Postcards from the Post-HTTP World: Amplification of HTTPS Vulnerabilities in the Web Ecosystem

Stefano Calzavara
Ca' Foscari Univ.
calzavara@dais.unive.it

Riccardo Focardi
Ca' Foscari Univ.
& Cryptosense
focardi@unive.it

Matus Nemec
Ca' Foscari Univ.
& Masaryk Univ.
matus.nemec@unive.it

Alvise Rabitti
Ca' Foscari Univ.
alvise.rabitti@unive.it

Marco Squarcina
TU Wien
marco.squarcina@tuwien.ac.at

***Abstract*—HTTPS aims at securing communication over the Web by providing a cryptographic protection layer that ensures the confidentiality and integrity of communication and enables client/server authentication. However, HTTPS is based on the SSL/TLS protocol suites that have been shown to be vulnerable to various attacks in the years. This has required fixes and mitigations both in the servers and in the browsers, producing a complicated mixture of protocol versions and implementations in the wild, which makes it unclear which attacks are still effective on the modern Web and what is their import on web application security. In this paper, we present the first systematic quantitative evaluation of web application insecurity due to cryptographic vulnerabilities. We specify attack conditions against TLS using attack trees and we crawl the Alexa Top 10k to assess the import of these issues on *page integrity*, *authentication credentials* and *web tracking*. Our results show that the security of a consistent number of websites is severely harmed by cryptographic weaknesses that, in many cases, are due to external or related-domain hosts. This empirically, yet systematically demonstrates how a relatively limited number of exploitable HTTPS vulnerabilities are amplified by the complexity of the web ecosystem.**

I. Introduction

The HTTP protocol is the central building block of the Web, yet it does not natively provide any confidentiality or integrity guarantee. HTTPS protects network communication against eavesdropping and tampering by running HTTP on top of cryptographic protocols like Secure Socket Layer (SSL) and its successor Transport Layer Security (TLS), which allow for the establishment of encrypted bidirectional communication channels. Besides confidentiality and integrity, HTTPS also ensures authentication, because clients and servers may prove their identity by presenting certificates signed by a trusted certification authority. HTTPS has been increasingly recognized as a cornerstone of web application security over time and it is routinely employed by more and more websites, to the point that the average volume of encrypted web traffic has surpassed the average volume of unencrypted traffic according to data from Mozilla [36]. It is plausible to believe that, in a near future, HTTP will be (almost) entirely replaced by HTTPS, thanks to initiatives like Let's Encrypt and the actions taken by major browser vendors to mark HTTP as 'not secure' [73].

Security experts know well that the adoption of HTTPS is necessary for web application security, but not sufficient. Web applications can be attacked at many different layers, for example on session management [17]. Moreover, the correct deployment of HTTPS itself is far from straightforward [52]. For instance, bad security practices like the lack of adoption of HTTP Strict Transport Security (HSTS) may allow attackers to sidestep HTTPS and completely void its security guarantees. But even when HTTPS is up and running, cryptographic flaws in SSL/TLS may undermine its intended security expectations. Many attacks against SSL/TLS have been found, allowing for information disclosure via side-channels or fully compromising the cryptographic keys used to protect communication [1], [4], [9], [11], [14], [59]. These attacks are not merely theoretical: they have been shown to be effective in the wild and open data from Qualys [64] suggest that many servers are vulnerable to them. Several papers have also discussed the results of similar data collections [14], [28], [39], [84], [85].

Despite this availability of data, however, previous analyses provide only a very limited picture of how much cryptographic weaknesses in HTTPS implementations harm the security of the current Web. First, these studies are based on large-scale detections of server-side vulnerabilities, but they do not provide a thorough account of their *exploitability on modern clients*. Many known vulnerabilities such as Bleichenbacher's padding oracle attack on PKCS #1 v1.5 RSA encryption [13] or various padding oracle attacks on Cipher Block Chaining (CBC) mode ciphers [3], [59], [88] rely on specific assumptions on both the client and the server to be exploited, such as that the TLS peers will negotiate a specific ciphersuite like RSA key exchange or use a symmetric cipher in CBC-mode, respectively. Hence, the mere existence of a vulnerability does not necessarily imply the possibility to attack a TLS connection between an up-to-date client and a vulnerable server, since all modern browsers implement various mitigations that prevent many of the known TLS attacks. Moreover, attacks against TLS at the transport layer may drastically differ in terms of their *impact at the application layer*: for example, the POODLE-TLS attack [78] can gradually leak a secret, but it requires the attacker to force the browser into re-sending the secret many times. Thus, the attack can leak a session cookie by injecting requests from a page under the attacker's control, but not a password that is inserted by the user on a secure login page and only sent once.

In this paper we present the first systematic quantitative evaluation of web application insecurity due to cryptographic HTTPS vulnerabilities. The analysis relies on a characteri-

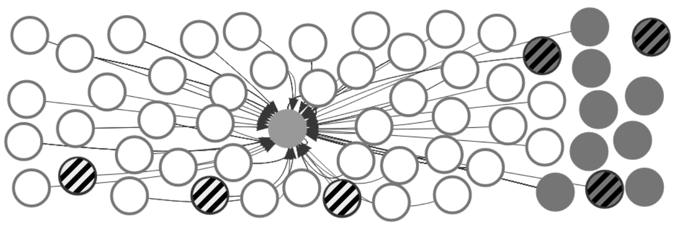

Fig. 1. An anonymized top Alexa website (central circle) and its sub-domains (gray, on the right) and dependencies (white, with arrows). The website is entirely deployed over HTTPS, but becomes insecure due to three vulnerable sub-domains and three vulnerable dependencies (striped circles).

zation of TLS vulnerabilities in terms of *attack trees* [74] capturing the conditions for the various attacks to be enabled and on a crawl of the top 10,000 websites from Alexa supporting HTTPS, including all their dependencies (hosts from which sub-resources are included) and sub-domains. Crawling dependencies and sub-domains is of ultimate importance, as secure websites might be broken by importing sub-resources or sending domain cookies over vulnerable TLS channels. The complexity of the web ecosystem, in fact, *amplifies* the effect of TLS vulnerabilities, as illustrated in Figure 1. Our results are disquieting:

- 898 websites are fully compromisable, allowing for script injection, while 977 websites present low integrity pages that the attacker can tamper with. Fully compromisable sites include e-commerce sites such as alibaba.com, e-banking services such as deutsche-bank.de and major websites such as myspace.com and verizon.com. 660 out of the 898 compromisable websites (73.5%) include external scripts from vulnerable hosts, thus empirically demonstrating that the complexity of web applications enormously amplifies their attack surface;
- 10% of the detected login forms have confidentiality issues, which may enable password theft. 412 websites may be subject to cookie theft, exposing to session hijacking, and 543 websites are subject to cookie integrity attacks. Interestingly, we found that more than 20% of the analyzed domain cookies can be potentially leaked, suggesting that the organization of web applications as related sub-domains amplifies their attack surface and needs to be carefully analyzed;
- 142 websites include content from vulnerable hosts of the popular tracker PubMatic and thus expose users to profiling attacks. Remarkably, this privacy attack can be amplified by the previous finding on compromisable websites, so as to affect up to 968 websites. This shows once more that attacks against TLS on external resources may expose otherwise secure websites to severe threats.

One of the original aspects of our work is that all of the presented attacks on web applications are exclusively due to practical TLS vulnerabilities that are enabled on the server and not prevented by modern browsers, thus potentially exploitable. Our findings show that a limited number of practical TLS vulnerabilities are amplified by the web ecosystem and have a huge practical impact on otherwise secure websites that depend on or are related to the vulnerable hosts. We found vulnerabilities in popular, security-conscious websites. For example, because of TLS weaknesses in related hosts or dependencies, it is possible to break password confidentiality on myspace.com, session security on yandex.com and cookie integrity on live.com. We responsibly disclosed our findings to the interested websites.

Contributions and paper structure: In this paper, we make the following contributions:

1) we review existing cryptographic attacks against TLS, identifying those which are still effective on modern clients. We then characterize such attacks in terms of attack trees, which identify conditions to break the confidentiality and/or integrity properties of the TLS protocol. To the best of our knowledge, this is the most systematic model of such attacks presented in the literature – with a special focus on their practical impact – and can serve other security researchers working in the area (Section III);
2) we build an analysis platform which implements the checks defined by the attack trees and we run it on the homepages of the top 10,000 websites of the Alexa ranking supporting HTTPS. As part of this data collection process, we also scan 90,816 hosts which either (i) store sub-resources included in the crawled pages or (ii) are sub-domains of the websites. These hosts have a major import on the security of the crawled websites, which we precisely assess (Section IV);
3) we rigorously identify a number of severe web application attacks enabled by vulnerable TLS implementations and we run automated checks for them on the collected data. We focus on three different aspects of web application security: *page integrity* (Section V), *authentication credentials* (Section VI) and *web tracking* (Section VII). This list is not meant to be exhaustive, yet it is rich enough to cover important security implications of existing cryptographic flaws of TLS on major websites.

Finally, Section II provides background on TLS and Section VIII provides our closing perspective, discussing related work, ethical issues and limitations of our study.

II. Background on TLS

In this section, we describe TLS 1.0, 1.1 and 1.2. Readers who are already familiar with TLS can safely skip this section. We do not discuss TLS 1.3 [66], as there are no known attacks against it due to the removal of vulnerable cryptographic constructions used in previous protocol versions [66, Section 1.2]. Notice that version 1.3 is not yet widely supported in the wild: only 5.2% of hosts in our scan supported some draft version of TLS 1.3 (the final version was not yet published at the time of the scan). Moreover, we do not discuss certificate-based client authentication as it is rarely adopted on the Web.

The TLS protocol consists of the following sub-protocols:

Record Protocol carries the data, that are optionally encrypted and authenticated, of the application data protocol and the remaining TLS sub-protocols;

Handshake Protocol negotiates cryptographic keys and authenticates the server;

Change Cipher Spec Protocol signals to the other peer that the subsequent records will be encrypted and authenticated under the negotiated keys;

Alert Protocol signals status changes, with warnings and terminating fatal alerts, following e.g., decryption errors.

A. The Handshake Protocol

We describe in detail the Handshake Protocol, as it is the one responsible for agreeing on the cryptographic algorithms and keys used to protect messages and for authenticating the server. As such, it constitutes a clearly sensitive target for network attackers. The Handshake Protocol is an authenticated key exchange protocol. The peers negotiate the TLS version and the cryptographic algorithms (ciphersuites) for key exchange, server authentication, and Record Protocol protection.

The client initiates the handshake with a `ClientHello` message, that includes the highest supported TLS protocol version, a random nonce for key derivation, the session identifier, the list of supported ciphersuites, the supported compression methods (usually empty, as TLS compression is deprecated for security reasons), and optional TLS extensions.

The server responds with a `ServerHello` message with the lower between its highest supported protocol version and the client's version, a random nonce, the session identifier, the selected ciphersuite and compression method, and selected extensions (a subset of those offered by the client). The server should follow an ordering of the ciphersuites, ideally selecting the most secure ciphersuite offered by the client. If there are no supported algorithms in common, the server responds with a handshake failure alert.

The server also sends its X.509 certificate in the `Certificate` message, that links its identity to its public key. Depending on the selected ciphersuite, it may send a `ServerKeyExchange` message contributing to the key material. The client sends the `ClientKeyExchange` message with its key material. The shared key material is called the Premaster Secret (PMS) and is used together with the exchanged random nonces to compute the Master Secret, which is in turn used to derive the session keys for the Record Protocol. Once the Master Secret is shared, the peers run the Change Cipher Spec Protocol and start protecting their messages.

