as they did not address a new flaw but rather extend already established considerations. Each attack can be assigned to one of the three categories as explained earlier. Figure 3 shows the distribution of the attacks on the categories. With 50% of all attacks the protocol is the most attacked category. This arises from the fact that attacks against a cipher suite also count as attacks on the protocol. In comparison, an implementation is less frequent attacked and any user the least frequent.

Most of the attacks against a user aim at the public key infrastructure of TLS certificates by trying to trick the

[2] https://www.paypal.com

CA. This is also a really serious threat as an end user has to trust any certificate signed by the CA and if they are compromised (either by breaking in their infrastructure or tricking an employee) it can not be determined whether a certificate is valid or not.

Unfortunately, the implementation does also sometimes fail as can be seen in the implementation category. Some of these issues would have been avoidable if proper test cases were implemented for this situations [7]. As already mentioned, more frequent security audits could also lead to improved code quality.

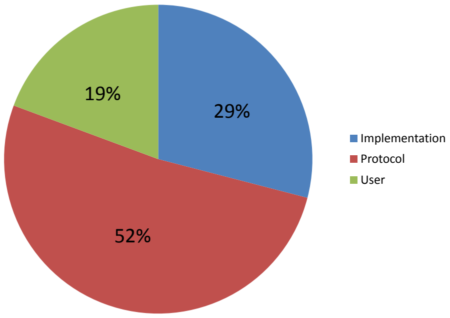

Figure 3. Attack frequency comparison of the different categories

V. Conclusion

These attacks show that communication over the Internet will most likely never be fully secure. Most of these attacks are several years old, but still can be applied under some circumstances. In early 2015 the SSL Pulse of Trustworthy Internet[3] stated that about 84% of the 200,000 most popular websites were vulnerable to the four year old BEAST attack. Generally, the SSL Pulse does not look that great. About 40% support the outdated, vulnerable SSL 3.0 and only 60% support TLS 1.1 or 1.2.

The protocol itself is well-considered and if there is a security flaw, it is fixed within the next version. As the security mostly relies on the used cipher suite and its underlying cryptography, it is crucial to keep an eye on the supported cipher suites. A lot of attacks are only feasible if a special cipher suite is negotiated (e.g. BEAST attack). A big problem with cryptography is the used key lengths. A key length that is currently considered secure, might become vulnerable some years later as computers are speeding up very fast (Moore's law). Moreover, it is important to design cipher suites with a solid cryptographic foundation. In practice, cipher suites are often kept enabled when considered insecure because of downwards compatibility. Two attacks that address flaws arising from legacy features or algorithms are described in [22] and [20].

Unfortunately, server administrators often keep downwards compatibility leaving their servers vulnerable. But sometimes it is needed to keep downwards compatibility as some users might not be able to access the resources if they run an outdated software version. Microsoft did not support TLS 1.1 and 1.2 unless Windows 7, that means Windows Vista and Windows Server 2008 did not support the newer TLS versions, leaving Microsoft IIS and Internet Explorer unprotected [23], [24]. Another problem is the computer layman, who does not know that it is crucial to keep the system up-to-date and has no idea what a certificate warning means and what happens when he presses "Continue anyway". Software vendors supplying secured applications such as browsers should provide their users with a security tour and tell them some best practices about todays Internet security. Another problem with an involved party is the responsibility of the CAs. As outlined before, they have great power because any certificate issued by them is trusted blindly. In consequence, any mistake by a CA has serious impacts on the security of a specific TLS domain.

Moreover, the implementation of the protocol can also be a problem as it is the critical point of TLS secured applications. A well designed protocol without any flaws does not provide reliable security if it is bad implemented. Typically, TLS implementations are provided as libraries that can be embedded in a variety of applications. Thus, a security vulnerability in such a library leads to an immense amount of vulnerable applications. Therefore, it is crucial to have a reliable TLS implementation.

Nevertheless, beside all these flaws, the security of TLS is really good when applied correctly. As seen in the countermeasures to the presented attacks, there are easy ways to defend against attacks, but it is the users or administrators responsibility to implement them. In addition, it is a good thing to mistrust anything and being always alerted when using applications secured by TLS (or any other security protocol).

Acknowledgment

This paper was written as a result to my studies during the Seminar "Past and Future of Science" at htw saar. At this point I would like to thank André Miede for supporting me during writing this paper and giving me countless tips on how to do research and to write a paper properly.

Besides the technical reports series of htw saar, this paper was also presented and published at the 6th IEEE Student Conference.

References

[1] I. Grigorik, *High Performance Browser Networking*. O'Reilly, 2013, ch. Transport Layer Security, pp. 47–79.

[2] W. Stallings, *Cryptography and Network Security*. Prentice Hall; 5th edition, 2010.

[3] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Engineering Task Force, RFC 5246, 2008, Proposed Standard.

[4] C. Meyer, "20 Years of SSL/TLS Research An Analysis of the Internet's Security Foundation," Ph.D. dissertation, Ruhr-University Bochum, 2014.

