Preprints.org

Article

Not peer-reviewed version

Detecting TLS Protocol Anomalies Through Network Monitoring and Compliance Tools

Diana Gratiela Berbecaru * and Marco De Santo

Posted Date: 2 December 2025

doi: 10.20944/preprints202512.0139.v1

Keywords: TLS vulnerabilities; network monitoring; Suricata rules; threat detection; TLS-Anvil

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Detecting TLS Protocol Anomalies Through Network Monitoring and Compliance Tools

Diana Gratiela Berbecaru * and Marco De Santo

Politecnico di Torino, Department of Control and Computer Engineering, Corso Duca degli Abruzzi 24, 10129, Torino (ITALY)
* Correspondence: diana.berbecaru@polito.it

Abstract

The Transport Layer Security (TLS) protocol is widely used nowadays to create secure communications over TCP/IP networks. Its purpose is to ensure confidentiality, authentication, and data integrity for messages exchanged between two endpoints. In order to facilitate its integration into widely used applications, the protocol is typically implemented through libraries, such as OpenSSL, BoringSSL, LibreSSL, WolfSSL, NSS, or mbedTLS. These libraries encompass functions that execute the specialized TLS handshake required for channel establishment, as well as the construction and processing of TLS records, and the procedures for closing the secure channel. However, these software libraries may contain vulnerabilities or errors that could potentially jeopardize the security of the TLS channel. To identify flaws or deviations from established standards within the implemented code, a specialized tool known as **TLS-Anvil** can be utilized. This tool also verifies the compliance of TLS libraries with the specifications outlined in the Request for Comments documents published by the IETF. TLS-Anvil conducts numerous tests with a client/server configuration utilizing a specified TLS library and subsequently generates a report that details the number of successful tests. In this work, we exploit the results obtained from a selected subset of TLS-Anvil tests to generate rules used for anomaly detection in **Suricata**, a well-known signature-based Intrusion Detection System. During the tests, TLS-Anvil generates .pcap capture files that report all the messages exchanged. Such files can be subsequently analyzed with **Wireshark**, allowing for a detailed examination of the messages exchanged during the tests and a thorough understanding of their structure on a byte-by-byte basis. Utilizing the analyzed TLS handshake messages, we write tailored Suricata rules designed to identify TLS anomalies arising from erroneous implementations within the intercepted traffic. We detail the specific testbed put in place for deriving and validating some derived Suricata rules for the **OpenSSL** library. The rules that identify TLS deviations or potential attacks can subsequently be incorporated into a Suricata-enabled threat detection platform. This integration will facilitate the detection of TLS anomalies generated by code that does not conform to the specifications.

Keywords: TLS vulnerabilities; network monitoring; Suricata rules; threat detection; TLS-Anvil

---

In today's digital landscape, securing communications over networks is more crucial than ever. The TLS protocol provides encryption, authentication, and integrity for the application data exchanged between two end nodes. Thus, it is widely used in different contexts to support secure channels, ranging from networked applications to IoT or embedded systems. However, the complex nature of TLS architecture, as well as implementation issues encountered in various (versions of) libraries supporting this protocol opened the door to vulnerabilities, posing risks to TLS-enabled secure communications.

In the initial stages, various design factors contributed to the emergence of TLS vulnerabilities, specifically the *complexity of the protocol* and *backward compatibility*. The layered structure of TLS, which includes the handshake, key exchange, cipher negotiation, and compression created multiple attack surfaces. For instance, the rollback attack (first identified in the mid-1990s) against SSL 2.0 protocol

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

(ancestor of TLS) is one of the earliest examples of how backward compatibility in cryptographic protocols can be exploited. A rollback, or downgrade, attack deceives one party into believing that the other party only supports less secure options. In this scenario, the attacker intercepts and alters the handshake messages, forcing the connection to revert to SSL 2.0 or weaker ciphers, despite both parties have the capability to utilize stronger versions. The rollback attack affecting SSL 2.0 enabled subsequent exploits such as the DROWN (Decrypting RSA with Obsolete and Weakened eNcryption) attack [1], which was disclosed in March 2016. In this attack, servers supporting SSL 2.0 could be used to decrypt TLS sessions. The DROWN attack showed that even obsolete protocols like SSL 2.0 could compromise modern TLS if backward compatibility was left enabled.