Finally, the client and the server mutually exchange the `Finished` message containing a transcript of the handshake. If the peers received different messages, possibly due to tampering by an attacker, their transcripts will differ. Since the communication is encrypted and authenticated with the session keys at this point, the attacker cannot tamper with the transcripts. The PMS is shared using a public key that is tied to the identity of the server, hence the server authenticates by using the PMS to compute the session keys.

B. Ciphersuites

A key ingredient of the Handshake Protocol is the negotiation of the cryptographic mechanisms in the ciphersuite. The most common algorithms are:

Key exchange: how to share the PMS:

- **RSA key exchange:** the client randomly generates a PMS, encrypts it with the RSA public key of the server obtained from the server's trusted certificate, and sends it in the `ClientKeyExchange`;
- **Static Diffie-Hellman key exchange – (EC)DH:** the DH parameters are defined either on a prime field (DH) or on an elliptic curve (ECDH). The client generates a random (EC)DH key and sends the public part in the `ClientKeyExchange`. The public key of the server is contained within its certificate. The shared DH secret is used as the PMS;
- **Ephemeral Diffie-Hellman key exchange – (EC)DHE:** similar to the previous case, however the client and the server generate fresh (ephemeral) (EC)DHE keys and send them in the `Client-` and `Server-KeyExchange` messages, respectively. The server must sign its message with a private key corresponding to its certificate. DHE uses RSA or DSA [60], ECDHE uses RSA or ECDSA [60].

Confidentiality and integrity: how messages sent over the Record Protocol are protected:

- **Block ciphers in AEAD mode:** Authenticated Encryption with Associated Data (AEAD) combines encryption and authentication in a single primitive. Examples are AES in the GCM or CCM mode of operation;
- **Block ciphers in CBC mode with MAC:** combination of CBC mode of operation of a symmetric block cipher with Keyed-hash Message Authentication Code (HMAC) for authentication. The order of operations is MAC-then-Pad-then-Encrypt. For example, AES, Camellia, Triple-DES or DES in CBC mode combined with HMAC based on SHA-2, SHA-1 or MD5;
- **Stream cipher with MAC:** for example, ChaCha20 with Poly1305 (that combine into an AEAD primitive) or RC4 with HMAC based on SHA-1 or MD5.

III. Attack Trees for TLS Security

We describe notable cryptographic attacks against TLS and divide them by their impact on confidentiality and integrity of the communication. We discuss how the attacks are mitigated by client configuration and specific countermeasures, focusing on attacks that fall under our threat model. See Appendix A for out of scope attacks and Appendix B for more details on the attacks introduced in this section.

A. Threat Model

We assume an active network attacker able to add, remove or modify messages sent between a client and a server. The attacker also controls a malicious website, say at `evil.com`, which is navigated by the attacked client. By means of the website, the attacker can inject scripts in the client from an attacker-controlled origin, which is relevant for a subset of the attacks. However, the attacker can neither break the Same Origin Policy (SOP)[1] nor exploit any bug in the browser. We assume the attacker cannot exploit timing side-channels, since the feasibility of such attacks is generally hard to assess.

The client is a modern browser that (i) supports TLS 1.0, 1.1, and 1.2 with key establishment based on ECDH and AEAD ciphersuites (cf. MozillaWiki [89] for the purpose of "Modern" compatibility); (ii) does not support SSLv3 or lower, does not offer weak or anonymous ciphersuites (such as DES, RC4 and EXPORT ciphers, or suites without encryption or authentication) and enforces a minimal key size of cryptographic algorithms; (iii) correctly handles certificate validation and rejects certificates with weak algorithms. All the major browsers released in the last two years satisfy these assumptions, starting from Firefox 44, Chrome 48, IE 11 on Windows 7, Edge, Opera 35, Safari 10, and Android 6.0.

B. Review of Known Attacks against TLS

Protocol version downgrade: A TLS server should respond to a `ClientHello` with the offered version of the protocol, or the highest it supports. However, some legacy servers simply drop connections with unsupported TLS versions, without offering an alternative. Thus, browsers may repeat the handshake with a lower protocol version. An attacker in the middle could drop `ClientHello` messages until the client downgrades to an older, vulnerable version of the protocol. To prevent this attack, the client attaches a fake ciphersuite to repeated handshake attempts, as defined in RFC 7507 [58], indicating that the handshake did not use the highest client-supported TLS version. The presence of that ciphersuite in a `ClientHello`, with a TLS version that is lower than the highest supported by the server, reveals a potential attack and should be treated as such by the server. Safari, Internet Explorer, and Edge fall back to TLS 1.0. Only Safari appends the ciphersuite. Firefox, Chrome, and Opera, instead, removed insecure fallback entirely when the `ClientHello` messages are dropped.

RSA decryption oracles: In the RSA key exchange, the client chooses the PMS and sends it to the server, encrypted under the server's public RSA key. TLS uses the padding scheme defined in PKCS #1 v1.5 [47], which is known to be vulnerable to a padding oracle attack [13]. The attack is possible when the server provides a padding oracle, i.e., when it behaves differently when decrypting messages that have invalid paddings. An attacker can multiply a ciphertext to create a new ciphertext (RSA is malleable), until a new correctly padded message is forged. When this happens, the attacker learns partial information about the plaintext message and the process can be iterated until the key exchange is fully decrypted. The original attack was proposed by Bleichenbacher in 1998 [13] and requires on the order of million connections to decrypt a ciphertext. The attack was later improved [5], [46], [50], [57], especially in the presence of an oracle that does not strictly enforce the padding scheme [5], to require on the order of tens of thousands of messages. In our analysis, we only consider such strong version of the oracle as exploitable.

RSA signature oracles: A very fast decryption oracle can be used to compute RSA signatures. Hence, even without the knowledge of the private key, an attacker can impersonate the server in the (EC)DHE exchange with such oracle. The attack applies to all TLS versions up to TLS 1.2. However, the signature generation using a Bleichenbacher's oracle is even slower than the decryption [14]. Therefore, the attacker would prefer the decryption of RSA key exchange, if supported by the targeted host. Interestingly, a signature oracle makes it possible to impersonate the target server even with other certificates valid for that target (such as wildcard certificates).

Advanced RSA padding oracles – DROWN and key reuse: When a server is vulnerable to the decryption oracle, all servers that use the same RSA key for key encryption (e.g., due to using the same certificate) are vulnerable to the decryption of the key exchange, even if they do not provide the oracle directly. Furthermore, TLS can be enabled for other application level protocols than HTTPS, such as email (SMTP, POP3, and IMAP with STARTTLS, or SMTPS, IMAPS, POP3S). The attack surface of the DROWN attack [4] was in fact amplified by the possibility of using vulnerable servers supporting SSLv2 in order to break servers running newer protocol versions. DROWN uses the fact that SSLv2 provides the padding oracle in combination with weak export grade ciphersuites and specific OpenSSL bugs. The attack comes in two variants, General and Special, requiring respectively about 8 hours and less than a minute to complete. Thus, only the Special case is suitable for Man In The Middle (MITM) attacks. Not all handshakes are vulnerable: 1 out of 900, for the General case, and 1 out of 260 for the Special case.

RSA padding oracle countermeasures: TLS 1.0 [25], 1.1 [26], and 1.2 [27] introduced countermeasures to remove the padding oracle, instead of replacing the padding scheme. However, the ROBOT attack [14] has shown that a surprisingly high number of implementations in the wild still present padding oracles that can be used to decrypt RSA encrypted messages. The attacks are partially mitigated by the support for Perfect Forward Secrecy, typically by preferring the elliptic curve Diffie-Hellman key establishment with ephemeral private keys (ECDHE) over the RSA key exchange on the server side. Since all modern web browsers support ECDHE cipher suites [89], the RSA key exchange will be voluntarily negotiated only with servers that prefer it due to lack of ECDHE support or bad configuration. It would be thus recommended to completely disable RSA encryption at the server side [14].

CBC mode padding oracles: TLS uses the CBC mode of operation of a symmetric block cipher with MAC-then-Pad-then-Encrypt scheme for record-level encryption. Since the padding is not covered by the MAC, changing the padding does not change the integrity of the message, and could enable a padding oracle vulnerability. A class of vulnerabilities of

[1]https://developer.mozilla.org/docs/Web/Security/Same-origin_policy

the MAC-then-Pad-then-Encrypt construction was described by Vaudenay [88] and Canvel et al. [21]. The attacks are based on distinguishing failures due to bad padding and due to failed integrity check. In TLS, the server should issue the same response in both situations, however there are buggy implementations (e.g., [79]) that produce different errors. The POODLE attack [59] leverages the above padding oracle problem in combination with the fact that SSLv3 (and some flawed TLS implementations) only checks the last byte of padding. Since a padding error ends in a termination of the session, the attacker must be able to force the client to open a new session every time she wants to make a guess. Furthermore, the client must repeat the target secret s in every connection, e.g., when s is a secret cookie attached to every HTTPS request. All CBC attacks can be mitigated in TLS 1.2 by supporting either AEAD ciphersuites or stream ciphers that do not require padding, on both servers and clients (as in modern browsers). TLS version downgrades must also be mitigated, to prevent a downgrade to a version that only supports CBC-mode ciphers.

Heartbleed: Due to memory management problems in server implementations, an attacker could reveal the long-term private keys of the server, thus allowing a full impersonation of the server [83], [33].

C. Insecure Channels

To understand the import of cryptographic flaws of TLS on web application security, it is useful to categorize known cryptographic attacks in terms of the security properties they break. We propose three categories of insecure channels:

Leaky: a channel established with servers vulnerable to confidentiality attacks, which give the attacker the ability to decrypt all the network traffic (Section III-D);

Tainted: a channel susceptible to Man In The Middle (MITM) attacks, which give the attacker the ability to decrypt and arbitrarily modify all the network traffic (Section III-E). Tainted channels are also leaky;

Partially leaky: a channel exposing side-channels which give the attacker the ability to disclose selected (small) secrets over time. These channels typically rely on a *secret repetition* assumption, because the attacker abuses the exchange of repeated messages containing the secret on the vulnerable channel (Section III-F). Leaky and tainted channels also qualify as partially leaky.

In the rest of this section, we precisely characterize how we mapped existing cryptographic attacks against TLS to the proposed channel categories in terms of attack trees.

D. Leaky Channels

Channels are *leaky* when established with servers vulnerable to attacks that fully compromise confidentiality. The attacker tries to obtain the PMS to learn the session keys, giving her the ability to decrypt all the captured network traffic.

