Threat-TLS: A Tool for Threat Identification in Weak, Malicious, or Suspicious TLS Connections

Threat-TLS: A Tool for Threat Identification in Weak, Malicious, or Suspicious TLS Connections

Diana Gratiela Berbecaru
Politecnico di Torino
Torino, Italy
diana.berbecaru@polito.it

Antonio Lioy
Politecnico di Torino
Torino, Italy
antonio.lioy@polito.it

ABSTRACT

Various applications running in network-based, mobile, Internet of Things, or embedded systems environments exploit the Transport Layer Security (TLS) protocol to secure communication channels. However, in the last decade, several attacks have been discovered that exploit weaknesses in the protocol specification, the extensions, the cryptographic algorithms, or in the implementation and deployment of TLS-enabled software or libraries. A classical solution to counter TLS attacks on a target is to scan the installed TLS software (via dedicated software or services) and update it with versions that are resistant to attacks. However, an (internal) attacker might even temporarily corrupt the end node so that it becomes vulnerable to TLS attacks. So, the TLS scanning operations should be performed often, wasting resources of the monitored target. We propose a network-based intrusion detection tool named **Threat-TLS**, aimed to individuate weak, suspicious, or malicious TLS connections in intercepted traffic by looking for TLS patterns that contain features exploited to perform attacks, like old protocol versions, weak algorithms, or extensions. We have tested the proposed tool in a testbed environment by exploiting two famous tools, namely Suricata and Zeek, illustrating its performance in detecting some TLS attacks.

CCS CONCEPTS

• **Security and privacy → Security protocols**.

KEYWORDS

Network security, TLS attacks, Malicious, Weak, Suspicious TLS Connections

ACM Reference Format:
Diana Gratiela Berbecaru and Antonio Lioy. 2024. Threat-TLS: A Tool for Threat Identification in Weak, Malicious, or Suspicious TLS Connections. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3664476.3670945

1 INTRODUCTION

TLS protocol provides communication security on the Internet, as it ensures confidentiality, integrity, data and peer authentication, as well as protection from replay and cancellation attacks for the data exchanged between two end points, namely a client and a server. To this end, they negotiate a combination of cryptographic algorithms (symmetric and asymmetric) grouped in cipher suites along with suitable keys, while the X.509 certificates [20] are exploited for authentication purposes, especially for the server side. TLS is also a *flexible* protocol, because for each session the parties may negotiate different cryptographic algorithms in the handshake phase, which is executed when establishing a TLS connection. Flexibility means that the deployed TLS software may be configured with different protocol versions and cipher suites. However, the latest protocol version recommended is TLS 1.3 [38], all the versions prior to version 1.1 are considered insecure, while the TLS 1.2 [37] is still used in a small percentage, though it's highly recommended to adopt and use the latest TLS version. Moreover, several cryptographic algorithms became obsolete and should no longer be used, e.g., MD5 or SHA-1.

This paper introduces **Threat-TLS**, a tool meant to individuate weak, suspicious, or malicious TLS connections. Such connections might be established by attackers to hide and distribute potentially dangerous data content, like malware [22]. For this reason, we name them *malicious* or *suspicious*. Alternatively, *weak* TLS connections could be opened by (legitimate) systems or servers that have been compromised and are prone to TLS attacks, such as systems whose TLS configuration has been changed to use an old TLS version, like TLS 1.0, or an outdated cryptographic algorithm. Lastly, some systems have vulnerabilities that can be exploited in case a specific hardware is used or if a particular TLS software is running. For example, as documented in CVE-2016-6307 [1], "implementation in OpenSSL 1.1.0 before 1.1.0a allocates memory before checking for an excessive length, which might allow remote attackers to cause a denial of service (memory consumption) via crafted TLS messages, related to statem/statem.c and statem/statem_lib.c". So, an attacker might attempt to install or replace an OpenSSL installation with a weak one to set up a DoS attack against a target node. In general, the attacker's final goal is to exploit the TLS weaknesses to obtain sensitive data (like a server's private key kept in a protected area), to set up a malicious MITM to capture the data exchanged between the client and the server, or to mount a DoS attack against a target node running vulnerable TLS software. Since it is difficult to prevent an attacker from (even temporarily) compromising a TLS-enabled node, we concentrate on monitoring and detecting vulnerable or malicious TLS connections that might affect target end nodes.

Motivations of proposal. To detect changes in the node configuration/ installation, one approach could consist of using trusted computing and remote attestation techniques [18] [11]. However,

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6307

ARES 2024, July 30–August 02, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3670945

this solution is not always feasible, as not all devices (especially the embedded or IoT systems) can support them [10] [9]. To detect vulnerable TLS servers, one method currently applied is checking the TLS server's configuration periodically, e.g., daily. Some dedicated tools already exist for this purpose, such as the SSL Server test (powered by Qualys SSL Labs) or TLSAssistant [33], even though more tools exist for this purpose, as indicated in Sect. 2.2. Nevertheless, checking the configuration provides information about the resistance of the TLS server to the known attacks only when the tools are run. That is, they provide a "shot" (picture) of the server's resistance to specific TLS attacks when the TLS scanning tool or service is run.

Another solution is to use TLS-enabled Intrusion Detection and Prevention Systems (IDPS) that inspect data or messages exchanged in the TLS handshake phase, like the X.509 certificates exchanged, the algorithms or the protocol version negotiated, or other connection-related information, such as flow IDs or TLS fingerprints. Some IDPS, like Suricata, Zeek, or Corelight have already rules used to analyze network traffic by processing the certificates, TLS fingerprints, or by analyzing handshake messages intercepted [32]. SELKS from Stamus Network[2] is a famous Suricata-based system for threat hunting.

However, in our approach, we aim to identify patterns of TLS attacks in addition to the ones already considered by the traditional IDPS. Such TLS threat patterns are not (always) trivial to express through specific IDS rules since they are closely related to the internals of TLS attacks described in academic papers or security blogs. We aim to define in the end TLS threat patterns for TLS attacks that could span several connections or data at different layers, though in this work, we start with some simple one(s). As mentioned, some attacks apply only to specific platforms, so the TLS-related alarms should be combined with platform information in an efficient way instead of periodically scanning the devices for all possible vulnerabilities. Thus, we consider TLS vulnerabilities specific to platforms as reported in the Common Vulnerabilities and Exposure (CVE) database[3], namely attacks that could exploit deficiencies in the TLS implementations and could manifest only in case of use of specific platforms or particular software (versions). Once a potential vulnerability is signaled, Threat-TLS checks the effectiveness of the TLS attack alarm through TLS threat validation tools, like Metasploit or Nmap.

