Lessons Learned From Previous SSL/TLS Attacks
A Brief Chronology Of Attacks And Weaknesses

Christopher Meyer, Jörg Schwenk
Horst Görtz Institute for IT-Security
Chair for Network and Data Security
Ruhr-University Bochum
{*christopher.meyer, joerg.schwenk*}@*rub.de*

***Abstract*—Since its introduction in 1994 the *Secure Socket Layer (SSL)* protocol (later renamed to *Transport Layer Security (TLS)*) evolved to the de facto standard for securing the transport layer. SSL/TLS can be used for ensuring data confidentiality, integrity and authenticity during transport. A main feature of the protocol is its flexibility. Modes of operation and security aims can easily be configured through different cipher suites. During its evolutionary development process several flaws were found. However, the flexible architecture of SSL/TLS allowed efficient fixes in order to counter the issues.**

This paper presents an overview on theoretical and practical attacks of the last 15 years, in chronological order and four categories: Attacks on the TLS Handshake protocol, on the TLS Record and Application Data Protocols, on the PKI infrastructure of TLS, and on various other attacks. We try to give a short "Lessons Learned" at the end of each paragraph.

***Keywords*-SSL, TLS, Handshake Protocol, Record Layer, Public Key Infrastructures, Bleichenbacher Attack, Padding Oracles**

I. Introduction

In 1994, *Netscape*[1] addressed the problem of securing data which is sent over "the (TCP) wire" in the early days of the *World Wide Web*, by introducing the Secure Sockets Layer protocol version 2. Over the decades SSL gained improvements, security fixes and from version 3.1 on a new name - *Transport Layer Security*[2] - , but the basic idea behind the protocol suite remained the same. A key feature of SSL/TLS is its layered design consisting of mainly two blocks:

Handshake protocol. This is an Authenticated Key Exchange (AKE) protocol for negotiating cryptographic secrets and algorithms.

Record and Application Data protocol. This is an intermediate MAC-then-PAD-then-Encrypt layer positioned between the application and the TCP network layer.

In addition, error messages are bundled in the **Alert protocol**, and the one-message **ChangeCipherSpec protocol** which signalizes the switch from unencrypted to encrypted mode.

[1]http://www.netscape.com
[2]http://datatracker.ietf.org/wg/tls/

A complete communication example (SSL 3.0/TLS 1.x) illustrating the handshake phase finally leading to the application data phase is given in Figure 1.

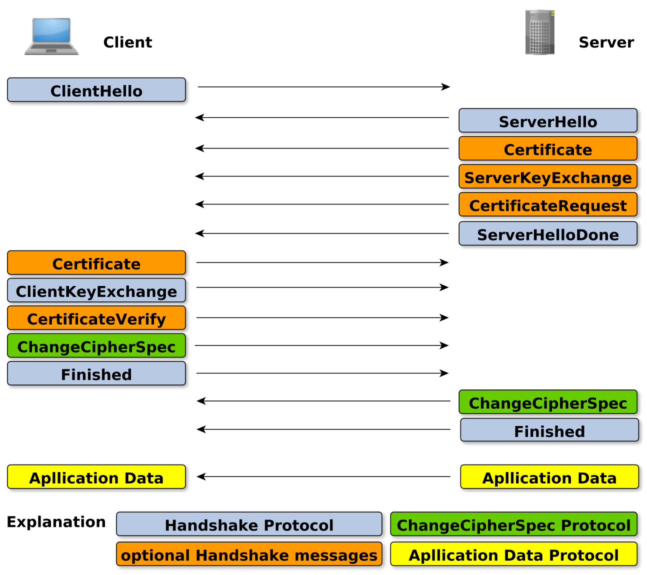

Figure 1: SSL/TLS communication example

Due to space limitations a comprehensive introduction to SSL/TLS is skipped. A detailed view on SSL/TLS is provided by Eric Rescorla in [1].

Many attacks of theoretical and practical nature have been found and partly exploited. Ongoing research improves recent attacks and aims to prove security or finding further security related flaws.

In the following attacks are discussed in four groups: attacks on the TLS Handshake protocol, the TLS Record and Application Data Protocols, attacks on the TLS Public Key Infrastructure, and various other attacks. In each of the four groups, they are presented in chronological order.We tried to formulate a very short "lessons learned" sentence after each attack.

II. Attacks on the Handshake Protocol

*1) **Cipher suite rollback**:* The cipher-suite rollback attack, discussed by Wagner and Schneier in [2] aims at limiting the offered cipher-suite list provided by the client to weaker ones or `NULL`-ciphers. An Man-in-the-middle (Mitm) attacker may alter the `ClientHello` message sent by the initiator of the connection, strips of the undesirable cipher-suites or completely replaces the cipher-suite list with a weak one and passes the manipulated message to the desired recipient. The server has no real choice - it can either reject the connection or accept the weaker cipher-suite. An example scenario is illustrated in Figure 2.

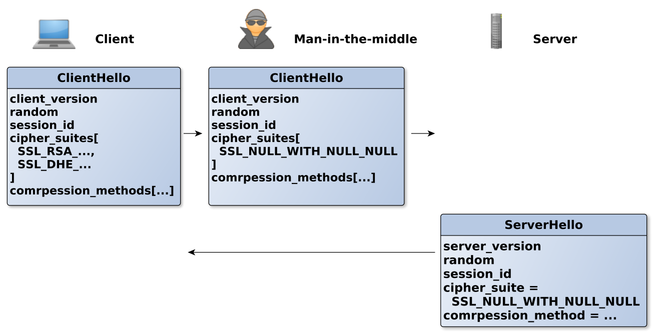

Figure 2: Example scenario for the cipher-suite rollback attack - *based on Source: [2]*

This problem was fixed with the release of SSL 3.0, by authenticating *all messages of the Handshake protocol*, by including hash value of all messages sent and received by the client (the server, resp.) into the computations of the ClientFinished (ServerFinished, resp.) message. However, this hash value explicitly excludes messages of the `Alert` and `ChangeCipherSpec` protocols, leaving room for future attacks.

Lesson learned: This attack illustrates that it is crucial to authenticate what exactly reached the desired target and what was sent. Theoretically, this idea was put forward in [3] with the concept of *matching conversations*.

*2) **ChangeCipherSpec message drop**:* This simple but effective attack described by Wagner and Schneier in [2] was feasible in SSL 2.0 only. During the handshake phase the cryptographic primitives and algorithms are determined. For activation of the new state it is necessary for both parties to send a `ChangeCipherSpec` message. This messages informs the other party that the following communication will be secured by the previously agreed parameters. The pending state is activated immediately after the `ChangeCipherSpec` message is received.

An attacker located as Mitm could simply drop the `ChangeCipherSpec` messages and cause both parties to never activate the pending states. An example is illustrated in Figure 3.

According to Wagner and Schneier the flaw was independently discovered by Dan Simon and addressed by Paul Kocher. The author's recommendation is to force both parties to ensure that a `ChangeCipherSpec` message is received

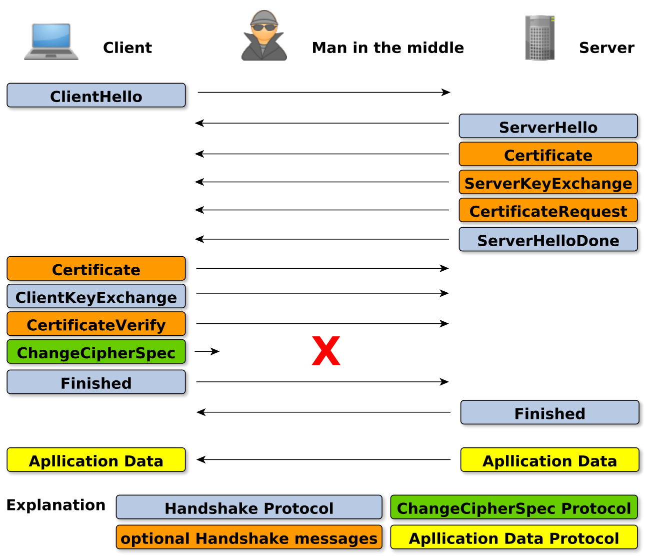

Figure 3: Example scenario for the ChangeCipherSpec message drop attack - *based on Source: [2]*

before accepting the `Finished` message. According to RFC 2246 [4] TLS 1.0 enforces this recommendation.

Lesson learned: See Section II-1.

*3) **Key exchange algorithm confusion**:* Another flaw pointed out by Wagner, Schneier in [2] is related to a feature concerning temporary key material. SSL 3.0 supports the use of temporary key material during the handshake phase (RSA public keys or DH public parameters) signed with a long term key. A problem arises from a missing type definition of the transfered material. Each party implicitly decides, based on the context, which key material is expected and decodes accordingly. More precise, there is no information on the type of the encoded key material. This creates a surface for a type confusion attack.