Figure 2 shows the attack tree of conditions that enable the attacker to learn the session keys. The main goal is listed on the first line. Each goal or sub-goal may have alternative ways of reaching it (marked as logical OR '|') or it may require several sub-goals to be valid at once (marked as logical AND '&'). Sub-goals are differentiated from their parent goal by increased indentation. Leaves, i.e., goals without sub-goals, evaluate to True or False based on a concrete test (e.g., for the presence of a vulnerability), a detected server configuration, or are the result of a stand-alone, separate tree. If the entire tree evaluates to True, the host suffers from an exploitable vulnerability that can facilitate the main goal.

```
GOAL Learn the session keys (allows decryption)
| 1 Decrypt RSA key exchange offline
  & 1 RSA key exchange is used
    | 1 RSA key exchange is preferred in the
        highest supported version of TLS
    | 2 Downgrade is possible to a version of TLS
        where RSA key exchange is preferred
  & 2 RSA decryption oracle (DROWN or Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
```

Fig. 2. Attack tree for leaky channels

```
GOAL Potential MITM (decryption and modification)
| 1 Force RSA key exchange by modifying ClientHello
    and decrypt it before the handshake times out
  & 1 RSA key exchange support in any TLS version
  & 2 Fast RSA decryption oracle (Special DROWN or
      Strong Bleichenbacher's oracle) available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
| 2 Learn the session keys of a long lived session
  & 1 Learn the session keys (Figure 2)
  & 2 Client resumes the session
    | 1 Session resumption with tickets
    | 2 Session resumption with session IDs
| 3 Forge an RSA signature in the key establishment
  & 1 Fast RSA signature oracle (Strong
      Bleichenbacher's oracle) is available on:
    | 1 This host
    | 2 Another host with the same certificate
    | 3 Another host with the same public RSA key
    | 4 A host with a certificate where the Subject
        Alternative Names (SAN) match this host
  & 2 The same RSA key is used for RSA key exchange
      and RSA signature in ECDHE key establishment
| 4 Private key leak due to the Heartbleed bug
```

Fig. 3. Attack tree for tainted channels

The attacker may obtain the PMS by decrypting the key exchange (1). The parties must use RSA key exchange (1.1). Hence, the client must support it and the server must prefer it either in the highest version of TLS supported by both parties (1.1.1), or in any other commonly supported version, if protocol version downgrade is not properly mitigated (1.1.2). The attacker decrypts the RSA key exchange (1.2) either using Strong Bleichenbacher's oracle [14] or with the DROWN attack [4]. The oracle could be present on the target host directly (1.2.1), or on a different host that uses the same certificate (1.2.2) or at least the same RSA key (1.2.3).

E. Tainted Channels

Channels are *tainted* if the attacker can mount a MITM attack that gives her the ability to decrypt and modify all the traffic between the server and the client. Hence, tainted channels are also leaky. The attacker must learn the PMS of an active session or she must influence its value and successfully impersonate the server. The attack tree is shown in Figure 3 and described below.

The attacker can force the use of RSA key exchange by modifying the `ClientHello` sent to the server to only contain such ciphersuites (1). Naturally, the server must support such ciphersuite (1.1). The modification leads to different handshake transcripts, hence the decryption of the key exchange must be performed very fast, in order to generate valid `Finished` messages before the peers time out. Hence, the attacker needs access to a fast instantiation of Strong Bleichenbacher's oracle [14] or to a server vulnerable to the Special variant of the DROWN attack [4] (1.2). The authors of the ROBOT attack [14] estimate that it should be feasible to decrypt the key exchange fast enough (in a few seconds) if the attacker can parallelize the requests across multiple servers of the attacker and the target. An analysis of such parallel attack was done by Ronen et al. [69].

Alternatively, the attacker may gain more time to obtain the session keys, if they are long lived (minutes to hours) (2). She captures an RSA key exchange and decrypts it offline (2.1), through the techniques of Section III-D (Figure 2) as she cannot modify the initial `ClientHello` at will. She then intercepts a resumed session with full MITM capabilities (2.2). Server may support session resumption without server-side state (2.2.1) [71] or with server-side state (2.2.2) [27].

Under some conditions, a very efficient RSA decryption oracle can be used to forge signatures (3). The oracle can be found on a variety of hosts $(3.1.1 - 3.1.3)$. Additionally, a host can be attacked using a certificate that it neither uses nor shares an RSA key with, if the host appears on the certificate's list of Subject Alternative Names (SAN) (3.1.4). The certificate's RSA key used for signing (EC)DHE parameters must be the same as the RSA key used for RSA key exchange by a server with a decryption oracle (3.2).

Finally, the attacker might obtain the private key of the server due to the Heartbleed memory disclosure bug (4) [83]. For ethical reasons, we did not attempt to extract the private keys when we detected Heartbleed, yet it was reliably shown possible [45].

F. Partially Leaky Channels

Channels are partially leaky if they allow for a partial confidentiality compromise of secrets sent by the client to the server. Leaky and tainted channels are also partially leaky. The conditions are described by the attack tree in Figure 4. To exploit a CBC padding oracle (1), the attacker must force repeated requests containing the secret (*secret repetition*) and she is required to partially control the plaintext sent by the client to a vulnerable server, e.g., by modifying the URL in the header of the request. We check the server for the presence of two CBC padding oracle types (as explained in Section III-B). They are instantiated as the TLS version of the POODLE attack [78], [59] (1.1) due to incorrect padding checks (1.1.1) and as a buggy implementation [79] providing a Vaudenay CBC padding oracle [88] (1.2) when using hardware accelerated AES (AES-NI) in certain versions of OpenSSL (1.2.1). Both attack types require the server to choose a vulnerable ciphersuite (1.1.2, 1.2.2). It could be chosen by the server in the highest TLS version (1.1.2.1, 1.2.2.1) or following a protocol version downgrade (1.1.2.2, 1.2.2.2).

```
GOAL Partial decryption of messages sent by Client
| 1 CBC padding oracle on the server
  | 1 POODLE-TLS padding oracle
    & 1 Server checks TLS padding as in SSLv3
    & 2 Any vulnerable CBC mode ciphersuite is used
      | 1 A CBC mode ciphersuite is preferred
          in the highest supported version of TLS
      | 2 Downgrade is possible to a version of TLS
          where a CBC mode ciphersuite is preferred
  | 2 CBC padding oracle - OpenSSL AES-NI bug
    & 1 Server is vulnerable to CVE-2016-2107
    & 2 A ciphersuite with AES in CBC mode is used
      | 1 AES in CBC mode is preferred in the
          highest supported TLS version
      | 2 Downgrade is possible to a TLS version
          where AES in CBC mode is preferred
```

Fig. 4. Attack tree for partially leaky channels

IV. Experimental Setup

We developed an analysis platform to identify exploitable cryptographic weaknesses in TLS implementations and estimate their import on web application security. The platform employs a crawler to perform a vulnerability scan of the target website, testing also hosts which either store sub-resources included by the homepage or belong to related domains. Confidentiality and integrity threats are identified by matching the relevant conditions of the attack trees introduced in Section III against the output of existing analysis tools.

A. Analysis Platform

The analysis platform performs the following steps: (i) access the website, such as `example.com`, by instrumenting Headless Chrome with Puppeteer;[2] (ii) collect the DOM of the page at `example.com`, along with its set of cookies and the hosts serving sub-resources (such as scripts, images, stylesheets and fonts) included by the page; (iii) enumerate the sub-domains of `example.com` by querying the Certificate Transparency[3] logs and by testing for the existence of common sub-domains, such as `mail.example.com`; (iv) run existing analysis tools to identify cryptographic vulnerabilities on the target website and on all the hosts collected in the previous steps; (v) map the output of the tools to the conditions of the attack trees to find exploitable vulnerabilities.

[2]https://github.com/GoogleChrome/puppeteer
[3]https://www.certificate-transparency.org/

The analysis tools include testssl.sh,[4] TLS-Attacker [80] and the nmap plugin for Special DROWN,[5] which combined provide enough information. For ethical reasons, we did not perform any aggressive testing for the presence of oracles other than the checks run by these tools, e.g., we did not evaluate the performance of servers with respect to the number of oracle queries they can answer in a short time. Still, if some untested conditions have been considered realistic in the literature, e.g., the performance of a Strong Bleichenbacher's Oracle for online decryption or for signature computation [14], we report the vulnerability as exploitable.

B. Data Collection and Findings

We used our analysis platform to collect data from the Alexa top 1M list retrieved on July 20, 2018. We scanned sequential batches of websites up to collecting 10,000 websites served over HTTPS. Their sub-resources and related domains added up to 90,816 more hosts that underwent a vulnerability analysis, completed at the beginning of August 2018.

Our tool reported exploitable TLS vulnerabilities in 5,574 hosts (5.5%). 4,818 hosts allow for the establishment of tainted channels, which is the most severe security threat. 733 hosts allow for the establishment of leaky channels, while 912 allow for partially leaky channels. The majority of vulnerabilities is due to the 20 years old Bleichenbacher's attack [13] and its newest improvement ROBOT [14]. Only 6.5% of the scanned hosts actually prefer RSA key exchange in their highest supported TLS version, yet 76.9% hosts support it, presumably to maintain backward compatibility with old clients. More than 90% of servers support a key exchange that provides Perfect Forward Secrecy. Hence, the majority of the exploitable hosts could be secured by stopping the support for RSA key exhange. We provide a breakdown of the identified insecure channels in Table I and we comment it below.

Leaky channels: The connections to 733 hosts could be decrypted using ROBOT or DROWN after the attacker captured the traffic – goal (1) of Figure 2. 727 hosts preferred the RSA key exchange $(1.1.1)$, hence no action would be necessary to make the peers negotiate RSA. Only on 6 hosts the attacker would need to use the protocol version downgrade to force the usage of RSA key exchange $(1.1.2)$ instead of Diffie-Hellman (DH). We found 136 hosts vulnerable to ROBOT that used ECDHE in their highest protocol version and properly implemented protocol version downgrade mitigation, showing the importance of the countermeasure. Out of the 733 vulnerable hosts, 592 hosts were directly exploitable $(1.2.1)$, while 141 were only exploitable due to sharing a certificate $(1.2.2)$ or an RSA key $(1.2.3)$ with a vulnerable host. Hence, a conventional tool that only checks the host directly for the presence of ROBOT would not detect confidentiality problems on 19% of the exploitable hosts.

Tainted channels: In total, 4,818 hosts made connections over tainted channels due to MITM attacks (Figure 3). 615 hosts were exploitable due to the compromise of a resumed session (2), where the attacker can decrypt the key exchange over a longer period. 1,877 additional hosts were susceptible to online RSA key exchange decryption attacks (1). The attack was also possible for the previously mentioned 615 hosts, without relying on the client to resume the session (2.2), yet requiring a faster computation (1.2). When a decryption oracle is available on a host, each certificate that uses the same RSA key for signatures could be used to impersonate all the hosts that appear in its Subject Alternative Name extension (SAN) (3). We found 2,279 such hosts, that could not be impersonated with a less demanding version of the MITM attack: (1) or (2). It is worth noticing that only 1,893 hosts in our scan had a strong ROBOT oracle, yet the number of exploitable servers due to ROBOT is much higher. This shows that the sharing of certificates and RSA public keys, as well as the list of hostnames in the SAN extension, should be kept minimal. Luckily, only 47 hosts were vulnerable to Heartbleed (4). When a private RSA key is extracted in this way, the attacker can repeatedly impersonate the host without its involvement.

Partially leaky channels: Exploitable partially leaky channels (Figure 4) were found on 912 hosts. Out of the 816 hosts with an exploitable POODLE-TLS padding oracle (1.1), 797 hosts preferred the vulnerable ciphersuite $(1.1.2.1)$ and additional 19 hosts could be exploited after being downgraded to an older version of TLS due to a lack of protection from downgrades $(1.1.2.2)$. Out of the 96 hosts with an exploitable OpenSSL AES-NI padding oracle (1.2), only 20 hosts were vulnerable in the preferred TLS version $(1.2.2.1)$ and additional 76 hosts could be exploited after an unmitigated version downgrade $(1.2.2.2)$. Other 68 hosts have been found affected by POODLE-TLS and 2 exposed OpenSSL AES-NI padding oracle, yet a modern browser would negotiate a more secure cipher making the vulnerabilities non-exploitable.