[5] A. McCullagh and W. Caelli, "Non-repudiation in the digital environment," *First Monday*, vol. 5, no. 8, 2000, (last checked 31.05.2015). [Online]. Available: http://firstmonday.org/ojs/index.php/fm/article/view/778/687

[6] J. R. Black Jr., "Message Authentication Codes," Ph.D. dissertation, California State University at Hayward, 1988.

[7] C. Meyer and J. Schwenk, "SoK: Lessons Learned From SSL/TLS Attacks," in *Proceedings of the International Workshop on Information Security Applications (WISA 2013)*, 2013.

[3] https://www.trustworthyinternet.org/ssl-pulse/

Table I. LIST OF ATTACKS ON TLS SORTED BY THEIR CATEGORY

Implementation	Protocol	User
Random Number Prediction	MAC Does not Cover Padding Length	sslstrip [11]
Wildcard Certficate Validation	Cipher Suite Rollback	Conquest of Comodo CA
Vulnerable Certificate Validation Libs	ChangeCipherSpec Message Drop	Conquest of DigiNotar CA
Weak Random Numbers	Bleichenbachers Million Questions	Accidentally Issuing of Certificates by TURKTRUST
Null Prefix Attacks Against Certificates	Weaknesses Through CBC Usage	Weak Certificates by DigiCert Malaysia CA
ECC Based Timing Attacks	Colliding Certificates	Risks Caused by Unqualified Domain Names [18]
Certificate Validation Issues	Renegotiation Flaw	
Heartbleed [9], [19]	Message Distinguishing	
FREAK [20]	Key Exchange Confusion	
	BEAST [16]	
	CRIME	
	Lucky Thirteen [7]	
	RC4 Biases	
	BREACH [14]	
	POODLE [21]	
	Logjam [22]	

[8] OpenSSL. (2014) OpenSSL vulnerabilities. CVE-2014-0160. (last checked 31.05.2015). [Online]. Available: http://www.openssl.org/news/vulnerabilities.html

[9] B. Schneier. (2014) More on Heartbleed. Blog. (last checked 31.05.2015). [Online]. Available: https://www.schneier.com/blog/archives/2014/04/more_on_heartbl.html

[10] Y. Zhu, "Why the Web Needs Perfect Forward Secrecy More Than Ever," *Electronic Frontier Foundation*, 2014, (last checked 31.05.2015). [Online]. Available: https://www.eff.org/deeplinks/2014/04/why-web-needs-perfect-forward-secrecy

[11] M. Marlinspike. (2009) New Tricks For Defeating SSL In Practice. Presentation. Black Hat DC 2009.

[12] J. Lewis and P. Lunsford, "TLS Man-in-the-middle Laboratory Exercise for Network Security Education," in *Proceedings of the 2010 ACM Conference on Information Technology Education*, ser. SIGITE '10. New York, NY, USA: ACM, 2010, pp. 117–120.

[13] T. I. Movement, "SSL Pulse," Online, 05 2015, (last checked 31.05.2015). [Online]. Available: https://www.trustworthyinternet.org/ssl-pulse/

[14] P. G. Sarkar and S. Fitzgerald, "Attacks on SSL A Comprehensive Study of BEAST, CRIME, TIME, BREACH, LUCKY 13 & RC4 BIASES," iSEC Partners, Inc, Tech. Rep., 2013, (last checked 31.05.2015).

[15] G. V. Bard, "The Vulnerability of SSL to Chosen Plaintext Attack," *Cryptology ePrint Archive: Listing for 2004*, 2004.

[16] T. Duong and J. Rizzo, "Here Come The ⊕ Ninjas," 2011.

[17] T. Duong. (2011) BEAST. Online. (last checked 31.05.2015). [Online]. Available: http://vnhacker.blogspot.de/2011/09/beast.html

[18] C. Palmer, "Unqualified Names in the SSL Observatory," *Electronic Frontier Foundation*, 2011.

[19] B. Schneier, "Heartbleed," Blog, 2014, (last checked 31.05.2015). [Online]. Available: https://www.schneier.com/blog/archives/2014/04/heartbleed.html

[20] K. Bhargavan, "FREAK: Factoring RSA Export Keys," Online, 2014, (last checked 31.05.2015). [Online]. Available: https://www.smacktls.com/#freak

[21] B. Möller, T. Duong, and K. Kotowicz, "This POODLE Bites: Exploiting The SSL 3.0 Fallback," Google, Tech. Rep., 2014.

[22] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelink, and P. Zimmermann, "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice," Online, 2015. [Online]. Available: https://weakdh.org/imperfect-forward-secrecy.pdf

[23] J. Schmidt, "Das BSI und der Elfenbeinturm," *heise Security*, 2015, (last checked 31.05.2015). [Online]. Available: http://heise.de/-2589893

[24] K. K. Panday, "Support for SSL/TLS protocols on Windows," Blog, 2011, (last checked 31.05.2015). [Online]. Available: http://blogs.msdn.com/b/kaushal/archive/2011/10/02/support-for-ssl-tls-protocols-on-windows.aspx