We observe that many frameworks or platforms impose requirements on the cryptographic algorithms or protocols to use, but few or even no mechanism or tool is put in place to **check** whether those requirements are fulfilled. For example, the eIDAS Network, a European digital identity infrastructure [7] connecting the electronic identity systems of various countries across European through national eIDAS nodes requests all the eIDAS nodes installed in Member States to use (only) TLS 1.3 in communicating with the other eIDAS nodes [4]. However, we are not aware of any specific mechanism or verification utility recommended or imposed to monitor the TLS protocol version employed by a single eIDAS node, so the operator of each eIDAS node (governmental agency, ministry, contract company, a.s.o.) individually decides the mechanism(s) it could employ to respond to such requirements. For example, the proposed Threat-TLS could be used to verify such requirement and generate an alert in case a different TLS version is negotiated in the connections.

Threat-TLS identifies vulnerable or suspect TLS connections by performing the following operations: 1) monitors the traffic toward and from a target TLS-enabled system (e.g., a server) with one or more IDPS systems that have been configured with rules for weak, suspicious, or malicious TLS connections. To detect such connections, Threat-TLS performs deep analysis on the TLS-related captured traffic, e.g., data exchanged in the TLS handshake including the TLS version negotiated, the ciphersuites, the extensions, and the server certificate. Moreover, since specific vulnerabilities are related to software or operating system versions, or hardware characteristics of the device, Threat-TLS uses context information about the software installed on a target as well as the hardware, and information about common vulnerabilities along with their "criticality" indicator, as obtained from the CVE database. Currently, as shown on MITRE website [4], there are more than 1110 CVE records related to TLS protocol, and probably new ones will appear. 2) in the case of alarms, Threat-TLS checks their effectiveness by employing dedicated TLS threat verification tools and X.509 certificate verification utilities and tools. Finally, it generates reports on the results obtained with the TLS threat verification tools.

Contributions. The main contributions of our work are: 1) definition of *TLS threat patterns* encountered in weak, malicious, or suspicious TLS connections. The patterns are various: they can be algorithm names, X.509 certificates, protocol extensions, protocol versions, or even string of bits that could be exploited to mount TLS attacks or create vulnerable TLS connections. We individuate *some* common patterns and express them in a format recognized by network analyzers or IDSs (such as Suricata, Zeek, Wireshark, ...) to identify vulnerable or malicious TLS connections. 2) design and implementation of the Threat-TLS tool exploiting the patterns we have specified so far, indicating the tools that may be exploited in each building block. 3) evaluation of Threat-TLS performance in an experimental testbed.

Organization. The paper is organized as follows: Section 2 describes the related work on TLS attacks and TLS monitoring tools, Section 3 describes the design of the proposed Threat-TLS tool, Section 4 describes the implementation performed so far and the results obtained. Finally, Section 5 resumes the conclusions and indicates future work.

2 BACKGROUND AND RELATED WORK

2.1 TLS attacks

If we look at the TLS protocol through a magnifying glass, we observe that it is composed of several elements (base blocks) that can carry or potentially introduce different vulnerabilities. In fact, as with its predecessor SSL (Secure Sockets Layer) protocol, the TLS protocol has also been subject, since its creation, to continuous investigation to individuate attacks. Any vulnerability in each of these base blocks can compromise the TLS protocol as a whole. Some TLS attacks exploited deficiencies in the protocol specification (especially in the SSL 2.0 and SSL 3.0), others have exploited

[2] https://www.stamus-networks.com/selks
[3] https://cve.mitre.org
[4] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TLS

weak cryptographic algorithms (like RSA, or MD5), the cipher block chaining (CBC) mode of application, the compression, or software errors in the protocol implementation. Examples of such attacks are the Bleichenbacher attack [14], DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) [3], transcript collision attack [13], Heartbleed [23], Vaudenay et al. attack [19], CRIME (Compression Ratio Info-leak Made Easy) [29], BREACH (Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) [26], BEAST (Browser Exploit Against SSL/TLS) [27], POODLE (Padding Oracle On Downgraded Legacy Encryption) [35], Lucky 13 [2], Logjam [1], ROBOT (Return Of Bleichenbacher's Oracle Threat) [15], ROCA (Return of Coppersmith's Attack) [36], or truncating attack [40] [5]. The latest TLS 1.3 protocol has been also investigated, and some attacks have been documented [31] [28] [24]. Other attacks aim to manipulate the configuration of the TLS server or the client, e.g., changing the X.509 certificate and the corresponding key on the server side or inserting a malicious root Certification Authority (CA)certificate in the list of trusted anchors at the client side. Such attacks can occur after the TLS-enabled software has been installed or deployed on an end node. An Adversary-in-the-Middle (AitM) attacker could exploit self-signed or malformed X.509 certificates [41] combined with other techniques (e.g., social engineering) to set up spoofed websites looking similar to the legitimate ones but whose purpose instead is to steal sensitive data, like authentication credentials.

Since the TLS vulnerabilities and attacks have exploded in the last ten years some works have proposed a classification for them. For example, Meyer and Schwenk [34] divided the attacks into four groups: a) attacks on the TLS Handshake protocol, b) attacks on the TLS Record and Application Data Protocols; c) attacks on the TLS Public key Infrastructure, and d) various other attacks.

Another classification [8] groups the attacks as follows: 1) attacks to SSL/TLS protocol functionality - due to flaws in the specification, 2) attacks to SSL/TLS library implementations - caused by errors in the TLS code, 3) cryptography attacks - due to the use of old or outdated cryptographic algorithms, and 4) attacks to SSL/TLS configuration - target the configurable parameters in any TLS installation (either on the client side or the server side) such as server certificate and client certificate (if used), or intermediate and root CA certificates. A well-detailed classification of various TLS attacks and their evolution is given in [17].

