Ahmed Alqattaa
Prof. Dr. Andreas Aßmuth

Analysis of the Internet Security Protocol TLS Version 1.3

Zusammenfassung

Das Transport Layer Security (TLS) Protokoll ist das wohl wichtigste Sicherheitsprotokoll im Internet. Es wird allgemein zur sicheren Authentifizierung und verschlüsselten Kommunikation, konkret beispielsweise für E-Commerce oder Online-Banking verwendet. Ohne TLS (bzw. seinen Vorgänger Secure Socket Layer, abgekürzt SSL) hätte sich das Internet in der uns bekannten Form nicht entwickeln können. Nachdem in den letzten Jahren verschiedene erfolgreiche Angriffe gegen TLS demonstriert werden konnten, sah sich die Internet Engineering Task Force (IETF) gezwungen, eine neue, verbesserte Version von TLS herauszugeben. Im Rahmen dieses Aufsatzes sollen einige wichtige bisherige Angriffe gegen TLS (bis Version 1.2) dargestellt und die neue Version 1.3 bzgl. dieser Angriffe und auch seiner Effizienz analysiert werden.

Abstract

The Transport Layer Security (TLS) protocol is probably the most important security protocol on the Internet. It is generally used for secure authentication and encrypted communication, specifically for e-commerce or online banking. Without TLS (or its predecessor Secure Socket Layer, abbreviated SSL), the Internet could not have developed in its recent form. After several successful attacks against TLS have been demonstrated in recent years, the Internet Engineering Task Force (IETF) was forced to issue a new, improved version of TLS. In this paper some important attacks against TLS (up to version 1.2) will be presented and the new version 1.3 regarding these attacks as well as its efficiency will be analysed.

1 Introduction

The first objective of this paper is to analyse the most important security protocol on the Internet, SSL/TLS, and to test the mechanism of the protocol itself, which consists of several subprotocols. These subprotocols must interact with each other perfectly to provide a secure connection. The second objective is to analyse the flaws in some of the previous versions of TLS and to know how these flaws were corrected in their successor versions. The third objective is to draw a comparison between the new version 1.3 and the current version 1.2; as a consequence, the comparison will show the important improvements by TLS 1.3 and how TLS 1.3 increases security and efficiency. However, the goal of the paper is not to give detailed instructions about how to tamper or attack the protocol but to understand the new level of security and efficiency of TLS 1.3.

1.1 Historical background

TLS and its predecessor SSL are both cryptographic protocols which are two of the most widely deployed cryptographic protocols on the Internet. It was introduced by Netscape in the context of securing the connections between web browsers and web servers and to provide privacy. The advantage of the SSL/TLS protocol is that it is an application independent protocol; that means, any higher-level protocol (e.g. HTTP, FTP, etc.) can layer on top of the SSL/TLS protocol. Nowadays, the protocol is also used for many other protocols, e.g. SMTP, IMAP (for e-mail transmissions) or LDAP (for accessing directories). The original protocol (SSL 1.0) had severe security issues and therefore was required to be fixed repeatedly. SSL 1.0 had never been publicly released because of these serious security flaws in the protocol itself. That is the reason, why the first protocol was version 2.0 which was released in 1995. Unfortunately, this second version had vulnerabilities, too. The next year, SSL 3.0 was released with significant improvements. However, after a couple of years SSL 3.0 had to be deprecated because of some weaknesses in the protocol such as MAC-then-Encrypt in CBC encryption mode.

IETF then introduced a new name for the protocol (Transport Layer Security) and made further improvements for the versions TLS 1.0, TLS 1.1 and the current version TLS 1.2. Just recently, IETF has finished the development of a new version of TLS (1.3) [1].

1.2 SSL/TLS Goals

The primary goal of SSL/TLS is to provide confidentiality between two communicating peers (client and server). It establishes an authenticated and encrypted tunnel through the untrusted Internet. More precisely, the protocol aims to have three basic properties [2][3]:

- Authenticity: The tunnel is authenticated. The server endpoint of the conversation is always authenticated, while the client side is optionally authenticated. Authenticating is realized using asymmetric cryptograghy, e.g. RSA.
- Confidentiality: The tunnel is private. Encryption is used for all messages after a simple handshake, then the data is only visible to the endpoints (end-to-end encryption). However, TLS does not hide the length of the sent data. Dependent on the used ciphersuite the endpoints have to pad the original message which obscures the lengths and improves protection.
- Integrity: The channel is reliable. No attacker can modify any part of the communication without being detected by the endpoints.