This attack is, to the best of our knowledge, strictly theoretical until time of writing. Figure 4 gives an attack sketch where a client is fooled into establishing a RSA based key agreement while at the same time performing DHE (ephemeral Diffie-Hellman key exchange) with the server. According to the author's, SSL 3.0b1 hinders this attack.

Lesson learned: This attack highlights the need for context-free message structures: Misinterpretation of a received message should be avoided by providing explicit information on the content.

*4) **Version rollback**:* Wagner and Schneier described in [2] an attack where a `ClientHello` message of SSL 3.0 is modified to look like a `ClientHello` message of SSL 2.0. This would force a server to switch back to the more vulnerable SSL 2.0.

As a countermeasure (proposed by Paul Kocher), the SSL/TLS version is also contained in the PKCS encoded

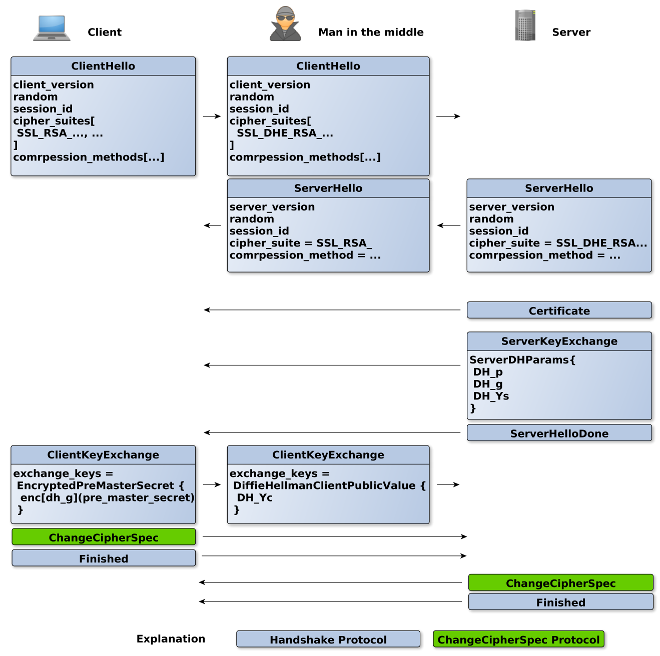

Figure 4: Example scenario for the key exchange algorithm confusion attack - *based on Source: [2]*

Block Type	Padding	Separation Byte	Encapsulated Data
00 02	...	00	PreMasterSecret

Table I: PKCS#1 v 1.5 encoded `PreMasterSecret`

`PreMasterSecret` of the `ClientKeyExchange` message (when RSA-based cipher suites are used). The countermeasure is sufficient, since SSL 2.0 only supports RSA-based key exchange.

Lesson learned: This attack shows that backward compatibility is a serious security threat: The countermeasure described in Section II-1 against modification of single messages does not help, since it was not present in Version 2.0!

*5) **Bleichenbacher Attack on PKCS#1:*** In 1998 Daniel Bleichenbacher presented in [5] an attack on RSA based SSL cipher suites. Bleichenbacher utilized the strict structure of the PKCS#1 v1.5 format and showed that it is possible to decrypt the `PreMasterSecret` in an reasonable amount of time. The `PreMasterSecret` in a RSA based cipher suites is a random value generated by the client and sent (encrypted and PKCS #1 formatted) within the `ClientKeyExchange`. An attacker eavesdropping this (encrypted) message can decrypt it later on by abusing the server as a decryption oracle. The format of PKCS #1 v1.5 is given in Table I.

Bleichenbacher's attack utilized a) the fixed structure and b) a known weakness of RSA to *Chosen Ciphertext Attacks* (cf., [6]). The idea is to *blind* the original ciphertext, pass it to the decrypter and finally separate the blinding value. In the following C denotes the ciphertext, P the plaintext, e RSA's public exponent, d the private exponent and n the modulus. RSA encryption is defined as $C \equiv P^e \mod n$ and decryption as $P \equiv C^d \mod n$.

Blinding sketch

1) Choose invertible integer s
2) Blind known ciphertext C: $C' \equiv s^e C \mod n$
3) Let the oracle decrypt C': $P' \equiv C'^d \mod n$
4) Separate s from P':$P \equiv P's^{-1} \mod n$

Depending on the validness of a received PKCS structure the processing at server side differs. In particular, SSL specified to send different error messages for different errors during processing (invalid padding, invalid MAC, ...). With this information one can build an oracle as given in Figure 5.

$$\mathcal{O}_{PKCS}(x) = \begin{cases} \text{true}, & \text{if } x \text{ is PKCS conforming} \\ \text{false}, & \text{otherwise} \end{cases}$$

Figure 5: PKCS oracle

By the use of this oracle it is possible to decrypt the `PreMasterSecret` by continuous blinding the eavesdropped, encrypted message. Based on the oracle's responses the intervals in which possible values may lie can be narrowed, until only a single value is left.

Lesson learned: Bleichenbacher's attack was possible because error messages sent by the server could be used as an oracle revealing partial information about the plaintext. Thus apparently negligible pieces of information such as distinguishable errors, which inform the counterpart on the exact error cause, can be leveraged by an attacker to break security. As a consequence it is necessary to reveal as little information as possible on the internal state of the protocol processing. Especially error messages are a valuable source of information for attackers.

*6) **The rise of timing based attacks:*** Brumley and Boneh outlined in [7] a timing attack on RSA based SSL/TLS. The attack extracts the private key from a target server by observing the timing differences between sending a specially crafted `ClientKeyExchange` message and receiving an `Alert` message inducing an invalid formatted `PreMasterSecret`. Even a relatively small difference in time allows to draw conclusions on the used RSA parameters. Brumley's and Boneh's attack of Brumley and Boneh is only applicable in case of RSA based cipher-suites. Additionally, the attack requires the presence of a fine resolute clock on the attacker's side. The authors were able to successfully attack OpenSSL.

OpenSSL's implementation relies on application of the Chinese Remainder Theorem (CRT) in order to enhance computation. CRT is generally not vulnerable to Paul Kocher's timing attack (cf., [8]), but additionally to the CRT optimization OpenSSL uses optimizations such as

sliding-window exponentiation which in turn heavily relies on Montgomery's reduction algorithm for efficient modulo reduction. Montgomery's algorithm and others require multi-precision multiplication routines. OpenSSL implements two different algorithms to perform such multiplications: Karatsuba algorithm and an unoptimized algorithm. For these algorithms the integer factors are represented as sequences of words of a predefined size. Due to efficiency reasons OpenSSL uses Karatsuba if words with equal number of words are multiplied and the "standard" algorithm otherwise. According to the authors the "normal" algorithm is generally much slower than Karatsuba, resulting in a measurable timing difference. But due to a peculiarity of Montogmery's algorithm (an additional extra reduction in some cases) the optimizations of Montgomery and Karatsuba counteract one another. This prevents a direct timing attack.
Moreover, the authors' attack determines the dominant effect at a specific phase and leverages the differences. The authors define their algorithm as a kind of binary search for the lower prime of the RSA modulus n ($n = pq$).

As a countermeasure the authors suggest, as the most promising solution, the use of RSA blinding during decryption. Blinding uses a a random value r and computes $p = r^e c \mod n$ before decryption ($p:$ plaintext, $c:$ ciphertext, $n:$ RSA modulus, $e:$ public exponent). The original plaintext can be recovered by decrypting and dividing by r: $p \equiv \frac{(r^e c)^d \mod n}{r} \equiv \frac{(r^e p^e)^d \mod n}{r} \equiv \frac{rp \mod n}{r}$.
The attack was significantly improved in 2005 by Aciicmez, Schindler and Koc in [9].

Lesson learned: Brumley and Boneh demonstrated that designers have to take special care on building implementations with nearly equal response times for each conditional branch of message processing.

*7) **Improvements on Bleichenbacher's attack**:* The researchers Klíma, Pokorny and Rosa not only improved Bleichenbacher's attack (cf. II-5) in [10], but were able to defeat a countermeasure against Bleichenbacher's attack.

Breaking the countermeasure A countermeasure against Bleichenbacher's attack is to generate a random `PreMasterSecret` in any kind of failure and continue with the handshake until the verification and decryption of the `Finished` message fails due to different key material (the `PreMasterSecret` differs at client and server side). Additionally, the implementations are encouraged to send no distinguishable error messages. This countermeasure is regarded as best-practice. Moreover, because of a different countermeasure concerning version rollback attacks (cf., II-4) the encrypted data includes not only the `PreMasterSecret`, but also the major and minor version number of the negotiated SSL/TLS version. Implementations should check for equality of the sent and negotiated protocol versions. But in case of version mismatch some implementations again returned distinguishable error messages to the sender (e.g. `decode_error` in case of OpenSSL). It is obvious that an attacker can build a new (bad version) oracle from this, as shown in Figure 6.

$$\mathcal{O}_{BadVersion}(c^*) = \begin{cases} \text{true}, & \text{if version number is valid} \\ \text{false}, & \text{otherwise} \end{cases}$$

Figure 6: Bad Version Oracle

With a new decryption oracle $\mathcal{O}_{BadVersion}$ Klíma, Pokorny and Rosa were able to mount Bleichenbacher's attack, in spite recommended countermeasures are present.