2.2 TLS Scanning and TLS monitoring tools

Several TLS scanning tools can be employed to scan the TLS configuration of specific targets, either when needed or periodically. Examples of these tools are: SSL Server test powered by Qualys SSL Labs [5], SSL Checker [6], TLS Assistant [7], SSL/TLS Scanner [8], SSLyze [9], TLSSLed [10], SiteLock [11], Sensu [12], or TrackSSL [13].

[5]https://www.ssllabs.com/ssltest
[6]https://comodosslstore.com/ssltools/ssl-checker.php
[7]https://st.fbk.eu/tools/TLSAssistant/
[8]https://pentest-tools.com/network-vulnerability-scanning/ssl-tls-scanner
[9]hackertarget.com/ssl-check/
[10]http://blog.taddong.com/2013/02/tlssled-v13.html
[11]https://www.one.com/en/website-security/sitelock
[12]https://sensu.io/blog/tls-monitoring
[13]https://trackssl.com/tls-certificate-monitoring/

The TLS monitoring tools instead are often derived or based on powerful tools used for intrusion detection. Suricata is one such tool, whose first version of the software was released in 2009, in agreement with the management of the Open Information Security Foundation (OISF), a community-run non-profit foundation organized to build an IDS/IPS (IDPS) engine of new generation. Besides being completely open source, the other peculiarities that make Suricata a prominent software are the multithreading support and the ability to analyze network traffic even offline, using pcap. A fundamental part of the use of this IDPS is the management of the rules. In most cases, users exploit the current set of rules within Suricata. However, it is possible to create custom rules to integrate or replace those already present, and some of them have been indeed tailored for TLS. Since version 2.0, Suricata has also added support for Lua scripting. The idea is to be able to decide if an alert is matching based on the return of a Lua script.

Another worth-mentioning IDS is Zeek [14], which can be used along with other mechanisms, such as JA3 and JA3S [15] (supported also in Suricata), to profile the TLS implementations and derive alerts. JA3 and JA3S gather information from fields that are not encrypted in the Client Hello and Server Hello TLS handshake messages, such as the version of SSL/TLS being used, the ciphers supported, extensions available, and concatenates them to generate a fingerprint of the SSL/TLS client and server.

3 THREAT-TLS DESIGN

This section provides details of the Threat-TLS architecture. The tool is composed of four main blocks, named `Capture`, `Analyze`, `Validate`, and `Report` (see Fig. 1).

The `Capture` block holds the IDPS and network analyzers used for intercepting the network traffic and looking for suspicious, malicious, or weak TLS threat patterns according to specific rules expressed in a format or scripting language. In this block, we consider two of the most widely used software tools widely used to set up an IDPS, that is Suricata [16] and Zeek [17]. Moreover, we consider also Wireshark [18], a very famous packet analyzer, which allows to perform in-depth inspections on the intercepted network traffic.

The `Analyze` block is in charge of processing in the first place the files generated by the IDPS(s) and packet analyzer(s). Every time there is a match between the intercepted network traffic against (one of the) rules that have been configured for the IDPS, the Alert Generator creates an an alert string in the following format:

```
|TLS Threat| #TLS threat type #TLS pattern (intercepted)
```

For example, a possible alert string indicating an outdated TLS version is:

```
|TLS Threat| #TLS version #TLSv1.0
```

The alert string is then managed by an Alert Manager, which investigates whether the related alert applies to the monitored target device and whether it has been flagged critical. Some alerts applies to all types of devices, while other alerts apply only to

[14]https://docs.zeek.org/en/master/monitoring.html
[15]https://github.com/salesforce/ja3
[16]https://suricata.io
[17]https://zeek.org
[18]https://wireshark.org/