These properties should hold even if the attacker has complete control over the network.

2 Attacks on SSL/TLS

2.1 CBC Weaknesses in SSL 3 and TLS 1.0

There are two types of symmetric encryption in SSL/TLS: stream-based (e.g. RC4) and block-based (e.g. AES-CBC). Implementations usually utilise block ciphers. A stream-based encryption system can be used to encrypt any number of bytes. A block-based encryption algorithm encrypts text in blocks. If the plaintext is shorter than a multiple of the block length, it may be padded to fill up the block before it is encrypted dependent on the chosen mode of operation. Counter encryption mode, for instance, does not require padding, while CBC mode encryption does. There are several ways to realise the padding, like PKCS\#7 for example. In CBC mode, a random initialization vector (IV) is used in its first block which is XORed with the plaintext before the encryption algorithm of the block cipher is applied and the resulting value (ciphertext) is XORed with the next plaintext block. The ciphertext block of the previous step is used as IV for the current step [4]:

$$C_0 = E_k(IV \oplus M_0),$$
$$C_i = E_k(C_{i-1} \oplus M_i);\ \text{for } 1 \leq i \leq m\text{-}2,$$
$$C_i = E_k(C_{m-2} \oplus M_{m-1} \,\|\, \text{padding});\ \text{for } i = m\text{-}1.$$

If a new IV is used for every message, then CBC mode provides so called randomised encryption: even if the same message is encrypted using the same secret key, the corresponding ciphertext is different for every encryption process. However, there are two basic ways how CBC is used in SSL/TLS:

- The records are treated as if they were a single block. That means that the IV for record n is the last cipher block C_{n-1}. SSL 3 and TLS 1.0 treat the records in exactly the same manner.
- The records are treated as if each one were independent; that means for each record a new IV is generated. TLS 1.1 and 1.2 treat the records in this way of encryption.

The way SSL 3 and TLS 1.0 treat the records has turned out to be insecure for two: the first one is how SSL 3 and TLS 1.0 treat the IV and the second one is in the padding scheme which causes another security problem [4].

2.1.1 The Padding Scheme Problem in SSL 3 and TLS 1.0

The protocols SSL 3 and TLS 1.0 have two different alert messages to deal with padding-related errors:

- decryption_failed: This alert is received when a TLS ciphertext is decrypted in an invalid way. Either the checked padding was not correct or the message length was not even a multiple of the block length. This alert is always fatal and supported only by TLS 1.0.
- bad_record_mac: This alert is received when a record is received with an incorrect MAC. This alert is always fatal.

If these two different cases can be distinguished by an attacker, then an attack called the padding oracle may work. The main problem in SSL 3 and TLS 1.0 is that they rely on MAC-then-Encrypt and Decrypt-then-MAC, which makes it possible to be able to release information from these errors. In TLS 1.0, it is easy to distinguish the errors because the version supports both types of errors. In SSL 3.0, only one error is supported, namely bad_record_mac. Nevertheless, using a timing attack an attacker can determine whether a bad_record_mac error has occurred. The attacker can simply observe the time until the alert message is generated. If the time is short, the adversary knows the pad was invalid. If the time is long, he knows the padding was valid, but the MAC was invalid. As a result, the attacker still has a padding oracle that enables him to distinguish whether the padding was valid or invalid [4][5].

2.1.2 The IV Security Problem in SSL 3.0 and TLS 1.0

As mentioned earlier, the CBC IV for each record except the first record is the previous ciphertext block, in other words $C_0 = E_k(IV \oplus M_0)$ for the first record and $C_0 = E_k(C_{i-1} \oplus M_i)$ for the rest. However, CBC mode is not secure against chosen plaintext attacks if the IV is predicable for the attacker. In other words, CBC mode is not secure if an attacker can eavesdrop the ciphertexts knowing the IV before the next plaintext block is encrypted [4][5].