C. Roadmap

The presence of so many insecure channels is concerning, but their actual import on web application security is unclear. In the rest of the paper, we investigate and quantify this delicate point by focusing on selected aspects of web application security. Since we are interested in cryptographic attacks against HTTPS, we stipulate that every time we refer to *pages / channels* we implicitly refer to HTTPS pages / channels, unless otherwise specified. Attacks enabled by the (partial) adoption of HTTP are out of the scope of this study.

V. Page Integrity

In this section, we describe a number of attacks enabled by the presence of tainted channels, whose security import ranges from content injection to SOP bypasses.

A. Security Analysis

If a web page is received from a tainted channel, the attacker may be able to arbitrarily corrupt its contents, thus completely undermining its integrity guarantees. Moreover, even if the page was received from an untainted channel, the subsequent

[4]https://github.com/drwetter/testssl.sh

[5]https://nmap.org/nsedoc/scripts/sslv2-drown.html

TABLE I
OVERVIEW OF THE DETECTED INSECURE CHANNELS

Insecure channel	*Attack*	*Attack tree reference*		*Vulnerable hosts*
Leaky	Decrypt RSA key exchange offline	(1)	Figure 2	733
Tainted	Force RSA key exchange and decrypt it online	(1)	Figure 3	1,877
	Learn the session keys of a long lived session	(2)		615
	Forge an RSA signature in the key establishment	(3)		2,279
	Private key leak due to the Heartbleed bug	(4)		47
Partially leaky	POODLE-TLS padding oracle	(1.1)	Figure 4	816
	CBC padding oracle – OpenSSL AES-NI bug	(1.2)		96

inclusion of scripts sent over tainted channels in the top-level document may fully compromise integrity. The only protection mechanism available in modern browsers against the latter threat is Subresource Integrity (SRI) [2], a relatively recent web standard which allows websites to bind to `<script>` tags an `integrity` attribute storing a cryptographic hash of the script which is expected to be included by them. If the included script does not match the hash, the script is not executed, so SRI can be used to prevent the threats of script injection via network attacks.

The two integrity attacks above are equally dangerous and the most severe ones in terms of security, because they grant to the attacker active scripting capabilities on the web page, which we can thus deem as *compromisable*.

Definition 1 (Compromisable Page)**.** *A page is* compromisable *if and only if any of the following conditions holds:*

1) *the page is received from a tainted channel;*
2) *the page includes scripts in the top-level document from tainted channels without using SRI.*

Notice that the definition does not refer to Content Security Policy (CSP) [93], a web standard which can be used to prevent the execution of inline scripts and restrict content inclusion on web pages by means of a white-listing mechanism. In fact, CSP is ineffective against network attackers: if a page is compromisable because it is received from a tainted channel, the attacker may just strip away the CSP headers and `<meta>` tags to disable the protection; if instead a page is compromisable because it includes scripts from tainted channels, observe that CSP does not prevent the replacement of legitimate scripts with arbitrary malicious contents.

A second class of threats we are interested in allows SOP bypasses through compromisable pages. If a host contains at least one compromisable page, SOP becomes largely ineffective at defending it, because the attacker may get active scripting capabilities in its web origin and get access e.g., to its cookies and web storage. This motivates the following definition.

Definition 2 (Compromisable Host)**.** *A host is* compromisable *iff it is possible to retrieve a compromisable page from it.*

Finally, besides these obvious threats, it is worth noticing that there are also other integrity attacks which are subtler than script injection, but may achieve results as severe as page compromise under specific circumstances. For example: (i) the inclusion of stylesheets and web fonts can be used to perform *scriptless attacks*, which may enable the exfiltration of confidential information stored in the DOM [41]; (ii) the inclusion of Scalable Vector Graphics (SVG) images using tags like `<embed>` may lead to the injection of malicious HTML and JavaScript contents [40]; (iii) the inclusion of iframes can lead to exploitations against the top-level document via the postMessage API [81]; (iv) the result of an XMLHttpRequest can be passed to a function like `eval`, which converts strings into executable code and thus enables script injection [91].

To comprehensively characterize the pages suffering from these potential integrity issues, we leverage the Mixed Content [92] specification, which defines the reference security policy for the inclusion in HTTPS pages of contents delivered over HTTP channels. The key idea to uniformly capture these attacks is to reuse the definition of *blockable request* introduced in the Mixed Content specification, which mandates that compliant browsers must prevent HTTPS pages from sending this type of requests over HTTP channels.

Definition 3 (Blockable Request)**.** *A request is* blockable *if and only if it is not requesting any of the following resources:*

1) *images loaded via `<img>` or CSS;*
2) *video loaded via `<video>` and `<source>`;*
3) *audio loaded via `<audio>` and `<source>`.*

We similarly consider blockable requests over tainted channels as a possible source of integrity attacks, which leads to the following definition of *low integrity* page.

Definition 4 (Low Integrity Page)**.** *A page has* low integrity *if and only if any of the following conditions holds:*

1) *the page is compromisable;*
2) *the page includes sub-resources (other than scripts) via blockable requests sent over tainted channels.*

Low integrity pages which only satisfy the second condition do not necessarily provide active scripting capabilities to the attacker, yet they might still pose significant security threats in specific scenarios. That said, in the next sections we will often reason about the integrity of web pages to characterize additional web application attacks and our analysis will always be *optimistic*, i.e., we will assume that the attacker gets active scripting capabilities only in compromisable web pages and not in low integrity pages. We will also dispense with potential information leakages enabled by scriptless attacks [41], because they are not easy to exploit and depend on the details of specific web technologies. This conservative approach will limit the number of false positives in our security analysis.

B. Experimental Results

The homepages of the 10,000 crawled websites included sub-resources from 32,642 hosts. Our analysis exposed 977 low integrity pages (9.8%), including 898 compromisable pages where an attacker can get active scripting capabilities. Examples of major security-sensitive websites whose homepage was found compromisable include e-shops (alibaba.com, aliexpress.com, tmall.com), online banks (bankia.es, deutsche-bank.de, sparkasse.at, icicibank.com), social networks (myspace.com, linkedin.com, last.fm) and other prominent services (verizon.com, webex.com, livejournal.com).

Out of 898 compromisable pages, there are 238 pages received from tainted channels and 660 pages including scripts from tainted channels. Although the security dangers of these two cases are the same, the latter cases are particularly intriguing, because they show that the majority of the compromisable pages (73.5%) is harmed by the inclusion of external scripts. Since the majority of these scripts is hosted on domains which are not under the direct control of the embedding pages, SRI is the way to go to mitigate their threats: unfortunately, SRI is only used in 329 pages (3.3%) and does not prevent any page compromise in our dataset. Rather, we observe that there are 25 pages using SRI on some script tags, but are still compromisable because SRI is not deployed on *all* the script tags including contents from tainted channels.

Based on the previous considerations on external scripts, it is noteworthy that there exist popular script providers which are deployed on top of vulnerable HTTPS implementations, thus severely harming the integrity of a very large number of websites which include contents from them. Table II reports the most popular script providers which allow for the establishment of tainted channels, along with the number of the Alexa websites which include at least one script from them in their top-level document. These numbers show that by targeting only a couple of carefully chosen hosts, an attacker can fully undermine the integrity of a much larger number of websites, thus making integrity attacks cost-effective. For instance, consider the *LinkedIn Insight Tag*, a JavaScript code that enables the collection of visitors' data on webpages which include it and provides web analytics for LinkedIn ad campaigns. The script is loaded from a tainted channel served on snap.licdn.com (second row of Table II), which is vulnerable to MITM attacks due to a host affected by ROBOT at rewards.wholefoodsmarket.com, that presents a valid certificate for snap.licdn.com. The inclusion of this script threatens the integrity of 126 websites among the ones we analyzed, including notable examples such as auth0.com, britishairways.com, linode.com and teamviewer.com.

TABLE II
TOP SCRIPT PROVIDERS INTRODUCING INTEGRITY FLAWS

Script Provider	*Including Websites*
hm.baidu.com	188
snap.licdn.com	126
ads.pubmatic.com	47
zz.bdstatic.com	39
cdn.tagcommander.com	37
tag.baidu.com	20
geid.wbtrk.net	19
cdn.wbtrk.net	19
cdn.blueconic.net	14
dup.baidustatic.com	12

VI. AUTHENTICATION CREDENTIALS

In this section, we discuss the import of (partially) leaky and tainted channels on the security of common authentication credentials, i.e., passwords and cookies.

A. Security Analysis

In a typical web session, a website authenticates a user by checking her access credentials in the form of a username and a password. Upon their successful verification, the website stores in the user's browser a set of *session cookies*, which are automatically attached to the next requests sent to the website in order to authenticate them. There are quite a few well-known security threats in this common scenario [17] and vulnerable HTTPS implementations may severely compromise the security of web sessions. For example, if a user's password is disclosed to the attacker, the attacker will become able to start new sessions on the user's behalf and impersonate her at the website. Moreover, web session security requires both the confidentiality and the integrity of session cookies: lack of the former allows the attacker to hijack the user's session [16], while lack of the latter allows the attacker to force the user in the attacker's session [94]. Though the latter threat is easily underestimated, it may have serious security consequences on many web applications: for instance, e-payment websites may be targeted by such attacks to fool honest users into storing their credit card numbers in an attacker-controlled session.

Confidentiality of Passwords: A critical requirement for the confidentiality of passwords is that they are only input on HTTPS pages and only sent over HTTPS channels. Modern web browsers indeed warn users when these security important requirements are not met [72]. Unfortunately, vulnerable HTTPS implementations may make this security check insufficient: password confidentiality cannot be ensured when the password is sent over a leaky channel or entered into a compromisable web page where the attacker can get active scripting capabilities, thus becoming able to leak the password from the DOM.

Definition 5 (Low Confidentiality Password)**.** *A password has* low confidentiality *if and only if any of the following conditions holds:*

1) *the password is submitted over a leaky channel;*
2) *the page where the password is input is compromisable.*

Notice that partially leaky channels cannot be exploited to steal passwords, because the *secret repetition* assumption required by such side-channels is not satisfied by them.

Confidentiality of Cookies: The confidentiality of cookies against network attackers can be enforced by means of the

Secure attribute, because browsers ensure that Secure cookies are only sent on HTTPS channels and only made accessible to scripts running in HTTPS pages [6]. However, this defense mechanism becomes useless when HTTPS does not provide the expected security guarantees: for example, even partially leaky channels may be sufficient to disclose the content of Secure cookies, since cookies are automatically attached by browsers and thus satisfy the *secret repetition* assumption required by attacks like POODLE-TLS. Moreover, compromisable pages can be exploited to steal Secure cookies by means of malicious scripts which exfiltrate them, unless these cookies are also protected with the *HttpOnly* attribute, which prevents script accesses to them.

To make this intuition more precise, given a cookie c, we let $hosts(c)$ note the set of the hosts matching the domains which are entitled to access the content of c, as prescribed by RFC 6265 [6]. Intuitively, c is attached to a request towards h if and only if $h \in hosts(c)$.

Definition 6 (Low Confidentiality Cookie)**.** *A cookie c set by the host h has* low confidentiality *if and only if any of the following conditions holds:*

1) *there exists a host $h' \in hosts(c)$ which allows for the establishment of partially leaky channels;*
2) *c does not have the HttpOnly attribute set and there exists a compromisable host $h' \in hosts(c)$.*

Notice that breaking the confidentiality of a single session cookie may not be enough to let the attacker hijack the sessions of legitimate users, because websites may use multiple cookies for authentication purposes [20]. However, if *all* the session cookies of a website have low confidentiality, we have definite evidence that there is room for session hijacking.