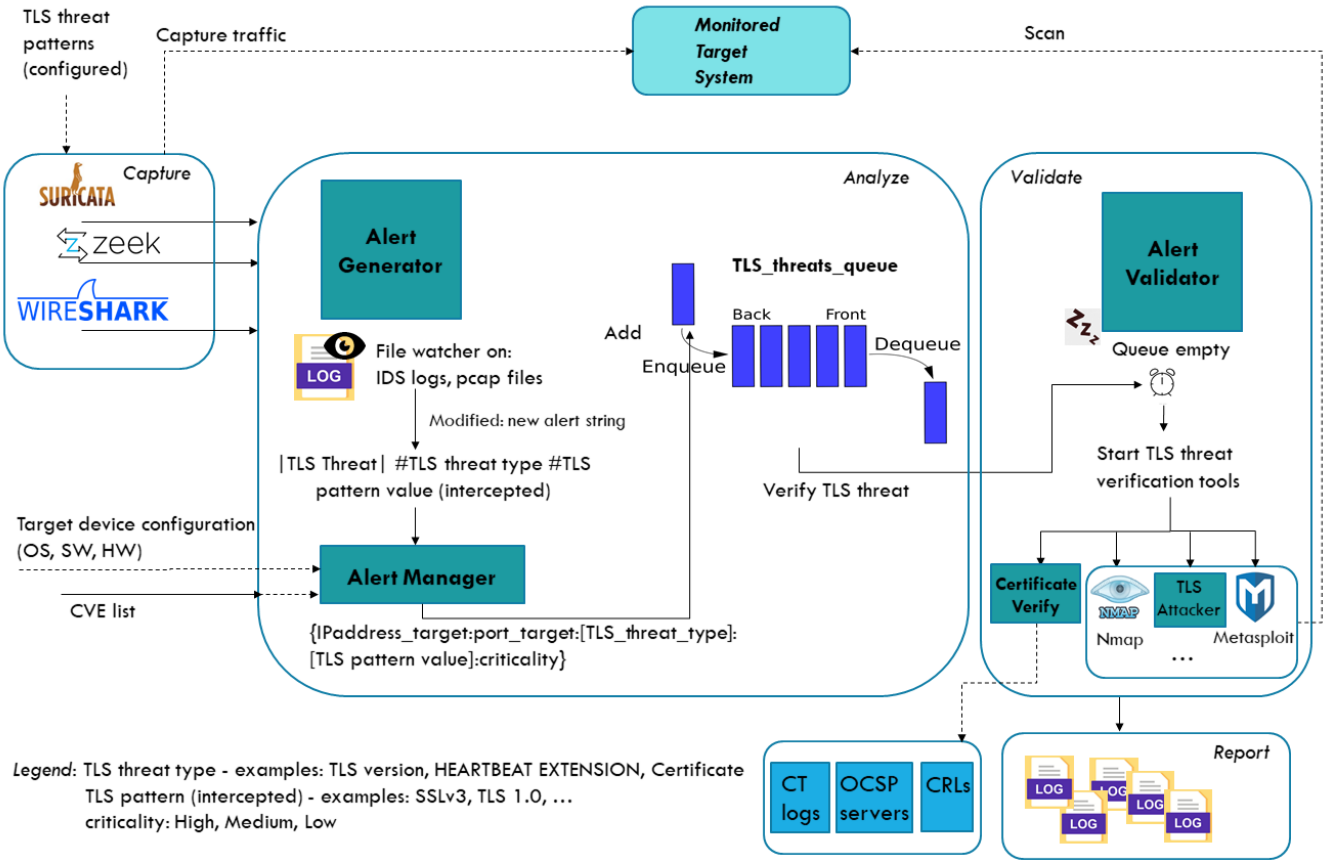

Figure 1: Threat-TLS Architecture.

specific operating systems or TLS libraries. For example, the CVE-2016-2183, a vulnerability in DES and triple DES ciphers, is related to the Sweet32 attack [12] on any platform. Assuming an alert applies to the monitored system, the Alert Manager produces events that are added to a queue, which is monitored by the Alert Validator in the `Validate` block. This module is thought to react every time a new TLS threat item is inserted into the *TLS_Threats_queue*. The TLS threat items have the following format:

```
{IPaddress_target:port_target:[TLS_threat_type]:
[TLS pattern]:criticality}
```

For the TLS threat example considered above and the monitored system running the TLS-enabled service on port 443 at 10.0.2.8 IP address, the threat item is:

```
{10.0.2.8:443:TLS version:TLSv1.0:criticality}
```

Every TLS threat item is validated by launching one or more TLS threat verification tools, such as Nmap [19], Metasploit [20], or TLS Attacker [21], which scans the indicated target system. The results of the scanning operations are saved in (log) files that are part of the `Report` block.

In the `Validator` block we place a Certificate Verify module, which is in charge of deep inspection and verification of the X.509 certificates. It is already known that if certificate validation is performed incompletely or incorrectly, some attacks may occur, like the AiTM one. Many implementations (especially on the client side) ignore the revoked certificates [25], but also process erroneously some extensions [6]. The Certificate Verify module is responsible for detecting erroneous, malformed, or malicious X.509v3 certificates, by relying also on external sources, including the Certificate Transparency logs, the OCSP responders [39], or the Certification Authorities (CAs) to download the corresponding CRLs (Certificate Revocation Lists) [20].

4 THREAT-TLS IMPLEMENTATION DETAILS

The building blocks of Threat-TLS have been implemented in Python 3.10.6, through dedicated scripts developed for the Alert Generator, Alert Manager, Alert Validator, and Certificate Verify. The Alert Generator (along with the Alert Manager) and the Validator are currently developed as threads in a producer-consumer multi-threaded model. In this model, the producer inserts a produced object into a shared dictionary, so that the consumer can retrieve it and perform further processing. A dictionary in Python is a collection that is ordered and changeable and

[19] https://nmap.org
[20] https://www.metasploit.com
[21] https://github.com/tls-attacker/TLS-Attacker

Table 1: Threat-TLS detects weak, malicious, and suspicious connections based on specific TLS threat patterns (whose corresponding rules are configured in IDS). For each pattern, we show possible TLS attacks exploiting the pattern, and the TLS threat verification tools we have exploited to verify whether the threat applies.

TLS threat patterns	Possible TLS Attack	Specific HW/SW	TLS threat verification tools
Heartbeat extension	Heartbleed		Metasploit, Nmap, TestSSL, TLS-Attacker
Compression enabled	CRIME		TestSSL
SSLv2 enabled	DROWN		TestSSL, TLS-Attacker
RSA Ciphersuites	Bleichenbacher		TestSSL, TLS-Attacker, Metasploit
RSA Ciphersuites	ROBOT	As indicated in [16]	TestSSL, TLS-Attacker, Metasploit
DES/3DES Ciphersuite	Sweet32		TestSSL
Export Ciphersuite	LogJam		TestSSL, Nmap
CBC mode (ciphersuite)	Lucky13		TestSSL
RSA_CBC Ciphersuite	Padding Oracle Attack		TLS-Attacker
POODLE	SSLv3 enabled and RSA_CBC Ciphersuite		TestSSL, Nmap, TLS-Attacker
Certificate Self Signed/Malformed/Revoked/ Compromised	MITM		Certificate Verify
TLSv1, TLSv1.1, TLS1.2	Ticketbleed	As indicated in [21]	Nmap
SSLv3, TLSv1, TLSv1.1, TLSv1.2	Change Cipher Spec injection		Metasploit, Nmap
RSA Ciphersuites	ROCA	BitLocker with TPM 1.2, YubiKey 4 (before 4.3.5), as indicated in CVE-2017-15361	Nmap

cannot have duplicate members. In the Alert Generator, the producer thread is represented by a Watchdog object that monitors an object called **Handler**. When the **Handler** object is created, the logfile variable inside it is initialized with the path of the IDS log file used, that is `/var/log/fast.log` (for Suricata) and `/usr/local/zeek/log/current` (for Zeek). When the IDS intercepts the network traffic (according to the rules) and modifies the log file, the Watchdog starts a function to read the new line in the log file. To read the log file the Watchdog calls 2 different functions:

(1) file_reader_suricata
(2) file_reader_zeek

The called function is different based on the IDS used because the 2 IDS use different kinds of log file. The Zeek log file is a table where for each field the IDS writes the corresponding value. The Suricata log file is written by a custom message for the **msg** keyword used in the Suricata rules. The difference for the two functions is only on how they extract the data from the logfile, but the goal for both functions is the same: produce an object in the dictionary .