2.1.3 The Padding Oracle Attack against CBC

A padding oracle is a system that reveals information (wrong or correct) about the padding. This padding oracle is needed to perform the padding oracle attack.
In addition, an attacker has to be able to modify the encrypted message and send these modified messages to the oracle. The goal of this attack is to know plaintexts from cipher blocks in CBC mode which are padded with PKCS\#7. For example, if the padding is of incorrect length, the ciphertext are rejected and the padding value is considered invalid. This basically checks that the ending of the decrypted record contains the correct padding length. And if the padding has a correct length, the data is considered to be authenticated and is given to the application layer. The problem occurs when an attacker can distinguish between both errors, decryption_failed or bad_record_mac error [5].

The attack supposedly happens when an attacker has obtained a certain ciphertext and tries to determine the corresponding plaintext. The attacker sends this certain ciphertext to the server, the server decrypts the ciphertext and then checks the padding value. If the padding value is invalid then a bad_record_mac error is received. Otherwise a decryption_failed error is received. So the attacker learns something about the decrypted ciphertext. This is a chosen ciphertext attack, but the goal of an oracle padding attack is to know the content of plaintext. The attacker has a valid padding for his chosen ciphertext, so he will try to crack the content of this ciphertext as follows [5][6]:

Assuming the attacker attempts to figure out M_3 from the ciphertext blocks C_1, C_2 and C_3. The attacker generates a guess W for the last byte in M_3; where W has $2^8 = 256$ possibilities. Next, the attacker will XOR the guessed value W with the expected padding 0x01 into the last byte of the previous block C_2 as following $0xP \oplus 0x01 \oplus C_2$; where P is the guessed value W. The attacker sends the modified ciphertext to the server. If the guessed value W was correct then $C_2 \oplus 0xP = 0x00$ and as a consequence $0x00 \oplus 0x01 = 0x01$. If W was wrong the attacker will receive an extra value (not 0x01) and he has to try other values of W. Once the attacker figures out the last byte (byte 16) in M_3 he will try to crack the previous last byte (byte 15), but this time with 0x0202 so $0xYP \oplus 0x0202 \oplus C_2$; where Y is the guessed value or byte 15 and P is the guessed value for byte 16. For byte 14 it must hold that $0xZYP \oplus 0x030303 \oplus C_2$; in this way, the attacker is able to reconstruct the whole block.

Basically, since the length of the block is 16 bytes, the attacker will learn them all after sixteen times 256 queries. So this is a significant attack that enables an attacker to decrypt blocks of the TLS record. However, the problem in this attack is that the errors bad_record_mac and decryption_failed are always fatal, which means when SSL/TLS receives a record with these errors it terminates the connection and renegotiates new keys. As a result, the attacker can submit only a single query to the server and then the connection will be terminated and aborted. Therefore, the correct way to do an authentication and encryption is Encrypt-then-MAC and MAC-then-Decrypt respectively; the result will be an authenticated encryption scheme. TLS 1.1 and newer versions provide such authenticated encryption [7].

2.2 Heartbleed Attack

The Heartbleed bug was a vulnerability in OpenSSL that was published in April 2014 and discovered by Neel Mehta. OpenSSL is an open-source implementation of the SSL and TLS protocols. The library in OpenSSL implements basic cryptographic functions and provides various utility functions. The heartbleed vulnerability in OpenSSL is classified as a buffer over-read that allows an attacker to read portions of the affected system's memory revealing some of its data. Although a connection is protected between both peers by, e.g. TLS 1.2, the vulnerability lets an attacker retrieve data up to 64 kb with each query and the attack may use an unlimited number of queries [8].

2.2.1 How the Heartbleed Attack works

The attack exploits a part that is using a vulnerable OpenSSL instance for TLS client and server. The client has to communicate with the server by sending so called heartbeats which inform the server that the client is still online. TLS heartbeats can be sent by both peers, so the vulnerability can be used to attack clients as well as servers. Heartbeat messages (heartbeat_request and heartbeat_response) consist of type, payload length, payload content, and padding length. Actually, the length of payload must be equal to the payload length sent in the heartbeat request. In other words, the receiving peer should check the payload data length with the actual length of data sent in the heartbeat request. However, older versions of OpenSSL do not check if both values are equal. An attacker can take advantage of this vulnerability in OpenSSL by faking the length of the payload with a much bigger number. For example, instead of saying the payload 'OTH-AW' is six bytes, an attacker claims the length is 100 bytes. In this case, the receiving peer sends to the attacker the input data 'OTH-AW' plus 94 bytes of memory data which could contain IDs, passwords, credit card numbers or personal data of other users [8].