As ciphers become outdated, weak algorithms must be avoided in TLS, such as MD5, SHA1, or the block algorithms in CBC mode. To mitigate design issues affecting the protocol specification, the algorithms employed within TLS, or some attacks against CBC mode, new versions of the protocol have been proposed, namely TLS 1.2 and TLS 1.3 [13]. Nonetheless, many systems are required to continue supporting old versions (SSL 3.0, TLS 1.0, TLS 1.1) and outdated ciphers for legacy clients. Consequently, standard TLS libraries have retained compatibility with multiple TLS versions and legacy algorithms, such as 3DES and SHA1. Historically, the complexity of TLS and its emphasis on backward compatibility have led to other significant security vulnerabilities, including POODLE [15], BEAST [7], and CRIME [8], which are summarized in Section 1.1.

Another significant cause of TLS attacks lies in the *implementation errors*. For example, minor coding mistakes, timing leaks and padding checks, have created vulnerabilities that have resulted in various high-impact attacks, such as Heartbleed [2], which are frequently more dangerous than the protocol flaws themselves. As an example, soon after the Heartbleed vulnerability has been publicly disclosed, there have been "exploit attempts from almost 700 sources, beginning less than 24 hours after disclosure" [3]. This situation arises from the fact that libraries, such as OpenSSL, have accumulated years of complexity, rendering subtle bugs unavoidable. Additionally, detailed error reporting has frequently provided attackers with "oracles" to exploit cryptographic vulnerabilities.

Motivation. To counter TLS attacks, several mitigations can be adopted. For instance, old versions of the protocol (i.e., SSL 3.0, TLS 1.0, and TLS 1.1) have been deprecated and (only) TLS 1.3 is recommended nowadays [10], while TLS 1.2 can still be used. Only strong cipher suites must be used, while weak cryptographic algorithms and modes, like RC4 [9], DES, SHA1, and the CBC mode, are disabled. TLS 1.3 does not support anymore RSA for key exchange but only ephemeral Diffie-Hellman, ensuring perfect forward secrecy. Additionally, TLS 1.3 includes mechanisms like "downgrade sentinels" in the ServerHello to prevent forced rollbacks. To support secure renegotiation, standardized fixes prevent injection attacks.

Typically, shortcomings in the implementations are solved in subsequent versions of the released TLS libraries. However, if an adversary successfully substitutes a non-vulnerable version of a TLS library at a designated node with a vulnerable one, the result is a compromised (TLS-aware) node. This node will receive and transmit data over a TLS channel, though it will be operating over a low-security channel. Several signature-based IDS or network monitoring tools, including Suricata, Zeek, or Qualys are already capable of analyzing TLS connections, checking (negotiated) protocol versions, certificate fields (issuer, subject, expiration date, a.s.o), or cipher suites. Qualys SSL Server Test simulates client connections, inspects the handshake, and evaluates the strength of the tested server to known TLS attacks. Other tools exploit Suricata or Zeek to look for more specific TLS attack patterns in the network traffic. For instance, TLS-Monitor [31] and Threat-TLS [30] searches for the Heartbeat extension in the network traffic as a potential indicator of the Heartbleed attack, or for the presence of RSA or a block algorithm in CBC mode in the negotiated ciphersuites (in the TLS handshake) that could allow POODLE, ROBOT, or Bleichenbacher-related. To the best of our knowledge none of these tools however analyze the network traffic looking for anomalies generated by flawed TLS software (i.e. non-compliant to standards or CA/Browser forum recommendations) to generate alerts.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

Contribution. To address the implementation flaws, researchers have proposed tools that can identify errors in TLS libraries or deviations from the standards. One such tool is TLS-Anvil [4], which checks the compliance of TLS 1.2 and 1.3 implementations with industry standards by executing a significant number of security tests. The authors of TLS-Anvil have developed a library of docker images to allow researchers to quickly start TLS clients and servers in different versions. The docker library contains "around 700 versions of 23 different implementations and provides a Java interface to start and stop TLS implementations easily." This docker library has been used to assess 13 widely used TLS libraries, namely BearSSL (v. 0.6), BoringSSL (v. 3945), Botan (v. 2.17.3), GnuTLS (v. 3.7.0), LibreSSL (v. 3.2.3), MatrixSSL (v. 4.3.0), mbed TLS (v. 2.25.0), NSS (v 3.60), OpenSSL (v. 1.1.1i), Rustls (v 0.19.0), s2n (v. 0.10.24), tlslite-ng (v. 0.8.0-a39), and wolfSSL (v 4.5.0).