Improving Bleichenbacher's attack on PKCS#1 Additionally to the resurrection of Bleichenbacher's attack the authors could improve the algorithm for better performance. These optimizations included redefinition of interval boundaries for possible PKCS conforming plaintexts. These improvements are based on advanced knowledge gained from the SSL/TLS specification:

- A `PreMasterSecret` is exactly 46 bytes longer
- The `PreMasterSecret` is prefixed with two version bytes
- Padding bytes are unequal to 00
- A PKCS conforming plaintext M_i contains a null-byte separating the padding from the payload data. The length of the padding is known in advance (2 type bytes, $k - 51$ padding bytes, a single null byte as a separator, 2 bytes for the version number and 46 `PreMasterSecret` bytes)

Even tighter adjustment is possible, since the used protocol version is known in advance, revealing another 2 bytes.

Another optimization was made to the original algorithm by introducing a a new method on how to find suitable blinding values. The authors call this the *Beta method*.

As a last optimization Klíma, Pokorny and Rosa suggested to parallelize steps of Bleichenbacher's algorithm, which speeds up the attack.

Lesson learned: Countermeasures against one vulnerability (cf. II-4) may lead to other vulnerabilities.

*8) **ECC based timing attacks**:* At ESORICS[3] 2011 Brumley and Tuveri [11] presented an attack on ECDSA based TLS connections. As to their research only OpenSSL seemed to be vulnerable.

The problem arose from the strict implementation of an algorithm for improving scalar multiplications, which ECC heavily relies on, such as e.g., point multiplication. The implemented algorithm for performing such scalar multiplications, known as the Montgomery power ladder [12] (with improvements by López and Dahab [13]), is timing resistant from a formal point of view, but from implementational point of view contained a timing side-channel. The developers optimized the performance of the algorithm by reducing the repetitions of internal loops, leading to a timing side channel.

[3]https://www.cosic.esat.kuleuven.be/esorics2011/

When arranging the integer of a scalar multiplication (k) in a binary tree (leading zeros stripped) the tree's height is $\lceil \log(k) \rceil - 1$ (-1 since the first bit is expected to be 1, thus ignoring the first level of the tree). This implies that the runtime of the algorithm is $t(\lceil \log(k) \rceil - 1)$, where t denotes some constant time of the algorithm. As a result, timing measurements enabled drawbacks on the multiplier.

Brumley and Tuveri combined this side channel with the lattice attack of Howgrave-Graham and Smart [14] to recover secret keys. For this to work they identified that creating ECDSA signatures relies on scalar multiplication. ECDSA signatures are generated in TLS/SSL when `ECDHE_ECDSA` cipher-suites are used. The authors measured the time between the `ClientHello` message and the arrival of the `ServerKeyExchange` message during the handshake phase. The latter message contains an ECDSA signature over a digest, consisting of relevant parts necessary for further establishment of cryptographic material. As this digital signature can only be created on-the-fly, and not in advance, an adversary is able to measure runtime of the vulnerable scalar multiplication function.

Lesson learned: Side channels may come from unexpected sources.

9) ***Even more improvements on Bleichenbacher's attack:*** In [15] Bardou, Focardi, Kawamoto, Simionato, Steel and Tsay significantly improved Bleichenbacher's attack (cf., II-5) far beyond the previous improvements of Klíma, Pokorny and Rosa (cf., II-7). Bardou et al. fine-tuned the algorithm to perform faster and with lesser oracle queries. Additionally, the authors combined their results with the previous improvements and were able to significantly speed up Bleichenbacher's algorithm.

Lesson learned: Attacks improve and adjust as time goes by. It is necessary to observe research on attacks (even if they are patched yet).

10) ***ECC-based key exchange algorithm confusion attack:*** In [16] Mavrogiannopoulos, Vercauteren, Velichkov and Preneel showed that the key exchange algorithm confusion attack by Wagner and Schneier, discussed in II-3, can also be applied to ECDH. According to the authors, their attack is not feasible yet, due to computational limitations. But, as already discovered with other theoretical only attacks, it may be a question of time when the attack is enhanced to be practical or the resources for computation increase. As discussed by Wagner and Schneier, the main problem remains the lack for a content type field indicating the algorithm type of the contained data - which implicitly indicates how to decode the message.

Lesson learned: See II-3

III. Attacks on the Record and Application Data Protocols

The attacks presented in this section are enabled by weaknesses of the Record or Application Data Protocol and/or the structure of record frames.

A. *Attack discussion*

1) ***MAC does not cover padding length:*** Wagner and Schneier pointed out in [2] that SSL 2.0 contained a major weakness concerning the Message Authentication Code (MAC). The MAC applied by SSL 2.0 only covered data and padding, but left the padding length field unencrypted. This may lead to message integrity compromise.

Lesson learned: Not only since the introduction of padding oracles by Vaudenay (cf., III-A2) each single bit of information should be considered useful for an attacker. Thus, information should be integrity protected and authenticated to keep the attack vector as small as possible.

2) ***Weaknesses through CBC usage:*** Serge Vaudenay introduced a new attack class, padding attacks, and forced the security community to rethink on padding usage in encryption schemes [17].
The attacks described by Vaudenay rely on the fact that block encryption schemes operate on blocks of fixed length, but in practice most plaintexts have to be *padded* to fit the requested length (a multiple of the block length). After padding, the input data is passed to the encryption function, where each plaintext block (of length of the block size) is processed and chained according to the Cipher Block Chaining Mode (CBC) scheme. The CBC mode chains consecutive blocks, so that a subsequent block is influenced by the output of its predecessor. This allows an attacker to directly influence the decryption process by altering the successive blocks. For convenience, Figure 7 gives a short recap of the CBC en-/decryption mode.

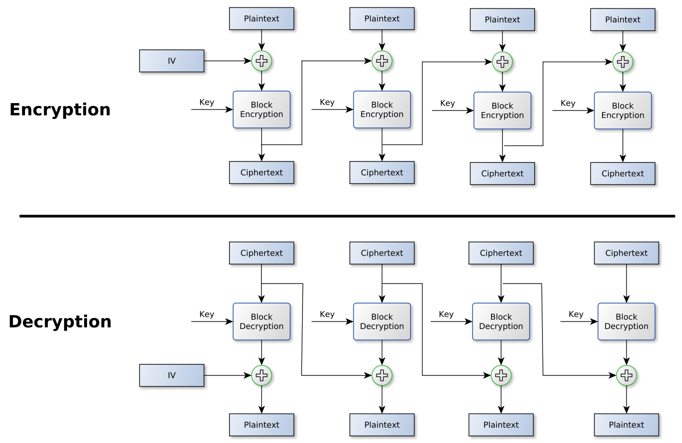

Figure 7: Encryption/Decryption in CBC Mode

Vaudenay builds a decryption oracle out of the receiver's reaction on a ciphertext in case of valid/invalid padding. In the case of SSL/TLS the receiver may send a `decryption_failure` alert, if invalid padding is encountered. The padding oracle is defined in Figure 8.

$$\mathcal{O}_{Padding}(C) = \begin{cases} \text{true}, & \text{if } C \text{ is correctly padded} \\ \text{false}, & \text{otherwise} \end{cases}$$

Figure 8: Padding oracle

By the use of such an oracle and clever changes on the ciphertext an attacker is able to decrypt a ciphertext without knowledge of the key. The optional MAC ensuring message integrity of SSL/TLS does not hinder this attack, since MAC creation takes place before the message is padded. Thus, the padding is not covered by the `MAC`.

Lesson learned: Although the attack is not directly applicable to standard SSL/TLS (since *Fatal* errors immediately invalidate the session and accordingly the key material), it is applicable to DTLS (as discussed later in III-A11). Thus, this theoretical attack should not be ignored. Again, error messages can form the base for decryption oracles.

As a solution, SSL defines equal error messages for padding and decryption errors. But there still remains some room for timing attacks (due to conditional execution paths in the SSL stack). The CBC mode leveraged the attack, so the usage of weak schemes like CBC should be rethought and replaced by more secure, authenticated encryption schemes such as Galois/Counter Mode (GCM) [18].

*3) **Information leakage by the use of compression:*** In [19] Kelsey described an information leak enabled by a side-channel based on compression. This is in absolute contrast to, what the author calls "folk wisdom", that applying compression leads to a security enhanced system. Kelsey showed that it adds little security or, in the worst case, is exactly the other way around, due to the fact that compression reveals information on the plaintext.