2.2.2 Preventing the Heartbleed Attack

The exploitation of this bug does not leave any trace of abnormal behaviour in the logs. More than a half million servers were affected by the Heartbleed bug. Thus, OpenSSL has solved this vulnerability by checking if the payload data length is equal with the actual length of data sent in the heartbeat request, otherwise, the receiving peer discards this request. The vulnerability is found in OpenSSL version 1.0.1f and 1.0.2-beta1. However, the vulnerability was fixed in OpenSSL version 1.0.1g by both Adam Langley and Bodo Möller. In order to avoid the Heartbleed attack, servers need to upgrade to OpenSSL version 1.0.1g or disable the Heartbeats with DOPENSSL_NO_HEARTBEATS. Clients can test the servers/websites if they are vulnerably for Heartbleed attack. Furthermore, they should not support version 4.1.1 of Android anymore because the Heartbleed bug can also attack this Android version [8].

3 TLS 1.2 vs. TLS 1.3

3.1 Major Differences

A significant improvement has been provided by TLS 1.3. The new version attends to remove obsolete and insecure algorithms that were part of TLS 1.2, including the following [3]:

- The old static RSA handshake without Diffie–Hellman key exchange which does not offer perfect forward secrecy (PFS). In TLS 1.3 it is dissolved with Elliptic Curve Diffie–Hellman Exchange (ECDHE).
- The CBC-MAC, which was responsible for some attacks like POODLE and BEAST. It is replaced with AEAD (Authenticated Encryption with Associated Data) which is a form of encryption which aims to provide confidentiality, integrity, and authenticity.
- The RC4 stream cipher, which is not secure.
- Outdated crypto algorithms, like SHA1, MD5, DES and 3DES.

In addition to the removal of the afore mentioned insecurity cryptographic algorithms, some new algorithms have been added to the new version (TLS 1.3). In the following, some of these are presented [3]:

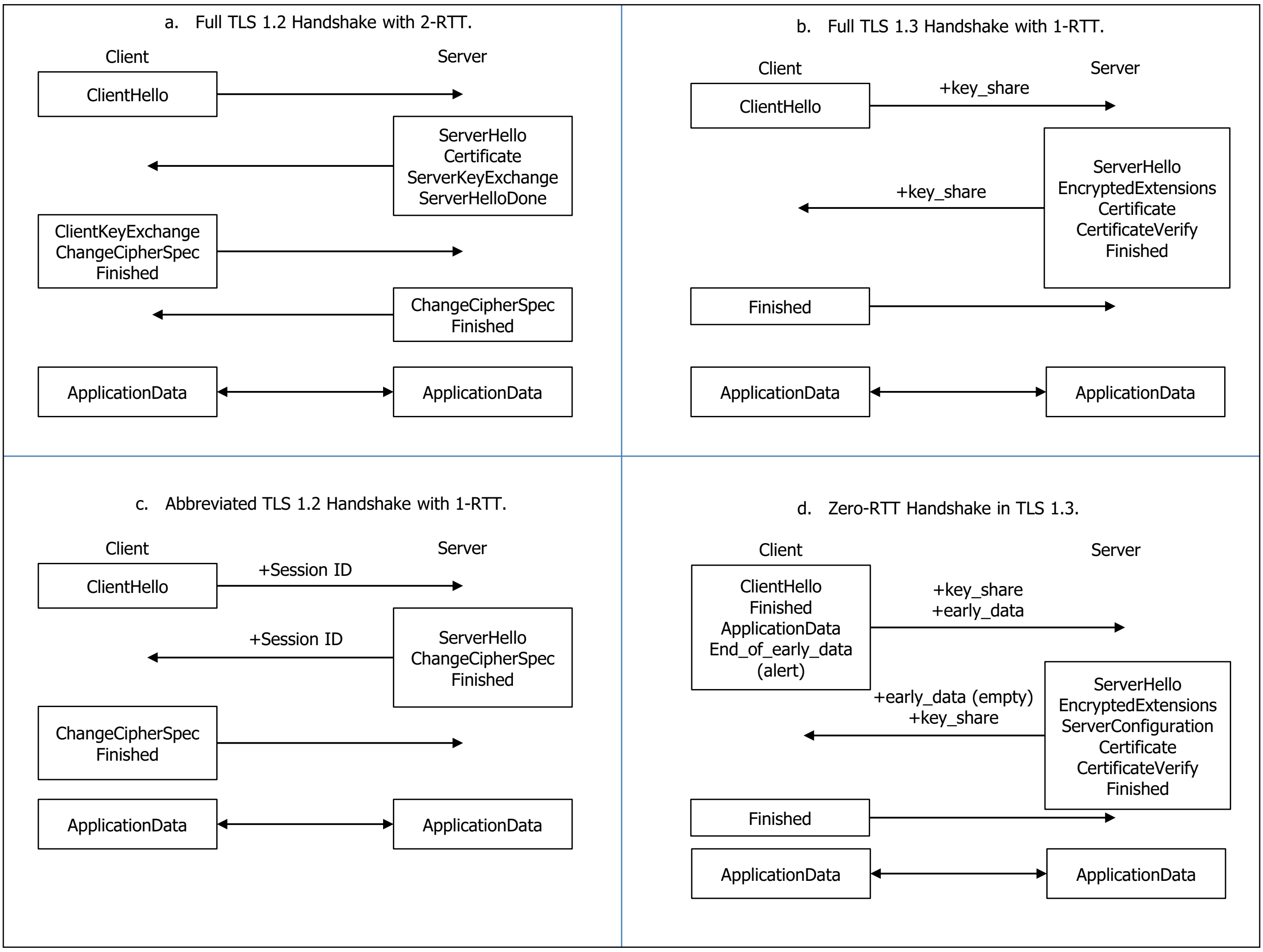

Figure 1: A handshake comparison between TLS 1.2 and TLS 1.3 in two situations. First with a full handshake then with abbreviated handshake [9].

- A full handshake signature. That means all handshake messages after the ServerHello are now encrypted in order to prevent particular attacks.
- Downgrade protection, which protects TLS 1.3 from future attacks in TLS 1.2. This means that even if attacks on TLS 1.2 were found in future, TLS 1.3 connections cannot be downgraded to its vulnerable former version.
- PFS with abbreviated handshakes. In order to resume a session, a TLS 1.2 client uses the same session ID and the same master secret. Therefore, once the master secret is compromised then all resumed sessions can be revealed. However, a TLS 1.3 client uses session tickets to resume a session. Once the master secret is used again, a shared secret is generated and combined with the master secret when calculating keys and IVs. This mechanism then provides PFS.
- In the handshake round trips a Zero-Round Trip Time (0-RTT) mode was added.

3.2 Handshake Protocol in TLS 1.2 vs. 1.3

Using a full TLS 1.2 and 1.3 handshake with ephemeral Diffie-Hellman key exchange, TLS 1.3 encrypts the connection after ServerHello (cf. 1.b). In TLS 1.2, this mechanism is not supported (cf. 1.a). As a result, the handshake in TLS 1.3 is reduced by a full round trip, which can save hundreds of milliseconds [9].
The figures [1.a] and [1.b] show a comparison between the handshake in TLS 1.2 and 1.3 (full handshake with ephemeral Diffie-Hellman key exchange). However, there is a way to do 1-RTT connections in TLS 1.2, but only if the client was connected to the server before. On the other hand, TLS 1.3 can now make the connection with 0-RTT, if the client was connected to a 1.3 server before. That means the endpoints can immediately start sending encrypted data (cf. 1.c and 1.d) [7].

4 Possible Replay Attacks against TLS 1.3 0-RTT

Let us assume that a server can be forced to reboot and to lose its state. Then the server would not reject every client 0-RTT connection and can instead ignore the client's initial data that sent in the first flight. In case that the server does not want the data in the first flight to be lost, the server retransmits the data using new keys. For any client sending 0-RTT key exchange messages together with some encrypted data, an attacker can hence make this data being delivered twice in the following attack (cf. 2). The attacker captures the Transmission Control