In this work, we explore how such TLS-Anvil tests can be leveraged to enhance security measures through anomaly detection, specifically within the framework of Suricata [5]. We aimed to identify anomalies in TLS communications that arise from non-compliant TLS implementations, based on intercepted network traffic. A tailored IDS, equipped with specific rules for detecting anomalous TLS connections, is particularly advantageous in situations where it is impractical to assess the software integrity of the node, such as in cases involving remote attestation techniques [32], due to the limited resources available on the node itself. Ultimately, the capability to identify anomalous TLS connections within networking nodes, including routers and switches, is essential for evaluating whether the machines within a local network have been compromised through the installation of non-compliant TLS libraries.

Organization. The paper is organized as follows. Section 1 provides details of key attacks due to TLS complexity, backward compatibility, and implementation errors. Section 2 discusses some of the tools that have been created to detect and monitor TLS anomalies, attacks, or weak and suspicious connections. Section 3 details our approach: based on the tests performed with TLS-Anvil, we derive Suricata rules aimed at detecting inconsistencies or TLS errors in the TLS traffic. The final goal is to identify compromised nodes that run vulnerable TLS software. We explain the testbed and commands used for deriving and validating the derived Suricata rules for detecting some TLS anomalies for OpenSSL library. Finally, Section 4 concludes the paper and indicates future work.

1. Related Work

1.1. *Key Attacks Stemming from TLS Complexity & Backward Compatibility*

TLS downgrade attack, also referred to as version rollback or bidding-down attacks, occur when attackers deceive clients and servers into utilizing outdated and less secure versions of TLS or SSL. For instance, forcing a connection to revert from TLS 1.2 to SSL 3.0 exposes the system to established vulnerabilities such as POODLE (Padding Oracle On Downgraded Legacy Encryption), which takes advantage of the flawed padding mechanisms in SSL 3.0's block ciphers. Even when a system is equipped to support more current TLS versions, its backward compatibility with SSL 3.0 permits attackers to enforce a downgrade and potentially harvest session cookies. BEAST (Browser Exploit Against SSL/TLS) attack specifically targeted the predictable initialization vectors employed in CBC mode by TLS 1.0, enabling attackers to decrypt secure HTTPS traffic. CRIME (Compression Ratio Info-leak Made Easy) is an attack wherein an adversary can take advantage of TLS compression to extract confidential information, such as authentication cookies, by scrutinizing compressed traffic. The Lucky 13 Attack is a timing attack targeting CBC-mode ciphers in TLS versions 1.1 and 1.2, which exploits slight discrepancies in MAC and padding validation. Another recognized form of attack is the RC4 Bias attack. The RC4 encryption algorithm was maintained to ensure backward compatibility; nevertheless, the statistical biases inherent in its keystream allowed attackers to extract plaintext from encrypted sessions. A different category of attacks arised from vulnerabilities related to TLS renegotiation. Previous iterations of the TLS protocol allowed for insecure renegotiation, thereby enabling man-in-the-middle attacks. This facilitated the injection of malicious commands into sessions that were presumed to be secure.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

1.2. Key Attacks from TLS Implementation Errors

TLS implementation errors have led to several high-impact attacks, often more dangerous than the flaws in the protocol specification themselves. These arise when developers misapply cryptographic checks, do not check the size of data received [6], mishandle memory, or introduce timing leaks, creating exploitable vulnerabilities even in otherwise secure versions of TLS. A list of known TLS attacks, together with an explanation of their origin and severity is provided in Table 1, while a short discussion on the software implementations that have been subject to these attacks is given below.