Cryptosystems aim at encrypting plaintexts in a way that the resulting ciphertext reveals little to no information on the plaintext. Kelsey observed that by the use of compression a new side-channel arises which could be used to gain hints on the plaintext. Therefor he correlates the output bytes of the compression to the input bytes and makes use of the fact that compression algorithms, when applied to the plaintext, reduce the size of the input data.

Compression algorithms encode redundant patterns in input data to shorter representations and substitute each occurrence with the new representation. Different input strings of the same length are likely to compress to strings of different length. Kelsey used this observation to gain knowledge about the plaintext. Kelsey advices that also timing issues have to be taken into consideration, since compression as an additional step requires additional processing time.

Although this attack was not exploited at that time it lead to the development of Rizzo and Duong's C.R.I.M.E. tool (cf. III-A12) in 2012.

Lesson learned: Kelsey's observation is interesting since it breaks with the "folk wisdom" that applying compression leads to security enhanced systems. Compression may lead to hidden side-channels (compression rate, timing, etc.). It seems that optimizing for performance in accordance with security is hazardous. Performance optimizations should be proved against possible side-channels and skipped or adjusted (which in turn may counterbalance the performance gain).

*4) **Intercepting SSL/TLS protected traffic:*** In [20] Canvel, Hiltgen, Vaudenay and Vuagnoux extended the weaknesses presented by Vaudenay (cf., III-A2) to decrypt a password from an SSL/TLS secured IMAP session.

Canvel et al. suggested three additional attack types based on Vaudenay's observations:

Timing Attacks The authors concluded that a successful MAC verification needs significantly more time compared to a premature abortion caused by an invalid padding. This observation relies on the fact that performing a padding check is less complex than performing cryptographic operations as they are necessary to verify a MAC.

Multi-session Attacks The basic idea of this attack type requires a critical plaintext to be present in each TLS session (such as e.g., a password) and that the corresponding ciphertext is known to the attacker. Due to the nature of security best-practice the corresponding ciphertexts look different every session, since the key material for MAC and encryption changes every session. Therefor, it is advantageous to check if a given ciphertext ends with a specific byte sequence (which should be identical in all sessions) instead of trying to guess the whole plaintext.

Dictionary attacks Leveraged by the previous attack type which checks for a specific byte sequence of the plaintext this attack aims at checking for byte sequences included in a dictionary.

A successful attack against an IMAP server was performed and the password used by the `LOGIN` command could be recovered. As a recommendation the authors propose to change the processing order when encrypting. The MAC should also cover the padding, which implies the order PAD-then-MAC-then-Encrypt.

Lesson learned: It may be advantageous to change the order of processing to PAD-then-MAC-then-Encrypt, but this in turn may leverage other attacks and has to be carefully considered and accurately specified before implementation. The order of processing makes a big difference and should payed special attention.

*5) **Chosen-Plain-text Attacks on SSL:*** Gregory Bard observed in [21] an interesting detail concerning Initialization Vectors (IVs) of SSL messages. As can be seen from Figure 7 every en- and decryption in CBC mode depends on an IV. Every new plaintext (consisting of multiple blocks) should get its own, fresh and independent IV.

The problem with SSL is that, according to the SSL specification, only the IV of the first plaintext is chosen randomly. All subsequent IVs are simply the last block of the previous encrypted plaintext. This is absolutely in contrast to cryptographic best-practice.

Bard observed that an attacker willing to verify a guess if a particular block has a special value and is in possession of an eavesdropped ciphertext can easily check her guess. An attacker could mount an attack if she knows exactly which block is of interest. This knowledge is not as strong as it seems to be, since the strict format of e.g., `https://` requests contains fixed header fields which are known in advance. Additionally, the previous block must be known which is mostly no problem. If the second precondition is fulfilled determining the IV used for this encryption remains an easy task, since it is the last block of the preceding message. Finally, an attacker has to find a way to inject data to the first block of the subsequent message.

The vulnerability was at the same time discovered by Bodo Möller who described the weakness in detail in a series of emails on an IETF mailing-list[4]. In this series Möller described a fix which was later used by the OpenSSL project: Prepending a single record with empty content, padding and MAC, to each message.

As potential fixes for the IV vulnerability Bard recommended the use of pseudo random IVs or dropping CBC and switching to a more secure encryption mode. TLS 1.1 follows the first recommendation by introducing an `IV` field into the `GenericBlockCipher` structure which encapsulates encrypted data.

The practicability of this attack was proven by Bard two years later (cf. III-A6).

Lesson learned: The weakness outlined by Bard is a good example for ignoring security best-practices for the sake of simplicity.

*6) **Chosen-Plain-text Attacks on SSL reloaded:*** The attack by Bard discussed in III-A5 was revisited by Bard in a publication in 2006 [22]. Overall Bard addressed the same topics as before, but provided an attack sketch how to exploit this problem. Bard described a scenario in which an attacker uses a Java applet, executed on the victim's machine, to mount the attack as described in III-A5. As a precondition it is necessary to access a common used SSL tunnel. Bard refers to HTTP-Proxy or SSL based tunneling scenarios.

It is outlined that this scenario does not work if compression is used. Many preconditions have to be fulfilled in order to be able to successfully attack a SSL connection (the author states that it is "challenging but feasible"). However, Bard gives an example for block-wise-adaptive chosen plaintext attacks targeting SSL. Rizzo and Duong proved in III-A8 that Bard's attack scenario is applicable, in a slightly different implementation (by using JavaScript instead of Java applets). The described attack was adopted and adjusted in their B.E.A.S.T. tool (cf. III-A8).

The vulnerability is fixed with TLS 1.1, since it dictates the use of explicit IVs.

Lesson learned: Not only the protocol has to be considered when evaluating security - the interplay between different layers and applications is relevant, too.

*7) **Traffic analysis of TLS:*** George Danezis highlighted in an unpublished manuscript [23] ways how an attacker may use the obvious fact that minimal information, despite the connection is TLS protected, remains unencrypted to analyze and track traffic. In particular, Danezis used unencrypted fields, part of every TLS message, of the TLS Record Header for analysis. Figure 9 shows an encrypted TLS Record containing unencrypted header fields and encrypted payload.

```
struct {
  ContentType type;
  ProtocolVersion version;
  uint16 length;
  select (CipherSpec.cipher_type) {
    case stream: GenericStreamCipher;
    case block: GenericBlockCipher;
  } fragment;
} TLSCiphertext;
```

Figure 9: TLS Cipher-text Record - *Source: RFC 2246 [4]*

As can be seen the fields `type`, `version` and `length` remain always unencrypted - even in an encrypted record. The fields are necessary for correct record decoding. In [2] the authors already criticized the presence of such unauthenticated and unencrypted fields. RFC 2246 is also aware of this information leak and advices to take care of this:

> "Any protocol designed for use over TLS must be carefully designed to deal with all possible attacks against it. Note that because the type and length of a record are not protected by encryption, care should be take to minimize the value of traffic analysis of these values."
>
> \- *Source: RFC 2246 [4]*

Danezis identified several information leaks introduced by these unencrypted fields:

- Requests to different URLs may differ in length which results in different sized TLS records.
- Responses to requests may also differ in size, which again yields to different sized TLS records.
- Different structured documents may lead to a predictable behavior of the client's application. For example a browser is normally gathers all images of

[4]http://www.openssl.org/~bodo/tls-cbc.txt

a website - causing different requests and different responses.
- Content on public sites is visible to everyone, an attacker may link content (e.g., by size) to site content.

Moreover, an attacker could also actively influence the victim's behavior and gain information about what she is doing (without knowledge of the encrypted content) by providing specially crafted documents with particular and distinguishable content lengths, structures, URLs or external resources. The traffic analysis is not only limited to TLS, but at least it reveals a weakness that can be exploited.

The author provides some hints on how the surface of the attack can be limited, but the practicability of the recommended measures remains questionable.

- URL padding - all URLs are of equal length
- Content padding - all content is of equal size
- Contribution padding - all submitted data is of equal size
- Structure padding - all sites rely on an equal structure (e.g., sites with equal number of external resources aso.)

This flaw was also discussed by Wagner and Schneier in [2], but in a limited manner. Wagner and Schneier credited Bennet Yee as the first one describing traffic analysis on SSL. As a countermeasure Wagner and Schneier suggested support for random length padding not only for block cipher mode, but for all cipher modes. The use of random length padding should be mandatory.

The feasibility of Danezis attack was proven by Chen, Wang, Wang and Zhang in [24] who investigated multiple real world web applications for these kind of side-channels.