The **TLS_threats_queue** is a dictionary containing key-value structures, where:

- the key is the IP address of the target (as passed from the Alert Manager)
- the value is an array with all the vulnerabilities found for a TLS connection from that IP address

If the TLS_threats_queue reaches a max value stored in the **MAX_NUM** variable, the producer thread waits until the queue is consumed. The Consumer thread is implemented by means of a function called **verify_vulnerability**. This thread waits until an item is inserted into the TLS_threats_queue. When this happens, the thread wakes up and retrieves an item from the TLS_threats_queue. Depending on the TLS threat type (e.g., TLS version) and the value of the TLS attack pattern (e.g., TLSv1.0) indicated the item, the thread starts one or more TLS threat verification tools against the IP address represented by the key of the item. The tools open TLS connections configured with parameters specific for the indicated threat. Finally, a report is generated with the verification results produced by each TLS threat verification tool.

Independently of the IDS used, we have implemented a tool for certificate verification, called `Certificate Verify` in Fig. 1. Thus, Threat-TLS may verify the target certificate, namely the one intercepted and passed from the `Analyze` block. The Alert Generator puts the |**CERTIFICATE**| into the TLS_threat_queue along with its value. The Alert Validator thread takes the certificate (value) and starts its verification by performing several checks, including the verification of the Issuer and Subject fields (e.g., to detect self-signed certificates), of the Subject Alternative name extension (to detect certificates issued for wrong domain names), of the validity period (to detect expired certificates). Moreover, the module checks the revocation status by exploiting OCSP [39] and CRL mechanisms [20]. Finally, it verifies the SCTs (Signed Certificate Timestamps) in the certificate [30] by exploiting the CT system.

4.1 Defining TLS threat patterns

The TLS protocol can be divided into two main phases, the TLS handshake phase and the TLS application data transfer. Both phases exploit the TLS Record protocol to protect the messages exchanged.

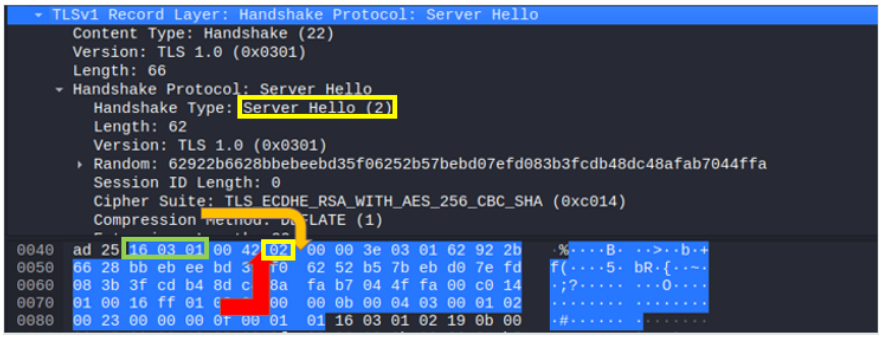

Figure 2: Analyzing TLS Server Hello message to express Suricata rule for Heartbeat extension detection.

Additionally, the alert messages in the TLS Alert protocol are very important, since they are used to close a legitimate connection once the data has been transferred or to indicate errors in the TLS connection. In the past, the alert messages have been exploited in some TLS attacks because the attacker could deduce part of the information based on the alert message(s) returned. Thus, TLS threat patterns for inspecting the alert messages could be defined.

To write the rules for suspect, malicious, or weak connections in Suricata, it is thus necessary to individuate TLS threat patterns in the intercepted traffic and generate an alert.

In Table 1, we indicate patterns for some TLS attacks, as well as the attack tools started by the Alert Validator when a TLS threat pattern is found.

These patterns are been intercepted in different ways in the two IDS we have considered. For Suricata a file called `patterns.rules` is written and then enabled in the Suricata's yaml file. For Zeek the file `/usr/local/zeek/share/zeek/base/protocols/ssl` is modified to add some messages in the log file.

We explain next how a TLS threat pattern can be expressed in a format recognized by the ID(P)S we have considered. We will take as example the Heartbeat extension, because this could be exploited to mount the Heartbleed attack. The TLS threat pattern 'Heartbeat extension' can be expressed with the following rule for Suricata:

```
alert tls any any <> any any (msg:"|VULNERABILITY|
#HEARTBEAT EXTENSION# $1.0$";flow:from_server;
content:"|16 03 01|";content:"|02|"; distance:2;
within:3;content:"|00 0f|";content:"|01|"; distance:2;
within:4; sid:11;)
```

Explanation. The above rule is written to intercept the Heartbeat extension for TLS v1.0. However, by changing the content with the appropriate hexadecimal value is still possible to intercept it for other TLS versions, i.e., SSL v3, TLS v1.1, TLS v1.2. This rule can be divided into 2 different Suricata rules joined with an AND logic operation. The first (part of the) rule indicates the `Server Hello` message in the TLS handshake and it is used also for the others rules. This rule verifies the bytes indicating the Server Hello message (02) that must start after 2 bytes (distance rule) and end within 3 bytes (within rule). The first rule is shown in Fig. 2 where the yellow indicates the Server Hello byte, the red arrow indicates the distance keyword and the orange arrow indicates the within keyword. The second part of the rule checks if the Heartbeat extension in the `Server Hello` message is enabled. The Heartbeat extension is represented in the TLS message structure by 2 bytes **00 0f**. After these 2 bytes, there are 2 bytes representing the length of the extension and then the byte which represents the extension enabled **01**, as shown on the last raw in Fig. 2.

4.2 Validating Threat-TLS implementation

To test Threat-TLS we have set up an experimental testbed composed of:

- a TLS Server (running Apache web server [22] and OpenSSL library [23])
- a TLS Client (running TLS-enabled web browser, like Firefox)
- a monitoring machine on which we have installed Threat-TLS and the exploited tools: Ettercap [24] (to deviate the traffic exchanged between the client and server through the monitoring machine), Suricata, Zeek (with the configured TLS threat patterns), Wireshark, and the Threat-TLS scripts for Alert Generator, Manager, Validator, and Certificate Verify, as well as the TLS threat validation utilities, i.e. Metasploit, TestSSL, TLS-Attacker, and Nmap.

We simulated different test cases in which an attacker managed to compromise the (monitored) TLS server. For each test case, we have installed on the server OpenSSL versions and Apache2 that are weak or vulnerable to attacks, like Apache2 v2.51.4 and v2.47.2. Table 2 summarizes the vulnerable OpenSSL versions installed and the TLS attacks associated with each version.