One of the most notorious TLS vulnerabilities is *Heartbleed*, which arose from a flaw (CVE-2014-0160) present in specific versions of the OpenSSL library. The implementation(s) that were susceptible did not adequately manage the heartbeat extension during the TLS handshake, thereby enabling remote attackers to extract sensitive information from process memory through specially crafted packets.

In particular, the flaw was found in OpenSSL versions 1.0.1 through 1.0.1f and 1.0.2-beta1. Heartbleed represented one of the most extensive Internet security vulnerabilities recorded, impacting millions of servers, devices, and applications globally. At its peak in April 2014, estimates indicated that over 20% of the world's secure web servers were compromised, including significant platforms such as Yahoo, GitHub, and Cloudflare [2].

The *Lucky 13 attack* instead (CVE-2013-0169) [18] was a timing side-channel issue in how padding and MAC checks were implemented. It affected multiple TLS libraries, including OpenSSL, GnuTLS, PolarSSL (later renamed mbedTLS), OpenJDK (Java Secure Socket Extension – JSSE), Network Security Services (NSS) [25], All the affected libraries were implementing CBC mode in TLS 1.1 and 1.2.

Bleichenbacher's attack is another famous one, which affected nearly all major TLS libraries implementing RSA key exchange with PKCS#1 v1.5 padding. The vulnerability arises when implementations leak information (via error messages or timing) about invalid padding, creating a "padding oracle" that attackers can exploit to decrypt TLS traffic. OpenSSL was vulnerable to Bleichenbacher's original attack and later variants, and multiple CVEs (including those tied to the ROBOT attack in 2017) required patches to fix RSA padding oracles. The GnuTLS library was also susceptible to Bleichenbacher-style padding oracle attacks, since researchers demonstrated practical exploits against certain versions. NSS (Network Security Services – used by Mozilla) has been also found vulnerable to Bleichenbacher oracles. Specifically, all NSS versions prior to 3.41 were affected by a cached side-channel vulnerability [11] so, it required fixes to ensure constant-time RSA decryption. The JSSE (Java Secure Socket Extension, part of OpenJDK) was also impacted by this attack [26] and it was updated to mitigate oracle leaks. The mbedTLS / PolarSSL library was vulnerable to Bleichenbacher attacks in earlier versions, and it was later patched to enforce stricter RSA decryption checks. Finally, Cisco confirmed that some on its products, namely Cisco Firepower Threat Defense [19], were affected by Bleichenbacher variants (ROBOT attack, CVE-2017-12373, CVE-2017-15533, CVE-2017-17428).

The ROBOT vulnerability significantly impacted numerous widely utilized TLS implementations and software that continued to support RSA PKCS#1 v1.5 key exchange. Systems exposed to this vulnerability permitted attackers to execute RSA decryption or signing operations utilizing the server's private key, thereby compromising the confidentiality of TLS. The flaw was present in various implementations, including OpenSSL, Cisco TLS stacks, F5 BIG-IP, Fortinet FortiGate, JSSE, NSS, and numerous web applications such as Facebook and PayPal, thereby demonstrating that Bleichenbacher-style vulnerabilities have persisted throughout decades of TLS implementations [20].

The FREAK attack (Factoring RSA Export Keys, CVE-2015-0204) impacted both TLS clients and servers that continued to support obsolete 'export-grade' RSA cipher suites. This vulnerability was notably prevalent among major operating systems, web browsers, and libraries. It affected OpenSSL, Apple SecureTransport, Microsoft SChannel, the TLS stack in Android, as well as prominent browsers such as Safari, Internet Explorer, Chrome, and Android Browser, in addition to any server configured with RSA_EXPORT cipher suites [21].

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025

doi:10.20944/preprints202512.0139.v1

Table 1. TLS Attacks due to implementation errors.