Lesson learned: Clever and creative attackers may find ways to use every obtainable part of information for further attacks. More sophisticated attacks are possible if fields are left unauthenticated. Protocol designers and developers should be aware of this fact and sparely disclose *any* information. This also applies to error messages. If in doubt only use a single error message for all occurring errors. One may question herself what the counterpart could really do if she knows exactly what went wrong (bad padding, invalid MAC, etc.) If she is a valid user she would simply try again or give up, independent from a detailed error message. On the other hand an attacker is grateful for every peace of information, even a single bit.

8) ***Practical IV Chaining vulnerability:*** Rizzo and Duong presented in [25] a tool called B.E.A.S.T. that is able to decrypt HTTPS traffic (e.g., cookies). The authors implemented and extended ideas of Bard [21], [22], Möller and Dai [5]. The combination of CBC mode applied to block ciphers (cf., Figure 7) and predictable IVs (cf., Chapter III-A5) enabled *guessing* of plaintext blocks and verify the validness. To verify a guess an oracle $\mathcal{O}_{Guess}$ is required returning true in case of a successful guess and false otherwise, Figure 10 illustrates such an oracle.

$$\mathcal{O}_{Guess}(P^*) = \begin{cases} \text{true}, & \text{if guess } P^* \text{ equals the plaintext} \\ \text{false}, & \text{otherwise} \end{cases}$$

Figure 10: Guessing oracle

Rizzo and Duong created such an oracle based on the precondition that the IVs used by CBC (the last encryption block of the preceding encryption) are known to the attacker.

To adopt the technique to SSL/TLS and decrypt ciphertexts byte-wise the authors propose a new kind of attack named *block-wise chosen-boundary attack*. It requires that an attacker is able to move a message before encryption in its block boundaries. This means an attacker may prepend a message with arbitrary data in such a way that it is split into multiple blocks of block-size of the cipher. Based on this, it is possible to split a message of full block-size into two blocks: the first one consisting of arbitrary data and the first byte of the original block, the second block consisting of the remaining bytes and a single free byte. So prefixing a message with an attacker defined amount of data shifts the message (if necessary into a new block). An attacker is absolutely free to prepend any data of her choice and length. An example is given in Figure 11.

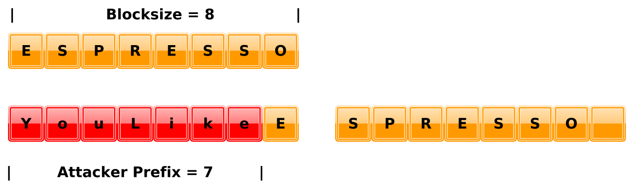

Figure 11: Example boundary shifting

Full message decryption

To decrypt a full message the attacker adjusts the amount of random prefix data so that the next *unknown* byte is always the last byte in a block. This means in detail that the message is shifted in such a way, that the scenario illustrated in Figure 11 applies to the next unknown byte. The unknown byte becomes the last and only byte of a single block unknown to the attacker. Finally, this leads to a byte by byte message decryption.

To apply the previously discussed algorithm Rizzo and Duong chose HTTPS as SSL/TLS protected application (protocol). The goal is to decrypt cookies sent by a browser to a server for authentication. For a successful attack some preconditions have to be met:

1) An attacker is able to sniff the traffic between victim and server
2) An attacker can force a victim to perform HTTP(S) `POST` requests to the server
3) An attacker can control the document path of the `POST` request

[5]http://www.weidai.com/ssh2-attack.txt

4) An attacker forces the victim to visit a website under control of the attacker to inject exploit code

Due to this massive vulnerability, migration to TLS Version 1.1 has been recommended since by IETF.

Lesson learned: This attack shows that vulnerabilities considered to be theoretical only can turn in to practice if skilled attackers put effort in the implementation and fine-tune the algorithm.

9) ***Short message collisions and busting the length hiding feature of SSL/TLS:*** In [26] Paterson, Ristenpart and Shrimpton outlined an attack related to the MAC-then-PAD-then-Encrypt scheme in combination with short messages. In particular their attack is applicable if all parts of a message (message, padding, MAC) fit into a single block of the cipher's block-size. Under special preconditions the authors described the creation of multiple ciphertexts leading to the same plaintext message.

Lesson learned: The surface for this attack is limited, since the precondition (message, padding and MAC have to fit into a single block) is quite strong. But it revealed that in some special cases the preconditions are met.

10) ***Message distinguishing:*** Paterson et al. extended in [26] the attack described in III-A9 enabling an attacker to distinguish between two messages. The authors sketch how to distinguish whether the encrypted message contains `YES` or `NO`. The attack is based on clever modification of the eavesdropped ciphertext so that it either passes the processing or leads to an error message. Based on the outcome (error/no error) it is possible to determine which content was send. The attack works only if the possible contents are of different, short length. At least, the attack remains unexploitable due to the fact that it is only possible for 80 bit truncated MACs. While in versions prior to TLS 1.2 the use of truncated MACs was possible, 1.2 restricts it. Anyway, truncated MACs are possible in TLS 1.2, too, by the use of protocol extensions.

Lesson learned: See III-A9.

11) ***Breaking DTLS:*** In [27] AlFardan and Paterson applied Vaudenay's attack (cf., III-A2) to DTLS. DTLS is a slightly different version of regular TLS adjusted to unreliable transport protocols, such as UDP. There are two major differences when compared to standard TLS:

1) Complete absence of `Alert` messages
2) Messages causing protocol errors (bad padding, invalid MAC, ...) are simply dropped, instead of causing a connection abort invalidating the session keys

These adjustments are advantageous, as well as disadvantageous at the same time. Vaudenay's attack may work on DTLS since bad messages do not cause session invalidation. But with the lack of error messages the oracles introduced by Vaudenay can not be used without adjustment. The attacker gets no feedback whether the modified messages contained a valid padding or not. The authors adjusted Vaudenay's algorithms by using a timing oracle arising from different processing branches with unequal time consumption.

AlFardan and Paterson covered the OpenSSL and GnuTLS implementations, both vulnerable to a timing oracle enhanced version of Vaudenay's attack. The timing oracle of OpenSSL arises from a padding dependent MAC check. In case of correctly padded data the MAC is checked, where in case of an incorrect padding the verification is skipped. This behavior leads to a measurable timing difference allowing drawbacks on the validity of the padding. GnuTLS in contrast contains a timing side-channel during the decryption process where sanity checks are performed prior to decryption.
While the side-channel of OpenSSL is a consequence of missing countermeasures, even GnuTLS - which implements all recommended countermeasures - is vulnerable. When timing differences are too little, reliable timing differences are gained by sending multiple copies leading to an accumulation of timing differences. According to the authors, it was necessary for the proof of concept attack to disable the protocol's *anti-replay* option, which is enabled by default.

Lesson learned: The authors recommend to specification authors that defining standards by only defining differences to previous standards should be avoided (DTLS is a sub-specification of TLS).

12) ***Practical compression based attacks:*** In September 2012 Juliano Rizzo and Thai Duong presented the *C.R.I.M.E.* attack tool. *C.R.I.M.E.* targets HTTPS and is able to decrypt traffic, enabling cookie stealing and session takeover. It exploits a known vulnerability caused by the use of message compression discovered by Kelsey in 2002 [19].

The attack relies on plaintext size reduction caused by compression. Given that an attacker is able to obtain the (real) size of an encrypted, compressed plaintext (without padding) she might decrypt parts of the ciphertext by observing the differences in size. An attacker does neither know the plaintext, nor the compressed plaintext, but is able to observe the length of the ciphertext. So basically, she prefixes the secret with guessed subsequences and observes if it leads to compression (by observing the resulting ciphertext length). A decreased ciphertext length implies redundancy, so it is very likely that the guessed, prefixed subsequence caused redundancy in the plaintext. It can be concluded that with higher redundancy of input data the better the compression ratio, resulting in length decreased output. This implies that a guess, having something in common with the secret, will have a higher compression rate leading to a shorter output. When such an output is encountered the attacker knows that the guess has something in common with the secret.

As preconditions an attacker must be able to sniff the network and attract a user to visit a website under the attacker's control. Further on, both - client and server - have to use compression enabled SSL/TLS. The website controlled by the attacker delivers *C.R.I.M.E.* as JavaScript to the victim's browser. Once the script is running an attacker starts guessing - e.g., the right cookie value - and forcing the victim to transport the guess concatenated with the desired cookie as an SSL/TLS message (at least encrypted).

Lesson learned: This attack is just another example for mainly theoretical attacks that were found to be practical by skilled attackers.

IV. Attacks on the PKI

Checking the validity of X.509 certificates is an area full of problems, which cannot be covered in an overview on TLS attacks (especially usability aspects). However we want to point to some specific problems that are technically very closely related to TLS.

A. Attack discussion

1) ***Weak cryptographic primitives lead to colliding certificates:*** Lenstra, Wang and de Weger described in 2005 how an attacker can create two valid certificates with equal hash values by computing MD5 collisions [28]. With colliding hash values it is possible to impersonate clients or servers - attacks of this kind render very hard to detect Mitm attacks possible.