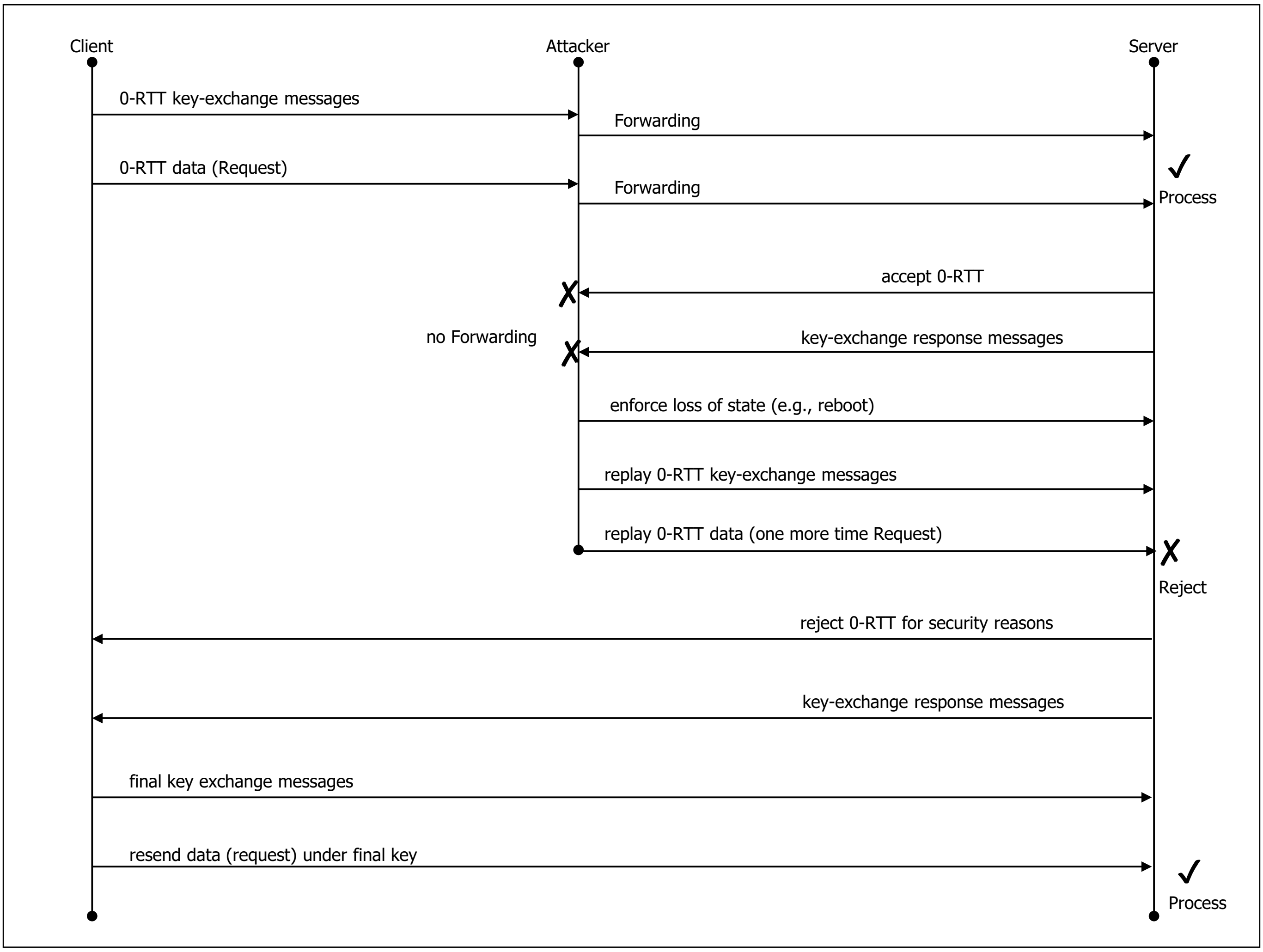

Figure 2: Replay attack against TLS 1.3 0-RTT [10].

Protocol (TCP) connection and creates his own TCP connection to the server, forwarding the first flight and encrypted data which the client has sent. The attacker discards the server's response and tries to force the server to lose its state by, for example, rebooting. Once the server has rebooted, the attacker presents the same messages (initial first flight) again to the server. As a result, the server, for security reasons, rejects the client's first flight and responds with its own key exchange contribution for the final key which the attacker now simply relays to the client. After deriving the final key, the client sends the desired data under the deriving key, which the server will receive and decrypt for the second time. However, because of need to reboot the server and getting in under the client's timeout window, this attack is not easy to process. However, a more realistic attack scenario is to forward the 0-RTT messages two times to a server that operates in two loosely synchronized data center and then drops the first server's response [10][11].