After installing the specific vulnerable OpenSSL library on the server machine, the client (web browser) connects to the vulnerable TLS server, triggering thus the alarms on the monitoring machine. For each vulnerability, the measurement was repeated 20 times. The test results (obtained with Suricata and Zeek) are shown in Fig. 3. This figure illustrates the time required by Threat-TLS to verify the indicated threat/attack by running the TLS attack tools for that specific vulnerability. The Heartbleed vulnerability takes more time than the others because Threat-TLS runs four TLS threat verification tools to verify it, namely TestSSL, Nmap, TLS-Attacker, Metasploit. Moreover, Metasploit first checks whether the machine

[22] https://httpd.apache.org/
[23] openssl.org
[24] https://www.ettercap-project.org/index.html

Table 2: TLS attacks associated with (old) Openssl versions used in the experimental tests.

Openssl Version	TLS Attack	CVE Reference
OpenSSL 0.9.7	Bleichenbacher	CVE-2012-0884
	CCSInjection	CVE-2014-0224
	POODLE	CVE-2014-3566
OpenSSL 1.0.1c	Heartbleed	CVE-2014-0160
	CCSInjection	CVE-2014-0224
	Lucky13	CVE-2013-0169
	POODLE	CVE-2014-3566
OpenSSL 1.0.2	Padding Oracle Attack	CVE-2016-2107
	Sweet32	CVE-2016-2183
	DROWN	CVE-2016-0800

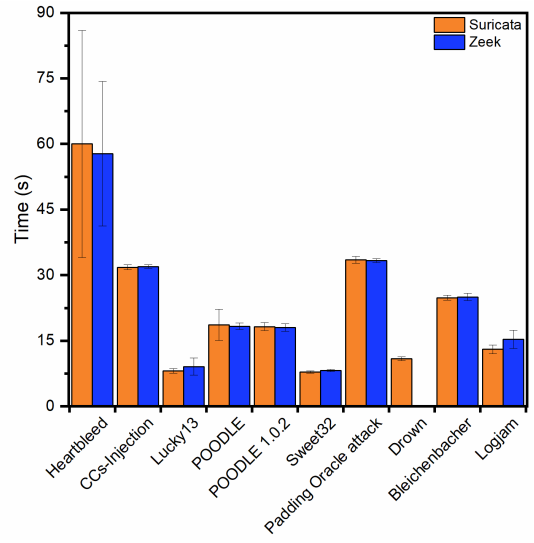

Figure 3: Test results for detection and verification of single TLS attacks.

is vulnerable, then it tries to retrieve the server's private key by using the Heartbleed exploit integrated in Metasploit.

In all tests, the time measurement starts when the IDS intercepts the (vulnerable) packets on the network based on the rules we have configured for the corresponding threat and finishes when all the TLS threat verification tools complete their execution. Fig. 3 shows that both Suricata and Zeek require about the same amount of time for verification, the only difference is that Zeek can't sniff the SSLv2 traffic. In fact, as shown in Fig. 3 it can not sniff traffic for the DROWN vulnerability.

Next, we have performed tests by considering a scenario in which one TLS connection is subject to multiple possible TLS threats, so we have measured the time spent to verify all the attacks (with one or more TLS threat verification tools, depending on the attack). In this testing scenario, Threat-TLS launches all the TLS threat verification tools for each possible vulnerability found in the packet. Each measurement has been performed 10 times. Fig. 4 shows the time spent by the tool for a TLS connection established with three vulnerable

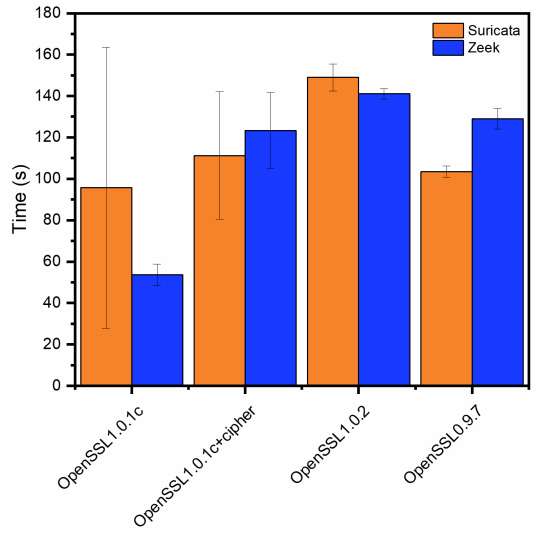

Figure 4: Time spent to verify all the TLS attacks applying to a vulnerable TLS connection. TLS threats for values on the X-axis: OpenSSL 1.0.1c: Heartbeat:YES, TLS Version: 1.2, Cipher:ECDHE-RSA-AES256-GCM-SHA384. OpenSSL 1.0.1c+cipher: Heartbeat:YES, TLS Version: 1.2, Cipher:DES-CBC3-SHA. OpenSSL 1.0.2: Heartbeat:NO, TLS Version: 1.2, Cipher:DES-CBC3-SHA. OpenSSL 0.9.7: Heartbeat:NO, TLS Version: 1.2, Cipher:DES-CBC3-SHA.

libraries, namely OpenSSL 1.0.1c, OpenSSL 1.0.2, and OpenSSL 0.9.7. Table 3 indicates, for each of the three vulnerable OpenSSL versions we have installed, the potential TLS threats derived from the traffic intercepted when establishing a TLS connection from a client to the server running the vulnerable OpenSSL library. By using the TLS threat patterns in Table 1, all TLS attacks are verified with the TLS verifications tools and the total time spent on verifying them is measured. The test with Openssl 1.0.1c occurs twice because in the second scenario a vulnerable cipher (containing CBC mode) is proposed in the TLS connection, thus more TLS attack tools are started.

5 CONCLUSIONS

To detect weak, malicious, or suspicious TLS connections, we propose Threat-TLS, a tool actively looking for TLS threat patterns in the intercepted traffic. We defined basic TLS threat patterns for a selected set of TLS attacks, we deployed them in Suricata and Zeek, and we tested their effectiveness with TLS threat verification tools installed in an experimental testbed. We show that it is possible to identify the attacks in a relatively short time. Future work will address the definition of more complex TLS threat patterns, as well as the enhancement of the Alert Manager component, which combines the TLS threats generated with the CVE list obtained from MITRE, to generate customized alert(s) that consider the specific configuration of a monitored TLS-enabled device.