The practicality of the attack was demonstrated in 2008 by Sotirov et al. [6] who were able, through clever interaction between certificate requests from a legal CA and a massively parallel search for MD5 collisions, to create a valid CA certificate for TLS. With the help of this certificate they could have issued TLS server certificates for any domain name, which would have been accepted by any user agent. (To make sure that their results cannot be misused, they however chose to create an expired CA certificate.)

Lesson learned: As long as user agents accept MD5 based certificates, the attack surface still exists, even if CAs stop issuing such certificates. Thus MD5 must be disabled for certificate checking in all user agents. Moreover, weak algorithms may lead to complete breach of the security when targeting the trust anchors.

2) ***Weaknesses in X.509 certificate constraint checking:*** In 2008, US hacker Matthew Rosenfeld, better known as Moxie Marlinspike, published a vulnerability report [29] concerning the certificate basic constraint validation of Microsoft's Internet Explorer. The Internet Explorer did not check if certificates were allowed to sign sub-certificates (to be more technical, if the certificate is in possession of a `CA:TRUE` flag). Any valid certificate, signed by a valid CA, was allowed to issue sub certificates for *any* domain.

[6]http://www.win.tue.nl/hashclash/rogue-ca/

Normally, issuing subsequent certificates is only valid if the `CA:TRUE` flag is present in the own certificate. Otherwise, certificate inspectors should reject certificates issued by (intermediate) CAs with insufficient rights.

The tool `sslsniff`[7] provides a proof of concept implementation with the attacker acting as Mitm, issuing certificates for a requested domain on the fly. After a successful attack the Mitm impersonates the requested server. The basic functionality of the tool is illustrated in Figure 12.

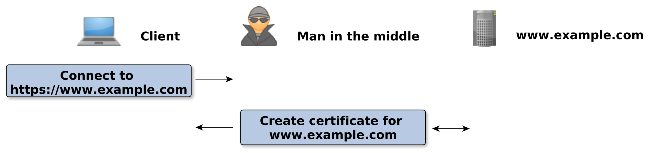
Client
Man in the middle
www.example.com
Connect to https://www.example.com
Create certificate for www.example.com

Figure 12: Basic mode of operation of `sslSniff` - Impersonating a server

Lesson learned: The attack relies on a specific implementation bug and has been fixed. However, certificate validation is a critical step.

3) ***Attacks on Certificate Issuer Application Logic:*** Attacks on the PKI by exploiting implementational bugs on CA side were demonstrated by Moxie Marlinspike in [30], who was able to trick the CA's issuance logic by using specially crafted domain strings. Marlinspike gained valid certificates for arbitrary domains, issued by trusted CAs. As a prerequisite for the attack it is necessary that the issuer only checks if the requester is the legitimate owner of the TLD. An owner of a TLD may request for certificates issued for arbitrary sub-domains (e.g., the owner of `example.com` may also request certificates for `subdomain.example.com`).

Marlinspike makes use of the encoding of X.509 - ASN1. ASN1 supports multiple String formats, all leading to slightly different PASCAL String representation conventions. PASCAL represents Strings length-prefixed (a leading length byte followed by the according number of bytes, each representing a single character). An example is given in Figure 13.

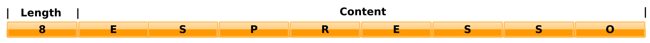

Figure 13: String representation - PASCAL convention

In contrast C Strings are stored in `NULL`-terminated representation. Here, the character bytes are followed by a `NULL` byte signalizing the end of the String. This implies that `NULL` bytes are prohibited within regular Strings, since they would lead to premature String termination. This scheme is illustrated in Figure 14.

[7]http://www.thoughtcrime.org/software/sslsniff/

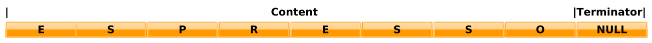

Figure 14: String representation - C convention

This knowledge prepares the way for the NULL-Prefix attack: A sample domain name which could be used in a Certificate Signing Request (CSR) is the following www.targetToAttack.com\0.example.com, assuming that the attacker is the owner of example.com. The attack works, because the CA logic only checks the TLD (example.com). The leading NULL-byte (\0) is valid because of ASN1's length-prefixed representation (where NULL-bytes within the payload String are valid). When the prepared domain String is presented to common application logic (mostly written in languages representing Strings NULL-terminated) the String is prematurely terminated. As a result only the String afore the NULL byte (www.targetToAttack.com) is being validated.

A specialization of the attack are wild-card certificates. The asterisk (*) can be used to create certificates, valid - if successfully signed by a trusted CA - for **any** domain (e.g., ⋆\0.example.com).

Lesson learned: Certification authorities should be prepared to deal with different encodings and security issues related to this.

*4) **Attacking the PKI**:* Marlinspike described in [31] an attack that aims at interfering the infrastructure to revoke certificates. By the use of the Online Certificate Status Protocol (OCSP) a client application can check the revocation status of a certificate. OCSP responds to query with a responseStatus (cf. Figure 15).

```
OCSPResponse ::= SEQUENCE {
    responseStatus
    responseBytes   (optional)
}
OCSPResponseStatus ::= ENUMERATED {
    successful
    malformedRequest
    internalError
    tryLater
    sigRequired
    unauthorized
}
```

Figure 15: OCSP Response data type - *based on Source: [32]*

The response structure contains a major design flaw: The responseBytes field is optional, but it is the only one containing a digital signature. This implies that the responseStatus field is not protected by a digital signature. Not all of the OCSPResponseStatus types require the presence of responseBytes (containing a digital signature). Especially tryLater does not force signed responseBytes to be present.

An attacker acting as Mitm could respond to every query with tryLater. Due to lack for a signature the client has no chance to detect the spoofed response. Thereby, a victim is not able to query the revocation status of a certificate.

Lesson learned: If real-time checks on a PKI are required, unsigned responses should lead to a halt in protocol execution.

*5) **Conquest of a Certification Authority**:* At 15 March 2011 the *Comodo CA Ltd.* [8] Certification Authority (CA) was successfully compromised [33]. An attacker used a reseller account to issue 9 certificates for popular domains. Except rumors, the purpose of the attack remains unclear.
Soon after attack discovery the concerned certificates have been revoked. Due to previously discussed weaknesses affecting the revocation infrastructure (cf. IV-A4) it could be possible that these certificates remain valid to parties with out-of-date or missing certificate revocation lists.
It is crucial to note that the attacker did not compromise Comodo's infrastructure directly (key material and servers were not compromised at any time). Moreover, valid credentials of a reseller were used to sign CSRs.

Lesson learned: Certification authorities have to protect their critical infrastructure with strong security mechanisms. Certificate issuing should not rely on weak authentication mechanisms like username/password only.

*6) **Conquest of another Certification Authority**:* Soon after the attack on Comodo, a Dutch Certification Authority - *DigiNotar* - was completely compromised by an attacker [34]. In contrast to the Comodo impact, the attacker was able to gain control over the DigiNotar infrastructure. The attack discovery was eased by Google's Chrome web browser who complained about mismatching certificates for Google-owned domains. The browser stores hard coded copies of the genuine certificates for Google and thus was able to detect bogus certificates.

Lesson learned: Beside the lesson learned from IV-A5, it can be seen that non-cryptographic security mechanisms like malware and intrusion detection must be present in CA systems.

*7) **Attacks on non-browser based certificate validation**:* At CCS 2012, Georgiev et al. [35] uncovered that widespread, common used libraries for SSL/TLS suffer from vulnerable certificate validation implementations. The authors revealed weaknesses in the source code of OpenSSL, GnuTLS, JSSE, ApacheHttpClient, Weberknecht, cURL, PHP, Python and applications build upon or with these products. The authors examined the root causes for the

[8]http://www.comodo.com

bugs and were able to exploit most of the vulnerabilities. As major causes for these problems bad and misleading API specifications, lacking interest for security concerns (even by banking applications!) and the absence of essential validation routines were identified. Especially OpenSSL and GnuTLS provide confusing APIs, leaving important tasks influencing security to the API consumer. Even worse, the API of cURL was attested to be ”almost perversely bad”.

Especially, the following security tasks and robustness of the libraries' code responsible for these tasks are considered:

- Certificate chaining and verification
- Host name verification
- Certificate revocation checks
- X.509 Extension handling and processing

Exploiting these vulnerabilities may lead to successful Mitm and impersonation attacks.

Lesson learned: The study of Georgiev et al. gives an example on the importance of clean, simple and well documented APIs.

V. Various Attacks

This section deals with various attacks not suitable for any of the previous attack sections.