Conclusion

It has been ten years since the last version (1.2) of the Transport Layer Security protocol was launched. This is quite a long time in the world of information security. So finally, it is time to update this protocol and to increase the level of security on the Internet. In recent years, the knowledge of security flaws and how to exploit these has grown. IETF always works on updating the security protocol SSL/TLS in order to secure data transfer through the Internet as good as possible. When we compare the first version and the current version of the protocol, we can notice the huge security improvement by the protocol. All these improvements were needed because of the number of attacks that have occurred in recent years and because of some algorithms that can no longer be considered to be secure. This paper covers a number of these attacks such as Padding Oracle and HEARTBLEED. Moreover, it covers some insufficient algorithms such as CBC with MAC-then-Encrypt mode. But there are a number of other attacks like CRIME, FREAK, LOGJAM, SUPERFISH, POODLE, and BEAST that have not been mentioned. However, the main focus of our research is to evaluate if TLS 1.3 is actually immune against the afore mentioned attacks and provides improvements for the mechanism of the protocol. Besides a thorough security analysis of TLS, additionally issues, like time efficiency, will be addressed in future.

Abbreviations

- CBC (Cipher Block Chaining mode) is an algorithm that uses a block cipher.
- FTP (File Transfer Protocol) is used for the transfer of computer files.
- HTTP (HyperText Transfer Protocol) is used to transfer data on the World Wide Web (WWW).
- IMAP (Internet Message Access Protocol) is used for emails transmission.
- LDAP (Lightweight Directory Access Protocol) is used to access and manage directory information.
- MAC (Message Authentication Code) is used to verify the authenticity of a message.
- RSA (Rivest–Shamir–Adleman) is one of the most important public-key cryptosystems.
- SMTP (Simple Mail Transfer Protocol) is used for emails transmission.

References:

[1] R. Oppliger, "SSL and TLS Theory and Practice", 2016.

[2] D. T. Elgamal and K. E. Hickman, "The SSL Protocol", Internet Engineering Task Force, Internal Draft "drafthickman-netscape-ssl-00", April 1995.

[3] E. Rescorla, "The Transport Layer Security (TLS) Protocol version 1.3", March 26, 2018. https://tlswg.github.io/tls13-spec/draft-ietf-tls-tls13.html

[4] B. Möller, "Security of CBC Ciphersuites in SSL/TLS: Problems and countermeasures", May 2004. https://www.openssl.org/_bodo/tls-cbc.txt

[5] T. Kurokawa, R. Nojima, and S. Moriai, "On the Security of CBC mode in SSL 3.0 and TLS 1.0", 2016. http://isyou. info/jisis/vol6/no1/jisis-2016-vol6-no1-01.pdf

[6] S. Vaudenay, "Security Flaws induced by CBC Padding – Applications to SSL, IPSEC, WTLS ...", pp. 534–545, 2002.

[7] D. Boneh, "CBC Padding Attacks". https://fr.coursera.org/learn/crypto/lecture/8s23o/cbc-padding-attacks

[8] OpenSSL, "The Heartbleed Bug", April 7, 2014. http://heartbleed.com/

[9] T. Taubert, "Improved Handshakes in TLS version 1.3", November 16, 2015. https://timtaubert.de/blog/2015/11/more-privacy-less-latency-improved-handshakes-in-tls-13/

[10] F. Günther and M. Fischlin, "Replay Attacks on Zero Round-Trip Time: The Case of the TLS 1.3 Handshake Candidates", 2017. http://www.felixguenther.info/publications/EUROSP FisGue17.pdf

[11] E. Rescorla, "TLS 0-RTT and Anti-Replay", Internet Engineering Task Force (IETF), 2015. https://www.ietf.org/mail-archive/web/tls/current/msg15594.html

Kontakt:

Ahmed Alqattaa

Ostbayerische Technische
Hochschule (OTH) Amberg-Weiden
Fakultät Elektrotechnik, Medien
und Informatik
Kaiser-Wilhelm-Ring 23
92224 Amberg

a.alqattaa@oth-aw.de

Prof. Dr. Andreas Aßmuth

Ostbayerische Technische
Hochschule (OTH) Amberg-Weiden
Fakultät Elektrotechnik, Medien
und Informatik
Kaiser-Wilhelm-Ring 23
92224 Amberg

a.assmuth@oth-aw.de