Table 3: For the vulnerable OpenSSL versions exploited in tests, we show the attacks successfully tested with the TLS threat verification tools indicated, and the ones not completed due to HW/SW requirements or not supported by the attack tools.

No.	IDS	OpenSSL Version	Potential attacks	TLS attack successfully tested with TLS threat verification tool	TLS attack not successfully tested
(1)	Suricata	OpenSSL 1.0.1c	11	Heartbleed with Metasploit, Heartbleed with Nmap, Heartbleed with TLS-Attacker, CCS-Injection with Metasploit, CCS-Injection with Nmap	Heartbleed with TestSSL, Bleichenbacher with TLS-Attacker, Bleichenbacher with Metasploit, Robot with TestSSL, ROCA with Nmap, Ticketbleed with Nmap
(2)	Suricata	OpenSSL 1.0.1c+cipher	16	Heartbleed with Metasploit, Heartbleed with Nmap, CCS-Injection with Metasploit, CCS-Injection with Nmap, Lucky13 with TestSSL	Heartbleed with TLS-Attacker, Heartbleed with TestSSL, Bleichenbacher with TLS-Attacker, Bleichenbacher with Metasploit, Robot with TestSSL, ROCA with Nmap, Ticketbleed with Nmap, POODLE with Nmap, POODLE with TestSSL, POODLE with TLS-Attacker
(3)	Suricata	OpenSSL 1.0.2	12	Padding Oracle Attack with TLS-Attacker	Bleichenbacher with TLS-Attacker, Bleichenbacher with Metasploit, CCS-Injection with Metasploit, CCS-Injection with Nmap, Robot with TestSSL, ROCA with Nmap, Ticketbleed with Nmap, Lucky13 with TestSSL, POODLE with Nmap, POODLE with TestSSL, POODLE with TLS-Attacker
(4)	Suricata	OpenSSL 0.9.7	12	Bleichenbacher with TLS-Attacker, Bleichenbacher with Metasploit, CCS-Injection with Metasploit, CCS-Injection with Nmap, Robot with TestSSL	ROCA with Nmap, Ticketbleed with Nmap, Lucky13 with TestSSL, POODLE with Nmap, POODLE with TestSSL, POODLE with TLS-Attacker, Padding Oracle Attack with TLS-Attacker
(5)	Zeek	OpenSSL 1.0.1c	11	same as (1)	same as (1)
(6)	Zeek	OpenSSL 1.0.1c+cipher	16	same as (2), but we obtained more false negatives	same as (2)
(7)	Zeek	OpenSSL 1.0.2	12	same as (3)	same as (3)
(8)	Zeek	OpenSSL 0.9.7	12	same as (4)	same as (4)

ACKNOWLEDGMENTS

Dr. Diana Gratiela Berbecaru carried out this study within the Ministerial Decree no. 1062/2021 and received funding from the FSE REACT-EU - PON Ricerca e Innovazione 2014-2020. Authors would like to thank Giuseppe Petraglia for performing part of the measurements described in this work during preparation of his graduation thesis at Politecnico di Torino. This manuscript reflects only the authors' views, findings, conclusions, and opinions, neither the European Union nor the European Commission can be considered responsible for them.

REFERENCES

[1] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. 2015. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 5–17. https://doi.org/10.1145/2810103.2813707

[2] Nadhem J. Al Fardan and Kenneth G. Paterson. 2013. Lucky Thirteen: Breaking the TLS and DTLS Record Protocols. In *Proceedings of 2013 IEEE Symposium on Security and Privacy*. 526–540. https://doi.org/10.1109/SP.2013.42

[3] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J. Alex Halderman, Viktor Dukhovni, Emilia Käsper, Shaanan Cohney, Susanne Engels, Christof Paar, and Yuval Shavitt. 2016. DROWN: Breaking TLS Using SSLv2. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 689–706. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/aviram

[4] Diana Berbecaru, Andrea Atzeni, Marco De Benedictis, and Paolo Smiraglia. 2017. Towards Stronger Data Security in an eID Management Infrastructure. In *Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2017)*. 391–395. https://doi.org/10.1109/PDP.2017.90

[5] Diana Berbecaru and Antonio Lioy. 2007. On the Robustness of Applications Based on the SSL and TLS Security Protocols. In *Public Key Infrastructure*, Javier Lopez, Pierangela Samarati, and Josep L. Ferrer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 248–264. https://doi.org/10.1007/978-3-540-73408-6_18

[6] Diana Gratiela Berbecaru and Antonio Lioy. 2023. An Evaluation of X.509 Certificate Revocation and Related Privacy Issues in the Web PKI Ecosystem. *IEEE*

Access 11 (2023), 79156–79175. https://doi.org/10.1109/ACCESS.2023.3299357

[7] Diana Gratiela Berbecaru, Antonio Lioy, and Cesare Cameroni. 2021. On Enabling Additional Natural Person and Domain-Specific Attributes in the eIDAS Network. *IEEE Access* 9 (2021), 134096–134121. https://doi.org/10.1109/ACCESS.2021.3115853

[8] Diana Gratiela Berbecaru and Giuseppe Petraglia. 2023. TLS-Monitor: A Monitor for TLS Attacks. In *Proceedings of the 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC 2023)*. 1–6. https://doi.org/10.1109/CCNC51644.2023.10059989

[9] Diana Gratiela Berbecaru and Lorenzo Pintaldi. 2023. Exploiting Emercoin Blockchain and Trusted Computing for IoT Scenarios: A Practical Approach. In *Proceedings of 2023 IEEE Symposium on Computers and Communications (ISCC 2023)*. 771–776. https://doi.org/10.1109/ISCC58397.2023.10217961

[10] Diana Gratiela Berbecaru and Silvia Sisinni. 2022. Counteracting software integrity attacks on IoT devices with remote attestation: a prototype. In *Proceedings of the 26th International Conference on System Theory, Control and Computing (ICSTCC 2022)*. 380–385. https://doi.org/10.1109/ICSTCC55426.2022.9931765

[11] Diana Gratiela Berbecaru, Silvia Sisinni, Antonio Lioy, Benoit Rat, Davide Margaria, and Andrea Vesco. 2023. Mitigating Software Integrity Attacks With Trusted Computing in a Time Distribution Network. *IEEE Access* 11 (2023), 50510–50527. https://doi.org/10.1109/ACCESS.2023.3276476

[12] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 456–467. https://doi.org/10.1145/2976749.2978423

[13] Karthikeyan Bhargavan and Gaëtan Leurent. 2016. Transcript Collision Attacks: Breaking Authentication in TLS, IKE and SSH. In *Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS 2016)*. San Diego, CA, USA.