A. Attack discussion

1) ***Random number prediction:*** In January 1996, Goldberg and Wagner published an article [36] on the quality of random numbers used for SSL connections by the Netscape Browser. The authors gained access to the application's Source Code by decompiling it and identified striking weaknesses in the algorithm responsible for random number generation. The entropy of the algorithm relied completely on few, predictable values:

- Current time
- Current process id
- Process id of the parent process

Moreover, the authors showed that due to export limitations the entropy of key material is extremely limited - enabling brute force attacks.

Lesson learned: The problem of weak random number generators is not a specific problem of SSL or TLS but reminds that a good (pseudo) random number generator (PRNG) is crucial for cryptographic purposes (cf. V-A2).

2) ***Weakened security through bad random numbers:*** In 2008 Luciano Bello [37] observed during code review that the PRNG of Debian-specific OpenSSL was predictable starting from version 0.9.8c-1, Sep 17 2006 until 0.9.8c-4, May 13 2008, due to an implementation bug. A Debian-specific patch removed two very important lines in the `libssl` source code responsible for providing adequate entropy[9] (cf. Figure 16).

```
MD_Update(&m, buf , j ) ;
        [ .. ]
MD_Update(&m, buf , j ) ; /* purify complains */
```

Figure 16: Lines commented out leading to a serious bug - *Source: Debian optimized OpenSSL Source Code*

This allowed a brute force attack, since the key space was significantly limited without these code lines.

Lesson learned: Developers should comment security critical parts of source code, explain exactly the code's intention and highlight the consequences when altered (e.g., in Java developers could decide to introduce a Java Doc keyword `SECURITY` that marks security related comments or additionally introduce a `@Security(critical=true)` annotation). Beyond this, test cases targeting the critical code lines and returning errors if removed or altered should be provided.

3) ***Denial of Service enabled by Exceptions:*** In [38] Zhao et al. provided an attack on the TLS handshake which leads to an immediate connection shutdown and can thus be used for a Denial of Service (DoS) attack. The authors exploited two previously discussed weaknesses to mount successful attacks.

- The first attack targets the Alert protocol of TLS and makes use of the fact that, due to yet missing completed cryptographic primitives negotiation during the handshake phase, all `Alert` messages remain strictly unauthenticated and thus spoof-able. This enables an obvious, but effective attack: Spoofing *Fatal* `Alert` messages which cause immediate connection shutdowns
- The second attack simply confuses a communication partner by sending either misleading or replayed messages or responding with wrong messages according to the expected handshake flow.

Lesson learned: Even obvious and self-evident weaknesses (such as DoS vulnerabilities) have to be discussed and focus of research.

4) ***Renegotiation flaw:*** Ray and Dispensa discovered in [39] a serious flaw induced by the renegotiation feature of TLS. The flaw enables an attacker to inject data into a running connection without destroying the session. A server would accept the data, believing its origin is the client. This could lead to abuse of established sessions - e.g., an attacker

[9]http://anonscm.debian.org/viewvc/pkg-openssl/openssl/trunk/rand/md_rand.c?p2=%2Fopenssl%2Ftrunk%2Frand%2Fmd_rand.c&p1=openssl%2Ftrunk%2Frand%2Fmd_rand.c&r1=141&r2=140&view=diff&pathrev=141

could impersonate a legitimate victim currently logged in to a web application.

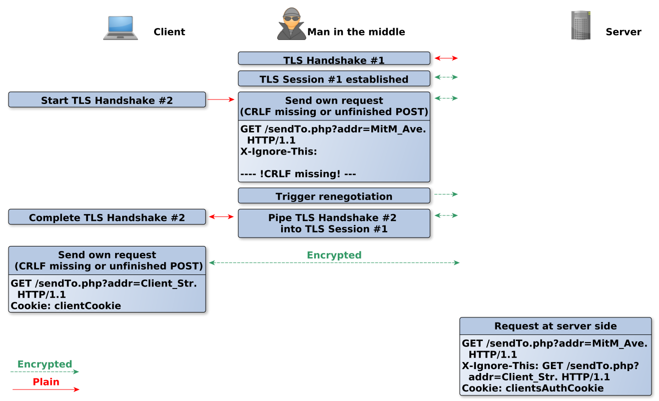
Client
Man in the middle
Server
TLS Handshake #1
TLS Session #1 established
Start TLS Handshake #2
Send own request
(CRLF missing or unfinished POST)
GET /sendTo.php?addr=MitM_Ave.
HTTP/1.1
X-Ignore-This:
---- !CRLF missing! ---
Trigger renegotiation
Complete TLS Handshake #2
Pipe TLS Handshake #2
into TLS Session #1
Send own request
(CRLF missing or unfinished POST)
GET /sendTo.php?addr=Client_Str.
HTTP/1.1
Cookie: clientCookie
Encrypted
Request at server side
GET /sendTo.php?addr=MitM_Ave.
HTTP/1.1
X-Ignore-This: GET /sendTo.php?
addr=Client_Str. HTTP/1.1
Cookie: clientsAuthCookie
Encrypted
Plain

Figure 17: Example scenario for the renegotiation attack - *based on Source: [39]*

Figure 17 shows how an attacker acting as Mitm may abuse the renegotiation feature to inject arbitrary data into a TLS connection. In the example scenario an attacker concatenates 2 `GET` requests to send a gift to `MitM Ave.` instead of `Client Str.`. This requires an authentication based on cookies. The client attaches its authentication cookie to the own request, but the attacker previously left its own request unfinished by omitting a final `CRLF` and adding a non-existent header field without value. At server side the following happens: since the last request is not finished yet, the following data (the original client request) is concatenated with the pending data. The `GET` request of the client becomes the value of the non-existent header field and is therefor ignored by the server. The authentication cookie in the next line remains valid (due to the `\n` after the `GET` line). The result is a valid request with a valid cookie. It is important to note, that the attacker does not get the authentication cookie in plaintext, but her request is constructed to be concatenated on server side in a way that the first line of the client's request gets dropped (by using the header trick) - the attacker is at no time able to decrypt traffic.

Anil Kurmus proved the flaw to be practical by stealing confidential data through a slightly different attack on Twitter[10] (he used an unfinished `POST` request) enabled by the renegotiation flaw[11].

There are multiple ways to fix the problem. E.g., Eric Rescorla proposed a special TLS extension [40] that fixes this flaw.

Lesson learned: When switching security contexts it needs to be guaranteed that there is no pending data left.

[10]http://www.twitter.com

[11]http://www.securegoose.org/2009/11/tls-renegotiation-vulnerability-cve.html

*5) **Disabling SSL/TLS at a higher layer:*** In February 2009, Moxie Marlinspike released the `sslstrip`[12] tool which disables SSL/TLS at a higher layer. As a precondition it is necessary for an attacker to act as Mitm. To disable the security layer the tool sends `HTTP 301 - permanent` redirection responses and replaces any occurrence of `https://` with `http://`. This causes the client to move to the redirected page and communicate unencrypted and unauthenticated (when the stripping succeeds and the client does not notice that she is being fooled). Finally, the attacker opens a fresh session to the (requested) server and passes-through or alters any client and server data. The attack sketch is outlined in Figure 18.

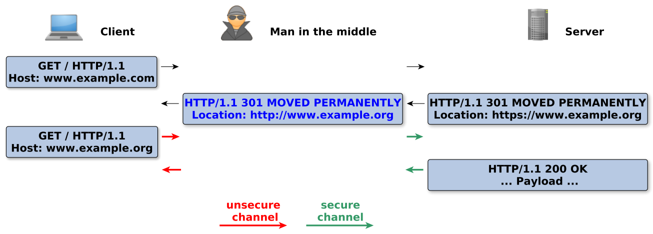

Figure 18: Example scenario for a SSL stripping attack

Lesson learned: Usability is the keyword here: In principle, a user should be able to detect the redirection to `http://` if it is properly visualized, e.g. by using red colored address bars for *all* non-SSL/TLS connections.

*6) **Computational Denial of Service:*** In 2011, the German Hacker Group *The Hackers Choice* released a tool called `THC-SSL-DoS`[13], which creates huge load on servers by overwhelming the target with SSL/TLS handshake requests. Boosting system load is done by establishing new connections or using renegotiation. Assuming that the majority of computation during a handshake is done by the server the attack creates more system load on the server than on the own device - leading to a DoS. The server is forced to continuously recompute random numbers and keys.

Lesson learned: In DoS attacks, cryptography is part of the problem, not a solution.

VI. Conclusion

When summarizing the attacks and lessons learned some guidelines for protocol designers and implementors emerge.

Since theoretical only weaknesses turned out to be adoptable in practice every outlined weakness should be taken seriously. Unexploitability may not last forever.

From a developer's point of view reliable cryptographic primitives (e.g., good and strong random numbers) and side-channel hardened algorithms turned out to be of importance.