Integrity of Cookies: Cookie integrity has notoriously been a major problem on the Web for many years, because cookies do not provide isolation by protocol, hence HTTP traffic can be abused to forge cookies which are indistinguishable from legitimate cookies set over HTTPS [6]. Also, cookies can be set by potentially untrusted *related* domains, i.e., domains that share a common suffix which is not included in the Public Suffix List.[6] The recommended way to enforce cookie integrity against network attacks on the current Web is configuring HSTS so that all the hosts entitled to set cookies can only be contacted over HTTPS [94]. An alternative approach is using *cookie prefixes*,[7] a recent addition to web browsers which can be used to prevent the setting of cookies over HTTP (when the *__Secure-* prefix appears in the cookie name) and, potentially, also from untrusted related domains (when the *__Host-* prefix appears in the cookie name, preventing cookie sharing between related domains). Unfortunately, these defenses might fail when HTTPS suffers from cryptographic flaws, because compromisable hosts would allow the attacker to break cookie integrity by corrupting HTTPS traffic; in particular, if the __Host- prefix is not used, any compromisable host on a related domain would be enough for the attack.

[6]https://publicsuffix.org/

[7]https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-02

More precisely, given a host h, we let $related(h)$ note the set of the hosts whose domain is related to the domain of h. Technically, this implies that any host $h' \in related(h)$ can set a cookie c such that $h \in hosts(c)$, which means that c might be eventually received by h and harm its security. Notice that, although h' may not be able to directly overwrite host-only cookies set by h, it could still obtain the same effect by *cookie shadowing*, i.e., by setting domain cookies with the same name of host-only cookies so that the target website is fooled into accessing the former [94]. Also, the domain cookies may be set before the host-only cookies are ever issued, which makes cookie shadowing attempts undetectable in general.

Definition 7 (Low Integrity Cookie)**.** *A cookie c set by the host h has* low integrity *if and only if any of the following conditions holds:*

1) *h is compromisable;*
2) *c does not have the __Host- prefix and there exists a compromisable host $h' \in related(h)$.*

B. Experimental Results

We first isolated from the 10,000 crawled websites the 4,018 websites with a private area, i.e., supporting the establishment of authenticated sessions. This was assessed heuristically by checking any of the following two conditions:

1) the page includes a login form, i.e., a form with both a text/email field and a password field;
2) the page includes a single sign-on library from a list of popular identity providers.

Out of the 4,018 websites with a private area, we found 404 cases where password confidentiality was not ensured (10.0%), either because the password was sent over a leaky channel or because the page with the login form was compromisable. Attacks against these pages would allow an attacker to impersonate legitimate users and start new sessions on their behalf.

We then turned our attention to the security analysis of cookies. The left portion of Table III reports the number of low confidentiality and low integrity cookies collected from the full set of 10,000 websites. In total, 19.1% of all cookies have low confidentiality, while 18.7% have low integrity, which suggests that the risks of cookie leakage and cookie tampering in the wild are far from remote. The most interesting observation is that ensuring confidentiality for domain cookies is much harder than for host-only cookies: 21.6% of the domain cookies have low confidentiality, while this percentage decreases to 12.5% for host-only cookies. The reason is that the attack surface for domain cookies is much larger, because it is enough to find one related domain which suffers from confidentiality issues to leak them; yet, 73.1% of the collected cookies are domain cookies. As to integrity, the difference between domain cookies and host-only cookies is almost negligible and the most concerning observation there is that only one of the 10,000 websites we crawled makes use of cookie prefixes to improve cookie integrity.

TABLE III
COOKIE CONFIDENTIALITY AND INTEGRITY ISSUES

	All cookies			*Session cookies*		
	Host-only (11,784)	Domain (31,998)	Total (43,782)	Host-only (3,942)	Domain (7,818)	Total (11,760)
Low confidentiality	1,469 (12.5%)	6,903 (21.6%)	8,372 (19.1%)	425 (10.8%)	1,633 (20.1%)	2,058 (17.5%)
Low integrity	2,093 (17.8%)	6,116 (19.1%)	8,209 (18.7%)	694 (17.6%)	1,435 (18.3%)	2,129 (18.1%)

To better understand the import of these numbers on web session security, we restricted our attention just to the session cookies set from the 4,018 websites featuring a private area. Session cookies were identified using a heuristic proposed in previous work [16], which was shown to be fairly accurate in practice and nicely fits our large-scale investigation. The right portion of Table III presents the results of such analysis, which shows that the high-level picture does not change significantly when we focus just on session cookies. Moreover, we observed that 412 websites (10.2%) may leak all their session cookies due to cryptographic flaws, which may allow network attackers to impersonate legitimate users of these websites. It is worth noticing that, if all these cookies could be marked as HttpOnly without breaking the functionality of the websites, the number of websites vulnerable to this threat would reduce to 207 (5.1%). This shows that a complete deployment of the HttpOnly attribute would be quite effective, yet not sufficient to fully protect honest users against session hijacking, since session cookies could still be sent over partially leaky channels.

Finally, we found 543 websites (13.5%) whose session cookies all have low integrity, which may allow the attacker to force honest users into attacker-controlled sessions (cookie forcing). In all cases, the cookie integrity problems were due to the presence of a vulnerability in a related domain, but we also found 404 cases where also the base domain suffers from integrity flaws. The __Host- cookie prefix would be useful to improve session security in the 139 cases (25.6%) where the integrity vulnerabilities are confined to related domains, but unfortunately only one of the crawled websites (dropbox.com) uses cookie prefixes. Remarkably, we observe that 22 out of these 139 cases (15.8%) could safely introduce the __Host- prefix without compatibility problems, as none of their session cookies is a domain cookie.

C. Detected Attacks

Since the numbers in the previous section may have been affected by the use of heuristics to detect private areas and session cookies, we report on a selected set of manual experiments to confirm the existence of credential stealing and session hijacking attacks on prominent websites in the wild. For ethical reasons, we did not tamper with websites to test concrete attacks. Rather, we carefully checked all the conditions required to mount attacks against the targets and employed a local proxy to simulate the attack.

One notable example where password confidentiality is not ensured is Myspace. The login page and the endpoint where the password is sent are both served on myspace.com, that is directly vulnerable to ROBOT. Thus, an attacker could either sniff the password from a tainted channel or actively inject a script in the page to leak access credentials from the DOM.

Session hijacking has been identified as a realistic threat on the yandex.com web portal. In this case the main host itself is secure, but the presence of a partially leaky channel on api.developer.store.yandex.com makes possible for an attacker to disclose all domain cookies by forcing the victim's client to iterate requests against that specific host from an attacker's controlled origin. All cookies set by the website after logging in are domain cookies, including `Session_id` that is used to authenticate user sessions, proving the attack to be practical.

Finally, cookie forcing has been found on the Microsoft webmail live.com. Our large-scale assessment found that the host exchange.backcountry.com is vulnerable to ROBOT and presents a certificate valid also for outlook.live.com. Since the host of one of the related domains of live.com is compromisable, an attacker could mount a MITM to overwrite the cookies of a honest user, forcing her into the attacker's session.

VII. WEB TRACKING

In this section, we discuss how leaky and tainted channels can be abused to track navigation behaviours of web users and breach privacy at scale.

A. Security Analysis

Online tracking is pervasive on the Web and has significant privacy implications [68], [34]. Third-party tracking is particularly dangerous for user privacy, because it allows trackers to reconstruct a cross-site navigation profile of online users at scale. In this form of tracking, the tracker is embedded on external websites in a *third-party* position, i.e., using iframes, so that it is able to set a tracker-owned cookie containing a unique identifier in the user's browser. Every time the user accesses a website where the tracker is present, her browser will automatically send a request including the cookie to the tracker: since this request also includes the Referer header, which tracks the page from which the request was sent, the tracker becomes able to reconstruct the navigation profile of the user identified by the cookie.

Network attackers can easily disclose a lot of information about navigation patterns just because they are in control of the network. For instance, they can link a given IP address to all the domain names requested from it. However, this does not necessarily allow the attacker to build a navigation profile of the target user, e.g., because the same IP address is shared by multiple users (in case of NATs) or because the same user is assigned different IP addresses upon different connections.

TABLE IV
TOP TRACKERS INTRODUCING PRIVACY FLAWS

Tracker	*Including Websites*
snap.licdn.com	126
l.betrad.com	100
hbopenbid.pubmatic.com	76
kraken.rambler.ru	66
ads.pubmatic.com	47
simage2.pubmatic.com	30
counter.rambler.ru	25
tag.1rx.io	20
fw-sync.nuggad.net	18
t.pubmatic.com	17

Still, it is known that network attackers may become able to build cross-site navigation profiles of users by monitoring the presence of tracking cookies in the HTTP traffic [35]. Here we discuss a similar attack, which exploits existing confidentiality issues in the HTTPS implementations of web trackers.

Assume the attacker wants to learn whether a user identified by the tracking cookie c has ever accessed the page p. If the page p includes sub-resources from a tracker-controlled host $h \in hosts(c)$ over a leaky channel, the attacker may be able to associate the value of c to the page p via the Referer header. However, even if p does not include anything from the tracker, the attacker can force such leaky content inclusion when p itself is compromisable, thus amplifying the privacy risks. This leads to the following definition.

Definition 8 (Profiling). *A tracking cookie c allows* profiling *on the page p if and only if there exists a host $h \in hosts(c)$ which allows for the establishment of leaky channels and any of the following conditions holds:*

1) *p sends a request to h;*
2) *p is compromisable.*

B. Experimental Results

We downloaded a list of 2,399 prominent tracking domains provided by Disconnect[8] and we checked for content inclusions from them in the 10,000 websites taken from Alexa. In particular, we focused on inclusions from any sub-domain of the trackers, because domain cookies could be used to perform tracking when including contents (of any type) from them. By doing this, we managed to identify a set of 4,226 tracker-controlled hosts which may potentially be abused to perform user profiling on the Alexa websites. We then analyzed these hosts, checking whether they allow the establishment of leaky channels, and it turned out that 82 (1.9%) of them suffer from this security issue.

We report in Table IV the list of the most popular vulnerable tracker-controlled hosts, along with the number of websites from Alexa which included contents from them. These vulnerable hosts are controlled by different companies basing their business on web tracking and analytics. By checking against Cookiepedia,[9] we confirmed that at least four of these companies rely on the practice of setting long-lived domain cookies for third-party tracking: PubMatic, Rambler, RhythmOne and nugg.ad. To understand the privacy implications of these security issues, we focused on the hosts controlled by PubMatic, which are the most numerous: attacking the vulnerable hosts of PubMatic would allow one to reconstruct navigation profiles over 142 websites which include contents from them. Moreover, by injecting references to these hosts in any of the 898 compromisable homepages from our dataset, this privacy attack could be further amplified to track navigation behaviors across 968 websites (9.7%).