[14] Daniel Bleichenbacher. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Proceedings of Advances in Cryptology - CRYPTO '98*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[15] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 817–849. https://www.usenix.org/conference/usenixsecurity18/presentation/bock

[16] Hanno Böck, Juraj Somorovsky, and Craig Young. 2018. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 817–849.

[17] Colin Boyd, Anish Mathuria, and Douglas Stebila. 2019. Protocols for Authentication and Key Establishment. (Nov. 2019). https://doi.org/10.1007/978-3-662-58146-9

[18] Enrico Bravi, Diana Gratiela Berbecaru, and Antonio Lioy. 2023. A Flexible Trust Manager for Remote Attestation in Heterogeneous Critical Infrastructures. In *Proceedings of the 2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2023)*. 91–98. https://doi.org/10.1109/CloudCom59040.2023.00027

[19] Brice Canvel, Alain Hiltgen, Serge Vaudenay, and Martin Vuagnoux. 2003. Password Interception in a SSL/TLS Channel. In *Proceedings of Advances in Cryptology - CRYPTO 2003*. Springer Berlin Heidelberg, Berlin, Heidelberg, 583–599.

[20] David Cooper, Stefan Santesson, Sharon Boeyen Stephen Farrell, Russell Housley, and William Polk. 2008. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. (May 2008). https://tools.ietf.org/pdf/rfc5280.pdf

[21] CVE-2016-9244. [n. d.]. Ticketbleed Bug. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9244

[22] Michael J. de Lucia and Chase Cotton. 2019. Detection of Encrypted Malicious Network Traffic using Machine Learning. In *Proceedings of the 2019 IEEE Military Communications Conference (MILCOM 2019)*. 1–6. https://doi.org/10.1109/MILCOM47813.2019.9020856

[23] Heartbleed documentation. [n. d.]. The Heartbleed Bug. https://heartbleed.com

[24] Nir Drucker and Shay Gueron. 2021. Selfie: reflections on TLS 1.3 with PSK. *Journal of Cryptology* 34, 27 (2021). https://doi.org/10.1007/s00145-021-09387-y

[25] Jens Friess, Haya Schulmann, and Michael Waidner. 2023. Revocation Speedrun: How the WebPKI Copes with Fraudulent Certificates. *Proc. ACM Netw.* 1, CoNEXT3, Article 26 (nov 2023), 20 pages. https://doi.org/10.1145/3629148

[26] Yoel Gluck, Neal Harris, and Angelo Prado. [n. d.]. BREACH: reviving the CRIME attack. ([n. d.]). http://breachattack.com/

[27] Matthew Green. 2011. A Diversion: BEAST Attack on TLS/SSL Encryption. (Sept. 2011). https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/

[28] Tibor Jager, Jörg Schwenk, and Juraj Somorovsky. 2015. 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* (Denver, Colorado, USA) *(CCS '15)*. Association for Computing Machinery, New York, NY, USA, 1185–1196. https://doi.org/10.1145/2810103.2813657

[29] Rizzo Juliano and Duong Thai. 2012. The CRIME attack. (Sept. 2012). https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4929

[30] B. Laurie, E. Messeri, and R. Stradling. 2021. Certificate Transparency Version 2.0. (Dec. 2021). https://www.rfc-editor.org/rfc/rfc9162.html

[31] Sangtae Lee, Youngjoo Shin, and Junbeom Hur. 2020. Return of Version Downgrade Attack in the Era of TLS 1.3. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies* (Barcelona, Spain) *(CoNEXT '20)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/3386367.3431310

[32] Peter Manev and Eric Leblond. 2020. Hunting Threats That Use Encrypted Network Traffic. (May 2020). https://suricata.io/wp-content/uploads/2020/05/suricata-and-tls.pdf

[33] Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. 2019. Lost in TLS? No More! Assisted Deployment of Secure TLS Configurations. In *Data and Applications Security and Privacy XXXIII*, Simon N. Foley (Ed.). Springer International Publishing, Cham, 201–220.

[34] Christopher Meyer and Jörg Schwenk. 2014. SoK: Lessons Learned from SSL/TLS Attacks. In *Information Security Applications, WISA 2013. Lecture Notes in Computer Science(), vol 8267*. Springer International Publishing, Cham, 189–209.

[35] Bodo Moller, Thai Duong, and Krzysztof Kotowicz. 2014. This POODLE Bites: Exploiting the SSL 3.0 Fallback. (Oct. 2014). https://security.googleblog.com/2014/10/this-poodle-bites-exploiting-ssl-30.html

[36] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. 2017. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (, Dallas, Texas, USA,) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1631–1648. https://doi.org/10.1145/3133956.3133969

[37] Eric Rescorla. 2008. The Transport Layer Security (TLS) Protocol Version 1.2. (April 2008). https://tools.ietf.org/pdf/rfc5246.pdf

[38] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. (Aug. 2018). https://tools.ietf.org/pdf/rfc8446.pdf

[39] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. 2013. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. (June 2013). https://tools.ietf.org/pdf/rfc6960.pdf

[40] Ben Smyth and Alfredo Pironti. 2013. Truncating TLS connections to violate beliefs in web applications. In *Proceedings of the 7th USENIX Conference on Offensive Technologies* (Washigton, D.C.) *(WOOT'13)*. USENIX Association, USA, 1.

[41] Shaun Stricot-Tarboton, Sivadon Chaisiri, and Ryan K.L. Ko. 2016. Taxonomy of Man-in-the-Middle Attacks on HTTPS. In *Proceedings of 2016 IEEE Trustcom/BigDataSE/ISPA*. 527–534. https://doi.org/10.1109/TrustCom.2016.0106