[12]http://www.thoughtcrime.org/software/sslstrip/

[13]http://www.thc.org/thc-ssl-dos/

Side-channel resistance evolved to a general, since side-channels of any kind (information leaks, timing differences, etc.) appear at different layers in different situations. Moreover, verbosity (e.g., error messages), normally a honorable intention, provides attackers valuable information on internal states and processing. Thus, as little information as possible should be provided. Another critical aspect in matters of honorable intersessions are own improvements on specifications for additional value (e.g., performance gains), because they may lead to unintended vulnerabilities. Therefore, a specification should be implemented exactly as is, without modifications and tweaks. Marking critical parts of source code with precise and clear comments highlighting potential risks could prevent disadvantageous modifications. This should especially be best-practice when fixing vulnerabilities to prevent bug reintroduction. Fixing software can be an error prone task, since fixes have to patch not only the current vulnerability occurrence, but all parts of the code that may suffer from the same weakness. This often requires to think about the interplay of different layers and applications. As a final remark for developers, it can be concluded that one should always be alarmed when working on security critical components. Security is a necessity, not a necessary evil.

Protocol designers, in turn, should be as precise as possible when defining a specification. Specifications with room for interpretation and implementational freedom can lead to misunderstanding or unawareness of security related concerns. Especially where processing order makes a huge difference highlighting the risks of process reordering is crucial. Risks related to divergence on the specification need to be clarified and highlighted. Thus, the specification of minimum requirements and mandatory preconditions is required. This implies strong and context-free message structures with little to no room for misinterpretation. Sensitive fields have to be authenticated and, if necessary, confidential fields should be encrypted. The communication parties need to authenticate what was sent and received at any time, to obviate deliberate or non deliberate modifications during transport. Finally, adhere that flexibility mostly means complexity which should be avoided.

DoS attacks remain a future problem that has to be focus of research and discussion. New directions and means to lower the surface emerged to be of increased relevance.

VII. Acknowledgments

This work was partially funded by the *Sec*^{2} project of the German Federal Ministry of Education and Research (BMBF, FKZ: 01BY1030).

References

[1] E. Rescorla, *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2001.

[2] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 protocol," *The Second USENIX Workshop on Electronic Commerce Proceedings*, pp. 29–40, 1996.

[3] M. Bellare and P. Rogaway, "Entity authentication and key distribution," 1994, pp. 232–249.

[4] T. Dierks and C. Allen, "The TLS Protocol Version 1.0," RFC 2246 (Proposed Standard), Internet Engineering Task Force, Jan. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2246.txt

[5] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1," in *Advances in Cryptology — CRYPTO '98*, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed. Springer Berlin / Heidelberg, 1998, vol. 1462, pp. 1–12.

[6] G. Davida, "Chosen Signature Cryptanalysis of the RSA (MIT)Public Key Cryptosystem," Tech. Rep., 1982.

[7] D. Brumley and D. Boneh, "Remote timing attacks are practical," in *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, ser. SSYM'03. Berkeley, CA, USA: USENIX Association, Jun. 2003, pp. 1–1.

[8] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. London, UK, UK: Springer-Verlag, Aug. 1996, pp. 104–113.

[9] O. Aciicmez, W. Schindler, and C. Koc, "Improving Brumley and Boneh timing attack on unprotected SSL implementations," in *Proceedings of the 12th ACM conference on Computer and communications security*. ACM, Nov. 2005, pp. 139–146.

[10] V. Klíma, O. Pokorný, and T. Rosa, "Attacking RSA-Based Sessions in SSL/TLS," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, ser. Lecture Notes in Computer Science, C. Walter, C. Koc, and C. Paar, Eds. Springer Berlin / Heidelberg, Sep. 2003, vol. 2779, pp. 426–440.

[11] B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *Computer Security - ESORICS 2011*, ser. Lecture Notes in Computer Science, V. Atluri and C. Diaz, Eds. Springer Berlin / Heidelberg, Sep. 2011, vol. 6879, pp. 355–371.

[12] P. L. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization," *MC*, no. 48, 1987.

[13] J. López and R. Dahab, "Fast Multiplication on Elliptic Curves Over GF(2m) without precomputation," in *Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, C. Koc and C. Paar, Eds. Springer Berlin / Heidelberg, 1999, vol. 1717, pp. 724–724.

[14] N. A. Howgrave-Graham and N. P. Smart, "Lattice Attacks on Digital Signature Schemes," *Designs, Codes and Cryptography*, vol. 23, pp. 283–290, 2001.

[15] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay, "Efficient Padding Oracle Attacks on Cryptographic Hardware," INRIA, Rapport de recherche RR-7944, Apr. 2012. [Online]. Available: http://hal.inria.fr/hal-00691958

[16] N. Mavrogiannopoulos, F. Vercauteren, V. Velichkov, and B. Preneel, "A Cross-Protocol Attack on the TLS Protocol," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. ACM, Oct. 2012, pp. 62–72.

[17] S. Vaudenay, "Security Flaws Induced by CBC Padding — Applications to SSL, IPSEC, WTLS..." in *Advances in Cryptology — EUROCRYPT 2002*, ser. Lecture Notes in Computer Science, L. Knudsen, Ed. Springer Berlin / Heidelberg, Apr. 2002, vol. 2332, pp. 534–545.

[18] D. A. McGrew and J. Viega, "The Galois/counter mode of operation (GCM)," 2005.

[19] J. Kelsey, "Compression and information leakage of plaintext," in *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2365. Springer, Nov. 2002, pp. 263–276.

[20] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," in *Advances in Cryptology - CRYPTO 2003*, ser. Lecture Notes in Computer Science, D. Boneh, Ed. Springer Berlin / Heidelberg, Aug. 2003, vol. 2729, pp. 583–599.

[21] G. V. Bard, "The vulnerability of ssl to chosen plaintext attack." *IACR Cryptology ePrint Archive*, vol. 2004, p. 111, May 2004.

[22] ——, "A Challenging But Feasible Blockwise-Adaptive Chosen-Plaintext Attack on SSL," in *SECRYPT 2006, Proceedings of the International Conference on Security and Cryptography*. INSTICC Press, Aug. 2006, pp. 7–10.

[23] G. Danezis, "Traffic Analysis of the HTTP Protocol over TLS," unpublished manuscript.

[24] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow," in *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, ser. SP '10. IEEE Computer Society, May 2010, pp. 191–206.

[25] J. Rizzo and T. Duong, "Here Come The XOR Ninjas," May 2011.

[26] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: attacks and proofs for the TLS record protocol," in *Proceedings of the 17th international conference on The Theory and Application of Cryptology and Information Security*, ser. ASIACRYPT'11. Berlin, Heidelberg: Springer-Verlag, Dec. 2011, pp. 372–389.

[27] N. AlFardan and K. Paterson, "Plaintext-Recovery Attacks Against Datagram TLS," in *Network and Distributed System Security Symposium (NDSS 2012)*, Feb. 2012.

[28] A. Lenstra, X. Wang, and B. de Weger, "Colliding X.509 Certificates," Cryptology ePrint Archive, Report 2005/067, Mar. 2005.

[29] M. Rosenfeld, "Internet Explorer SSL Vulnerability," May 2008. [Online]. Available: http://www.thoughtcrime.org/ie-ssl-chain.txt

[30] ——, "Null Prefix Attacks Against SSL/TLS Certificates," Feb. 2009. [Online]. Available: http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf

[31] ——, "Defeating OCSP With The Character '3'," Jul. 2009. [Online]. Available: http://www.thoughtcrime.org/papers/ocsp-attack.pdf

[32] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP," RFC 2560 (Proposed Standard), Internet Engineering Task Force, Jun. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2560.txt

[33] Comodo CA Ltd., "Comodo Report of Incident - Comodo detected and thwarted an intrusion on 26-MAR-2011," Tech. Rep., Mar. 2011.

[34] Fox-IT, "Black Tulip - Report of the investigation into the DigiNotar Certificate Authority breach," Tech. Rep., Aug. 2012.

[35] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," in *ACM Conference on Computer and Communications Security*, 2012.

[36] Goldberg and Wagner, "Randomness and the Netscape browser," *Dr. Dobb's Journal*, Jan. 1996.

[37] F. Weimer, "DSA-1571-1 openssl – predictable random number generator," Network Working Group, May 2008. [Online]. Available: http://lists.debian.org/debian-security-announce/2008/msg00152.html

[38] Y. Zhao, S. Vemuri, J. Chen, Y. Chen, H. Zhou, and Z. Fu, "Exception triggered DoS attacks on wireless networks," in *Proceedings of the 2009 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009*, Jun. 2009, pp. 13–22.

[39] M. Ray and S. Dispensa, "Renegotiating TLS," PhoneFactor, Inc., Tech. Rep., Nov. 2009.

[40] E. Rescorla, M. Ray, S. Dispensa, and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension," RFC 2246 (Proposed Standard), Network Working Group, Nov. 2009. [Online]. Available: http://tools.ietf.org/id/draft-rescorla-tls-renegotiation-01.txt