[8]https://github.com/disconnectme/disconnect-tracking-protection
[9]https://cookiepedia.co.uk/

VIII. CLOSING REMARKS

A. Related Work

Novel attacks against TLS were often released with the analysis of their impact in the wild, by measuring the number of vulnerable servers in scans of the IPv4 address space or the most popular websites ranked by Alexa. This was true for RSA keys factorable by Batch GCD algorithm [42] and attacks like DROWN [4] or Logjam [1]. Small subgroup attacks against Diffie-Hellman were measured by Valenta et al. [84]. Dorey et al. [28] measured misconfigured DH key parameters that potentially contain backdoors. The prevalence of several attacks against the Elliptic Curve DH key establishment in TLS was measured by Valenta et al. [85]. Some vulnerability measurements were revisited to track the progress of patching, such as Heartbleed [33] and the Batch GCD method [39]. The SSL Pulse project [64] releases monthly measurements on the prevalence of certain attacks and feature support. Novel variants of old vulnerabilities were discovered, such as in the ROBOT attack [14], or for CBC oracles via the TLS-Attacker fuzzing tool [80]. Summaries of known TLS vulnerabilities were published by Levillain et al. [54], [55] and by the IETF [75]. Lessons learned from attacks known before 2013 have been summarized by Meyer and Schwenk [56].

None of the papers above systematically discusses and quantifies web application security issues. However, the risks coming from the partial adoption of HTTP on HTTPS websites have been studied in several research papers. For instance, [22] performed a large-scale analysis of the security risks of mixed content websites, [51] analyzed the state of the HSTS deployment and [77] studied the threats posed by the leakage of cookies over HTTP channels. There are also a few papers quantifying how much incorrect TLS implementations affect the security of the email infrastructure [30], [43].

The present paper contributes to the increasingly popular research line on large-scale security evaluations of the Web [86]. Though several papers analyzed the security of the HTTPS certificate ecosystem [31], [44], [87], we are not aware of any scientific publication which quantifies how much cryptographic weaknesses in TLS implementations may harm web application security. Other important aspects of web application security which have been investigated by previous large-scale measurements include the dangers of remote JavaScript inclusion [61], the prevalence of DOM-based XSS [53] and the state of the CSP adoption [18], [19], [91].

B. Ethics and Limitations

Due to both legal and ethical reasons, our analysis of TLS vulnerabilities in the wild was limited to an unintrusive scan based on the use of publicly available tools. The exploitability of the discovered vulnerabilities was exclusively judged through a systematic analysis of the output of those tools, defined via an extensive account of the existing literature on attacks against TLS (summarized in the attack trees of Section III). All the vulnerabilities we tested have been first published at major computer security conferences and/or received extensive coverage in the hacking community. They have all been shown to be exploitable in the wild, requiring a practically feasible amount of computational power. Since we did not run any active attack attempt, it is possible that the vulnerabilities reported in the present study are not actually exploitable in practice, e.g., due to the deployment of anomaly detection systems. That said, the real effectiveness of such kind of mitigations is hard to assess and fixing the vulnerabilities would be certainly preferable from a security perspective.

The set of the studied web application vulnerabilities is not intended to be exhaustive: it just gives evidence of significant security threats posed by vulnerable TLS implementations and allows for a systematic quantification of their practical relevance. The usage of heuristics in a few parts of our experimental evaluation, e.g., for session cookie detection, may have introduced a bias in our quantitative assessment: better heuristics may make the analysis more precise, but they are likely not going to entail a significant change of the currently drawn picture, given the large scale of the experiments. We manually confirmed some of the security issues to provide further evidence of the effectiveness of our methodology. We also rechecked all the vulnerable sites explicitly mentioned in this paper at the beginning of January 2019 and most of them have fixed the issues since our first scan. We have responsibly reported the discovered flaws to the sites that are still vulnerable and only one has answered dismissively with: "this case has no direct security impact and we will not take an immediate action or a fix". In fact, we did not find a strong interest in TLS-related issues even in vulnerability reward programs but the fact that many sites fixed the problems is promising in terms of awareness of the risks due to wrong HTTPS implementations.

C. Summary and Perspective

Though the use of HTTPS is necessary for web application security, it is not a panacea, because flaws in the underlying TLS implementation may have a significant security import at the application layer. We have computed a few disquieting numbers in our present evaluation: we summarize here the most relevant observations and present our perspective on the main findings.

Almost 10% of the homepages of the crawled websites is *compromisable*, i.e., a determined network attacker may get active scripting capabilities on them. For approximately 25% of the compromisable pages, this security problem can be fixed just by revising the cryptographic implementation of their host. Unfortunately, the security of the other 75% pages is downgraded by the inclusion of external scripts retrieved over tainted channels: this makes it hard for web developers to get a realistic picture of the cryptographic robustness of their web applications and fix potential issues. Since we only crawled homepages, our findings under-approximate the real situation, as other webpages might include more insecure content. SRI is a potentially effective defense mechanism for these cases, but its adoption is minuscule and sub-optimal: approximately, just 3% of the pages are using SRI and none of the attacks we found is actually stopped by the current deployment.

For what concerns web session security, we found room for session hijacking attacks by cookie stealing in around 10% of the crawled websites, while more than 13% of the websites were found vulnerable to cookie forcing. The most concerning aspect of cookie security is the impact of related domains: even a single security issue on a related-domain host may completely undermine session security, because related-domain hosts may break both the confidentiality and the integrity of session cookies. Room for password theft was also found in 10% of the login pages.

Finally, cryptographic weaknesses in the TLS implementations of web trackers may pose major threats to user privacy at scale. In our experimental analysis, we discovered some prominent trackers inadvertently introducing this security problem on a significant amount of websites. The most disquieting aspect here is that just a single vulnerable tracker may significantly harm user privacy at scale, as long as it is popular enough to be included on many different websites: for instance, one problem we found allows for user profiling on 142 websites, which can be further increased to 968 websites by running a more powerful variant of the attack.

We expect this bleak picture to improve after both browsers and servers provide a better support for TLS 1.3. Major browser vendors already announced that they will deprecate TLS 1.0 and 1.1 in 2020 [8]. However, backward compatibility and slow adoption are always a major hindrance for web security improvements, so we expect old TLS versions to stick around for at least a few years. The present paper acts as a cautionary tale of the threats they pose: we plan to supply the toolchain developed for our study as a web application to support developers who are interested in mitigating these threats.

Acknowledgments: Riccardo Focardi and Marco Squarcina were partially supported by CINI project FilieraSicura, funded by CISCO Systems Inc. and Leonardo SpA. Marco Squarcina was also partially supported by the European Research Council (ERC) under the European Unions Horizon 2020 research (grant agreement No 771527-BROWSEC). Matus Nemec was partially supported by the Czech Science Foundation under project GA16-08565S. Marco Squarcina did most of the work on this project while he was a postdoctoral researcher at Ca' Foscari University.

References

[1] D. Adrian, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, P. Zimmermann, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, and E. Thomé, "Imperfect Forward Secrecy," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security – CCS '15*. ACM, 2015.

[2] D. Akhawe, F. Braun, F. Marier, and J. Weinberge, "W3C Recommendation: Subresource Integrity," https://www.w3.org/TR/SRI/, 2016.

[3] N. J. AlFardan and K. G. Paterson, "Lucky Thirteen: Breaking the TLS and DTLS Record Protocols," in *2013 IEEE Symposium on Security and Privacy*. IEEE, may 2013.

[4] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt, "DROWN: Breaking TLS Using SSLv2," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 689–706. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram

[5] R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, and J.-K. Tsay, "Efficient Padding Oracle Attacks on Cryptographic Hardware," in *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, 2012, pp. 608–625.

[6] A. Barth, "HTTP State Management Mechanism," http://tools.ietf.org/html/rfc6265, 2011.

[7] T. Be'ery and A. Shulman, "A Perfect CRIME? Only TIME Will Tell," Black Hat Europe 2013, 2013, online, cit. [2018-10-29]. [Online]. Available: https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf

[8] D. Benjamin, "Modernizing Transport Security," Google Security Blog, 2018, cit. [2019-01-29]. [Online]. Available: https://security.googleblog.com/2018/10/modernizing-transport-security.html

[9] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A Messy State of the Union: Taming the Composite State Machines of TLS," in *2015 IEEE Symposium on Security and Privacy*. IEEE, may 2015.

[10] K. Bhargavan, A. D. Lavaud, C. Fournet, A. Pironti, and P. Y. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS," in *2014 IEEE Symposium on Security and Privacy*. IEEE, may 2014.

[11] K. Bhargavan and G. Leurent, "Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH," in *Proceedings of the 2016 Network and Distributed System Security Symposium*. Internet Society, 2016.

[12] ——, "On the Practical (In-)Security of 64-bit Block Ciphers," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. ACM Press, 2016.

[13] D. Bleichenbacher, "Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS#1," in *Advances in Cryptology – CRYPTO '98*. Springer Berlin Heidelberg, 1998, pp. 1–12.

[14] H. Böck, J. Somorovsky, and C. Young, "Return Of Bleichenbacher's Oracle Threat (ROBOT)," in *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 817–849. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[15] B. B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *Computer Security – ESORICS 2011*. Springer Berlin Heidelberg, 2011, pp. 355–371.

[16] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan, "CookiExt: Patching the browser against session hijacking attacks," *Journal of Computer Security*, vol. 23, no. 4, pp. 509–537, 2015.

[17] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta, "Surviving the Web: A Journey into Web Session Security," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 13:1–13:34, 2017.

[18] S. Calzavara, A. Rabitti, and M. Bugliesi, "Content Security Problems?: Evaluating the Effectiveness of Content Security Policy in the Wild," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1365–1375.

[19] ——, "Semantics-based analysis of content security policy deployment," *TWEB*, vol. 12, no. 2, pp. 10:1–10:36, 2018.

[20] S. Calzavara, G. Tolomei, A. Casini, M. Bugliesi, and S. Orlando, "A Supervised Learning Approach to Protect Client Authentication on the Web," *TWEB*, vol. 9, no. 3, pp. 15:1–15:30, 2015.

[21] B. Canvel, A. Hiltgen, S. Vaudenay, and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," in *Advances in Cryptology – CRYPTO 2003*. Springer Berlin Heidelberg, 2003, pp. 583–599.

[22] P. Chen, N. Nikiforakis, C. Huygens, and L. Desmet, "A Dangerous Mix: Large-Scale Analysis of Mixed-Content Websites," in *Information Security, 16th International Conference, ISC 2013, Proceedings*, 2013, pp. 354–363.

[23] X. de Carné de Carnavalet and M. Mannan, "Killed by Proxy: Analyzing Client-end TLS Interception Software," in *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016.

[24] J. de Ruiter and E. Poll, "Protocol State Fuzzing of TLS Implementations," in *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 193–206. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter

[25] T. Dierks and C. Allen, "RFC 2246: The TLS Protocol Version 1.0," Internet Engineering Task Force (IETF), 1999. [Online]. Available: https://tools.ietf.org/html/rfc2246

[26] T. Dierks and E. Rescorla, "RFC 4346: The Transport Layer Security (TLS) Protocol Version 1.1," Internet Engineering Task Force (IETF), 2006. [Online]. Available: https://tools.ietf.org/html/rfc4346

[27] ——, "RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2," Internet Engineering Task Force (IETF), 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246

[28] K. Dorey, N. Chang-Fong, and A. Essex, "Indiscreet Logs: Diffie-Hellman Backdoors in TLS," in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017.

[29] T. Duong and J. Rizzo, "Here Come The XOR Ninjas," 2011, online, cit. [2018-10-29]. [Online]. Available: https://bug665814.bugzilla.mozilla.org/attachment.cgi?id=540839

[30] Z. Durumeric, J. A. Halderman, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, and M. Bailey, "Neither Snow Nor Rain Nor MITM..." in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference - IMC '15*. ACM Press, 2015.

[31] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Proceedings of the 2013 Internet Measurement Conference, IMC 2013*, 2013, pp. 291–304.

[32] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The Security Impact of HTTPS Interception," in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017.

[33] Z. Durumeric, M. Payer, V. Paxson, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, and J. Beekman, "The Matter of Heartbleed," in *Proceedings of the 2014 Conference on Internet Measurement Conference – IMC '14*. ACM Press, 2014.

[34] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[35] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten, "Cookies that give you away: The surveillance implications of web tracking," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 289–299.

[36] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz, "Measuring HTTPS Adoption on the Web," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1323–1338. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/felt

[37] C. Garman, K. G. Paterson, and T. V. der Merwe, "Attacks Only Get Better: Password Recovery Attacks Against RC4 in TLS," in *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015, pp. 113–128. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/garman

[38] 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 *Proceedings of the 2012 ACM conference on Computer and communications security – CCS '12*. ACM Press, 2012.

[39] M. Hastings, J. Fried, and N. Heninger, "Weak Keys Remain Widespread in Network Devices," in *Proceedings of the 2016 ACM on Internet Measurement Conference – IMC '16*. ACM Press, 2016.

[40] M. Heiderich, T. Frosch, M. Jensen, and T. Holz, "Crouching tiger - hidden payload: security risks of scalable vectors graphics," in *Proceed-*

ings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, 2011, pp. 239–250.

[41] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk, "Scriptless attacks: Stealing more pie without touching the sill," *Journal of Computer Security*, vol. 22, no. 4, pp. 567–599, 2014.

[42] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman, "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices," in *Proceedings of the 21st USENIX Security Symposium (USENIX Security 12)*. USENIX, 2012, pp. 205–220. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger

[43] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. A. Kaafar, "TLS in the Wild: An Internet-wide Analysis of TLS-based Protocols for Electronic Communication," in *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, 2016.

[44] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL landscape: a thorough analysis of the x.509 PKI using active and passive measurements," in *Proceedings of the 11th ACM SIGCOMM Internet Measurement Conference, IMC '11*, 2011, pp. 427–444.

[45] F. Indutny, "Extracting server private key using Heartbleed OpenSSL vulnerability," GitHub, 2014, cit. [2019-01-29]. [Online]. Available: https://github.com/indutny/heartbleed

[46] T. Jager, J. Schwenk, and J. Somorovsky, "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security – CCS '15*. ACM Press, 2015.

[47] B. Kaliski, "RFC 2313: PKCS #1: RSA Encryption Version 1.5," Internet Engineering Task Force (IETF), 1998. [Online]. Available: https://tools.ietf.org/html/rfc2313

[48] J. Kelsey, "Compression and Information Leakage of Plaintext," in *Fast Software Encryption*. Springer Berlin Heidelberg, 2002, pp. 263–276.

[49] M. Kikuchi, "How I discovered CCS Injection Vulnerability (CVE-2014-0224)," 2014, online, cit. [2018-10-29]. [Online]. Available: http://ccsinjection.lepidum.co.jp/blog/2014-06-05/CCS-Injection-en/index.html

[50] V. Klíma, O. Pokorný, and T. Rosa, "Attacking RSA-Based Sessions in SSL/TLS," in *Cryptographic Hardware and Embedded Systems – CHES 2003*. Springer Berlin Heidelberg, 2003, pp. 426–440.

[51] M. Kranch and J. Bonneau, "Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning," in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*, 2015.

[52] K. Krombholz, W. Mayer, M. Schmiedecker, and E. R. Weippl, ""I Have No Idea What I'm Doing" - On the Usability of Deploying HTTPS," in *26th USENIX Security Symposium, USENIX Security 2017*, 2017.

[53] S. Lekies, B. Stock, and M. Johns, "25 million flows later: large-scale detection of DOM-based XSS," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, 2013, pp. 1193–1204.

[54] O. Levillain, "A study of the TLS ecosystem," Dissertation thesis, 2017, online, cit. [2018-10-29]. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01454976/document

[55] O. Levillain, B. Gourdin, and H. Debar, "TLS Record Protocol: Security Analysis and Defense-in-depth Countermeasures for HTTPS," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '15*. ACM Press, 2015.

[56] C. Meyer and J. Schwenk, "SoK: Lessons Learned from SSL/TLS Attacks," in *Information Security Applications*. Springer International Publishing, 2014, pp. 189–209.

[57] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, "Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks," in *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014, pp. 733–748. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/meyer

[58] B. Moeller and A. Langley, "RFC 7507: TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks," Internet Engineering Task Force (IETF), 2015. [Online]. Available: https://tools.ietf.org/html/rfc7507

[59] B. Möller, T. Duong, and K. Kotowicz, "This POODLE Bites: Exploiting The SSL 3.0 Fallback," 2014, online, cit. [2018-10-29]. [Online]. Available: https://www.openssl.org/~bodo/ssl-poodle.pdf

[60] National Institute of Standards and Technology, "Digital Signature Standard (DSS)," FIPS 186-4, 2013, online, cit. [2018-10-29]. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[61] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. V. Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You are what you include: large-scale evaluation of remote javascript inclusions," in *ACM Conference on Computer and Communications Security, CCS'12*, 2012, pp. 736–747.

[62] A. Popov, "RFC 7465: Prohibiting RC4 Cipher Suites," Internet Engineering Task Force (IETF), 2015. [Online]. Available: https://tools.ietf.org/html/rfc7465

[63] A. Prado, N. Harris, and Y. Gluck, "SSL, gone in 30 seconds: A BREACH beyond CRIME," Black Hat USA 2013, 2013, cit. [2018-10-29]. [Online]. Available: https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf

[64] Qualys, "SSL Pulse; Monthly Scan: October 03, 2018," 2018, online, cit. [2018-10-29]. [Online]. Available: https://www.ssllabs.com/ssl-pulse/

[65] M. Ray and S. Dispensa, "Renegotiating TLS," 2009, online, cit. [2018-10-29]. [Online]. Available: https://pdfs.semanticscholar.org/1061/99bc6833cabeebef335437202c3245d5efb5.pdf

[66] E. Rescorla, "RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force (IETF), 2018. [Online]. Available: https://tools.ietf.org/html/rfc8446

[67] J. Rizzo and T. Duong, "The CRIME attack," ekoparty security conference 2012, 2012, online, cit. [2018-10-29]. [Online]. Available: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit#slide=id.g1d134dff_1_222

[68] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and Defending Against Third-Party Tracking on the Web," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*, 2012, pp. 155–168.

[69] E. Ronen, R. Gillham, D. Genkin, A. Shamir, D. Wong, and Y. Yarom, "The 9 Lives of Bleichenbacher's CAT: New Cache ATtacks on TLS Implementations," To appear in the IEEE Symposium on Security and Privacy, 2019, available online: Cryptology ePrint Archive, Report 2018/1173 https://eprint.iacr.org/2018/1173.

[70] E. Ronen, K. G. Paterson, and A. Shamir, "Pseudo Constant Time Implementations of TLS Are Only Pseudo Secure," Cryptology ePrint Archive, Report 2018/747, 2018, https://eprint.iacr.org/2018/747.

[71] J. Salowey, H. Zhou, P. Eronen, and H. Tschofenig, "RFC 5077: Transport Layer Security (TLS) Session Resumption without Server-Side State," Internet Engineering Task Force (IETF), 2008. [Online]. Available: https://tools.ietf.org/html/rfc5077

[72] E. Schechter, "Next Steps Toward More Connection Security," Google Security Blog, 2017, cit. [2019-01-29]. [Online]. Available: https://security.googleblog.com/2017/04/next-steps-toward-more-connection.html

[73] ——, "A milestone for Chrome security: marking HTTP as "not secure"," The Keyword, 2018, cit. [2019-01-29]. [Online]. Available: https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/

[74] B. Schneier, *Secrets and lies - digital security in a networked world: with new information about post-9/11 security*. Wiley, 2004.

[75] Y. Sheffer, R. Holz, and P. Saint-Andre, "RFC 7457: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)," Internet Engineering Task Force (IETF), 2015. [Online]. Available: https://tools.ietf.org/html/rfc7457

[76] I. Shparlinski, "The Insecurity of the Digital Signature Algorithm with Partially Known Nonces," in *Cryptographic Applications of Analytic Number Theory*. Birkhäuser Basel, 2003, pp. 201–206. [Online]. Available: https://doi.org/10.1007%2F978-3-0348-8037-4_17

[77] S. Sivakorn, I. Polakis, and A. D. Keromytis, "The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information," in *IEEE Symposium on Security and Privacy, SP 2016*, 2016, pp. 724–742.

[78] B. Smith, "POODLE applicability to TLS 1.0+," IETF TLS mailing list, 2014, online, cit. [2018-10-29]. [Online]. Available: https://www.ietf.org/mail-archive/web/tls/current/msg14058.html

[79] J. Somorovsky, "Curious Padding oracle in OpenSSL (CVE-2016-2107)," On Web-Security and -Insecurity blog, 2016, online, cit. [2018-10-29]. [Online]. Available: https://web-in-security.blogspot.com/2016/05/curious-padding-oracle-in-openssl-cve.html

[80] ——, "Systematic Fuzzing and Testing of TLS Libraries," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security – CCS'16*. ACM Press, 2016.

[81] S. Son and V. Shmatikov, "The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites," in *20th Annual Network and Distributed System Security Symposium, NDSS 2013*, 2013.

[82] D. Springall, Z. Durumeric, and J. A. Halderman, "Measuring the Security Harm of TLS Crypto Shortcuts," in *Proceedings of the 2016 ACM on Internet Measurement Conference - IMC '16*. ACM, 2016.

[83] Synopsys, "The Heartbleed Bug (CVE-2014-0160)," 2014, online, cit. [2018-10-29]. [Online]. Available: http://heartbleed.com/

[84] L. Valenta, D. Adrian, A. Sanso, S. Cohney, J. Fried, M. Hastings, J. A. Halderman, and N. Heninger, "Measuring small subgroup attacks against Diffie-Hellman," in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017.

[85] L. Valenta, N. Sullivan, A. Sanso, and N. Heninger, "In search of CurveSwap: Measuring elliptic curve implementations in the wild," Cryptology ePrint Archive, Report 2018/298, 2018, https://eprint.iacr.org/2018/298.

[86] T. van Goethem, P. Chen, N. Nikiforakis, L. Desmet, and W. Joosen, "Large-Scale Security Analysis of the Web: Challenges and Findings," in *Trust and Trustworthy Computing - 7th International Conference, TRUST 2014. Proceedings*, 2014, pp. 110–126.

[87] B. VanderSloot, J. Amann, M. Bernhard, Z. Durumeric, M. Bailey, and J. A. Halderman, "Towards a Complete View of the Certificate Ecosystem," in *Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016*, 2016, pp. 543–549.

[88] S. Vaudenay, "Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS..." in *Advances in Cryptology – EUROCRYPT 2002*. Springer Berlin Heidelberg, 2002, pp. 534–545.

[89] J. Vehent, "Security/Server Side TLS (version 4.1)," MozillaWiki, 2018, online, cit. [2018-10-29]. [Online]. Available: https://wiki.mozilla.org/Security/Server_Side_TLS#Recommended_configurations

[90] L. Waked, M. Mannan, and A. Youssef, "To Intercept or Not to Intercept," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS '18*. ACM Press, 2018. [Online]. Available: https://doi.org/10.1145%2F3196494.3196528

[91] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, "CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[92] M. West, "W3C Candidate Reccomendation: Mixed Content," https://www.w3.org/TR/mixed-content/, 2016.

[93] ——, "W3C Working Draft: Content Security Policy Level 3," https://www.w3.org/TR/CSP3/, 2018.

[94] X. Zheng, J. Jiang, J. Liang, H. Duan, S. Chen, T. Wan, and N. Weaver, "Cookies Lack Integrity: Real-World Implications," in *24th USENIX Security Symposium, USENIX Security 15*, 2015, pp. 707–721.

Appendix

A. Notable Out of Scope Attacks Against TLS

Several vulnerabilities of TLS are not exploitable in the wild, based on recent measurements or due to the configuration of modern clients.

Diffie-Hellman key establishment attacks (MITM attacks): Static DH key exchange susceptible to small subgroup attacks [84] is not supported by modern browsers and support for vulnerable static ECDH key exchange was removed in browsers we target. Furthermore, some browsers already deprecated DHE [28] and more should follow. Possibly backdoored DH groups were observed in the wild [28]. It is not possible to intercept the connection without the knowledge of the backdoor, hence only the attacker that generated the backdoored parameters could mount MITM attacks. The Logjam attack [1] forces the server to choose a small 512-bit DH group, however modern browsers enforce minimal group size, where the discrete logarithm problem is infeasible.

A recent paper [85] measured the prevalence and feasibility of several attacks on ECDH (static and ephemeral) key establishment. Many servers fail to check parameters and many reuse ephemeral keys [82], no server was found that would do both. Their further findings indicate that several other proposed attacks (such as CurveSwap) are infeasible in TLS.

State machine bugs (up to MITM): The state machines of TLS are complicated and not explicitly stated in the standards. Their implementations are a common source of bugs. The Early CCS attack found by [49] allowed a MITM attack. Due to a bug in OpenSSL, running the Change Cipher Spec Protocol early, both the server and the client used a zero-length master key. While the bug is still found on some servers [64], browsers have been patched. FREAK, another client-side bug [9] allowed the attacker to downgrade the client to `RSA_EXPORT` (easily factorable 512-bit keys), even when the client did not offer such ciphersuite. Searching for new state machine bugs was out of our scope and is the focus of systematic studies of state machine implementations [9], [24].

Private key leakage (MITM): Private RSA keys generated with insufficient entropy can lead to servers sharing primes in their keys, allowing such RSA keys to be factored by a simple greatest common divisor (GCD) computation. Batch GCD, an efficient version of the algorithm that can handle millions of moduli, revealed that such keys were widespread [42], [39], likely due to consumer devices that generate their keys shortly after boot, before entropy is collected. The bugs are not prevalent on commercial servers from the Alexa list.

DSA and ECDSA private keys can be recovered if the same secret nonce is used more than once [60], yet it happens with negligible probability. Even biased nonces can be used to reveal the private key, if enough signatures with a small number of known nonce bits are known [76]. However, testing for such side-channels is infeasible. Remote time side-channel attacks were demonstrated [15], yet the bugs were known beforehand. Timing attacks often rely on observing cache access [70] that cannot be performed from a MITM position.

Certificate validation bugs (MITM): Some non-browser clients were shown to have flawed certificate validation [38], accepting invalid certificates. We assume correct certificate validation in modern browsers and users following browser warnings. Certificate validation bugs in software and hardware that intercepts TLS connections [23], [32], [90] are also out of scope of our analysis.

Transcript collision attacks (MITM): We leave out transcript collision attacks [11] since the performance of the algorithms for finding (chosen prefix) collision in the hash functions is not yet practical enough.

Further CBC-mode attacks (partial secret leakage): Attacks based on timing side-channels like Lucky13 [3] are infeasible to assess over the Internet. The original POODLE attack [59] cannot be applied, since browsers disabled SSLv3 support. Browsers that fix bugs, such as an SOP-bypass, or implement the 1/n-1 split will resist BEAST [29]. We leave for future work the attacks that enable partially leaky channels from server to client, like BREACH [63], that requires specific conditions at the server's application layer to be exploited.

Weak ciphers (partial secret leakage): Authentication tokens and cookies could be disclosed due to collisions in CBC mode of a 64-bit block cipher, such as Triple-DES (3DES), via the Sweet32 attack [12]. Due to the birthday paradox, a ciphertext collision between a block that encrypts a known plaintext and a block that encrypts the cookie is expected with high probability after the client sends about 2^{32} messages. Modern browsers only support 3DES as a fallback since AES (with 128-bit blocks) is preferred by servers. An effective mitigation is to disable 3DES support or enforce a conservative bound for the amount of data encrypted under one key (and we assume such limit in browsers).

It is possible to extract short secrets using a statistical attack against the biased key stream of the RC4 stream cipher [37]. Although the current state of the art attack still requires a large number of secret repetitions, IETF deprecated RC4 use in TLS [62] and major browsers disabled RC4 support.

Compression oracles (partial secret leakage): A side-channel based on compression was described by Kelsey [48]. If the attacker injects into the plaintext a copy of the secret, the compression should reduce the size of the ciphertext, when compared to injecting random plaintext of the same size. The attacker could observe the size of the ciphertext (CRIME attack [67]) or the time of the transmission (TIME attack [7]) to build an oracle for verifying guesses of the secret. The attacks require secret repetition and partial control over the plaintext. Modern clients disable compression of TLS records, and so does the majority of the servers [64].

Renegotiation and Triple Handshake (integrity): We consider the Renegotiation attack [65] and the Triple Handshake attack [10] as out of scope. The main idea of the attacks is that the messages sent by the client are "spliced" into ongoing communication between the attacker and the server, and the server assumes continuity before and after renegotiation, despite TLS not giving such guarantee. We do not consider Client Authentication and do not test application layer authentication for such behavior.

B. More Detailed Attack Trees

Tests performed by security tools can be also described as attack trees. To illustrate the specific conditions of some attacks, we present an abstraction of the tests for Bleichenbacher's oracle in Figure 5 and its Strong variant in Figure 6, General and Special DROWN attack in Figure 7 and Figure 8, respectively, and the conditions for POODLE-TLS in Figure 9 and for a specific CBC padding oracle in Figure 10.

Some leaf conditions in the trees are represented by subtrees. We list some of them explicitly, namely the requirements for an attacker to mount a protocol version downgrade attack (Figure 11), the conditions indicating the presence of an RSA decryption oracle (Figure 12 and 13), and the tree for fast RSA signature oracle (Figure 14). Other leaf conditions are more intuitive or they are mapped to the outputs of the attack vulnerability testing tools, testssl.sh, TLS-Attacker [80], and the DROWN detection plugin for nmap.

```
GOAL Bleichenbacher's oracle on the server
| 1 The response to any of these client key
    exchanges differs:
  | 1 Correct padding:
      00 02 <random> 00 <TLS version> <PMS>
  | 2 Wrong first two bytes:
      41 17 <random> 00 <TLS version> <PMS>
  | 3 A 0x00 byte in a wrong position:
      00 02 <random> 11 <PMS> 00 11
  | 4 Missing 0x00 byte in the middle:
      00 02 <random> 11 11 11 <PMS>
  | 5 Wrong version number oracle [50]:
      00 02 <random> 00 02 02 <PMS>
```

Fig. 5. A simplified test for general Bleichenbacher's oracle from testssl.sh

```
GOAL Strong Bleichenbacher's oracle on the server
& 1 Bleichenbacher's oracle on the server (Figure 5)
& 2 The client key exchange messages 2, 3, and 4
    invoked at least 2 different server responses
```

Fig. 6. A simplified test for Strong Bleichenbacher's oracle from testssl.sh

```
GOAL Server is vulnerable to General DROWN
| 1 Server supports a vulnerable SSLv2 ciphersuite
     (using DES or a cipher with 40-bit keys)
  | 1 Server offers such ciphersuite (CVE-2016-0800)
  | 2 Server accepts such ciphersuite without
      advertising its support (CVE-2015-3197)
```

Fig. 7. The test for General DROWN according to the detection script (the test is repeated for different application protocols)

```
GOAL Server is vulnerable to Special DROWN
& 1 Server supports SSLv2
& 2 Server has the "extra clear" oracle (it allows
    clear_key_data bytes for non-export ciphers)
```

Fig. 8. The test for Special DROWN according to the detection script

```
GOAL POODLE-TLS padding oracle on the server
| 1 Server does not respond with a Fatal Alert to
    a message with an error on the first byte of the
    padding (the rest of the padding is correct)
```

Fig. 9. The test for a POODLE-TLS padding oracle as seen in TLS-Attacker

```
GOAL CBC padding oracle CVE-2016-2107 on the server
| 1 Server issues a RECORD_OVERFLOW alert
    as a response to a specially crafted message
```

Fig. 10. The test for a CBC padding oracle due to an OpenSSL bug in AES-NI code (CVE-2016-2107) as seen in TLS-Attacker (simplified)

```
GOAL Downgrade to a specific lower protocol version <V>
& 1 At least one of the peers does not support version downgrade mitigation
  | 1 Client does not support RFC 7507 TLS_FALLBACK_SCSV (i.e., the Client does not append
      the ciphersuite to a ClientHello with other than the highest supported TLS version)
  | 2 Server does not support RFC 7507 TLS_FALLBACK_SCSV (i.e., the Server does not check
      for the presence of the ciphersuite in the ClientHello)
& 2 Both Client and Server support a specific lower version <V> of the protocol (with some interesting
    property, e.g., with preferred CBC mode of symmetric encryption, or only supporting RSA key exchange)
  & 1 Server supports the lower protocol version <V>
  & 2 Client supports the lower protocol version <V>
      (e.g., modern web browsers support TLS 1.0, 1.1, 1.2 and possibly 1.3, but neither SSLv2 nor SSLv3)
```

Fig. 11. Attack sub-tree for protocol version downgrade

```
GOAL RSA decryption oracle is available
| 1 Oracle allows feasible decryption
  | 1 Strong Bleichenbacher's oracle on the server (Figure 6)
  | 2 General DROWN
    & 1 Server is vulnerable to General DROWN (Figure 7)
    & 2 Attacker can capture a key exchange in the required format (1 in 900) (assumption)
| 2 Fast RSA decryption oracle (Figure 13)
```

Fig. 12. Attack sub-tree for an RSA decryption oracle (that allows a decryption of key exchange messages)

```
GOAL Fast RSA decryption oracle
| 1 Strong Bleichenbacher's PKCS #1 v1.5 oracle and high performance
  & 1 Strong Bleichenbacher's oracle on the server (Figure 6)
  & 2 Attacker can decrypt before the handshake finishes
      (assumption about the performance of the Server and Attacker to handle many parallel connections)
| 2 Special DROWN
  & 1 Server is vulnerable to Special DROWN (Figure 8)
  & 2 Attacker can capture a key exchange in the required format (1 in 260) (assumption)
```

Fig. 13. Attack sub-tree for a fast RSA decryption oracle (that allows an online decryption)

```
GOAL Fast RSA signature oracle
| 1 Strong Bleichenbacher's PKCS #1 v1.5 oracle and high performance
  & 1 Strong Bleichenbacher's oracle on the server (Figure 6)
  & 2 Attacker can forge the signature before the handshake finishes
      (assumption about the performance of the Server and Attacker to handle many parallel connections)
```

Fig. 14. Attack sub-tree for a fast RSA signature oracle (that allows an online decryption or signature forgery)