TLS Attack (Year)	Origin	Impact	Severity/ Cause/ Persistence
Heartbleed (2014)	A buffer over-read has been identified in OpenSSL's implementation of the TLS heartbeat extension.	Allows attackers to read up to 64 KB of memory from servers, potentially exposing sensitive information such as private keys, passwords, and session data.	Approximately 20% of widely utilized HTTPS servers were affected at that time.
Lucky 13 (2013)	Timing side-channel in CBC padding and MAC verification.	Allowed attackers to gradually recover plaintext from TLS 1.1/1.2 connections.	Subtle differences in how implementations handled padding errors.
Bleichenbacher's Oracle Attacks (1998)	Incorrect handling of RSA PKCS#1 v1.5 padding in TLS implementations.	Enabled decryption of TLS sessions by exploiting error messages or timing differences.	Resurfaced repeatedly. Variants reappeared in different TLS libraries over decades.
ROBOT (Return Of Bleichenbacher's Oracle Threat) (2017)	Modern TLS stacks still vulnerable to Bleichenbacher-style RSA padding attacks.	Allowed decryption and impersonation attacks against servers using RSA key exchange.	The attack demonstrated that Bleichenbacher-style padding oracle vulnerabilities persisted for nearly 20 years after the original discovery in 1998.
FREAK (Factoring RSA Export Keys) (2015)	Implementation allowed forced downgrade to weak "export-grade" RSA keys.	Attackers could break 512-bit RSA keys and decrypt traffic.	Mismanagement of backward compatibility in libraries.
DROWN (2016)	Cross-protocol attack exploiting SSLv2 support in TLS implementations.	Attackers could decrypt TLS sessions if servers shared keys with SSLv2.	Poor isolation between protocol versions

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

Table 2. Categories of TLS Anomaly/Vulnerability Detection Tools.

Category	Example Tools	Focus
TLS Protocol Testing	TLS-Attacker, testssl.sh	Simulate attacks, detect protocol flaws.
Certificate Analysis	TLS hunting (Databricks), x509 monitoring (Censys)	Spot malicious infrastructure anomalies
TLS Server Scanners	Qualys SSL Labs, Pentest-Tool SSL/TLS Scanner, SSLyze, Nmap	Check configs, cipher suites, cert validity. Provide indication about TLS weaknesses when the scanners are run (when needed or at specific times/time intervals). May suggest possible remediation to the vulnerabilities found (patches to apply) .
Continuous Monitoring	Threat-TLS, TLS-Monitor	Detect evolving TLS threats in real time by exploiting network monitoring tools, like Suricata or Zeek.
TLS Compliance Tools	TLS-Anvil, FLEXTLS [28]	Test compliance of a TLS server or client implementation with the protocol specification. May be used also by penetration testers to estimate the quality of a TLS stack.

2. Tools for Detecting TLS Attacks, Anomalies or Suspect Connections

With the rise of TLS vulnerabilities, ensuring the integrity and safety of TLS connections has thus become a primary concern. On one hand, an intensive work has been done to update the specification, resulting in new versions, and the latest one (TLS 1.3) has been fully redesigned with respect to the previous versions. On the other hand, there have been created specialized tools, scanners, and tool for detecting TLS anomalies, attacks, or suspicious connections. They range from research frameworks for protocol testing to enterprise-grade monitoring solutions that analyze certificates, cipher suites, and traffic patterns. We provide further below a short description of some of these tools.

Qualys SSL Labs Server Test,Qualys is an online scanner widely used to assess TLS/SSL configurations of servers identified through their DNS names. It identifies weak cipher suites, protocol downgrades, expired certificates, and misconfigurations. Moreover, it generates a graded report to facilitate remediation.

TLS-Attacker,TLS-Attacker is a Java-based framework designed for the analysis of TLS libraries. It can simulate man-in-the-middle attacks, downgrade attempts, and protocol vulnerabilities. This framework is frequently utilized by researchers and penetration testers to assess the integrity of TLS implementations.

Threat-TLS,Threat-TLS and TLS-Monitor [31] are research tools developed to identify vulnerable, malicious, or suspicious TLS connections. These tools aim to detect anomalies that can generate attacks, cryptographic algorithms, and configuration errors. They are useful in various contexts both in networked environments exploiting a network-based Intrusion Detection System as well as in other contexts, like IoT, mobile, and embedded environments where TLS anomalies could be individuated in the network traffic.

Certificate-based Threat Hunting,Databricks utilizes TLS certificates obtained from the handshake process as a data source for anomaly detection. This approach aids in the identification of malicious infrastructures, botnets, or malware that utilize TLS to disguise their traffic. For instance, it can be employed to detect ransomware such as LockBit or Remote Access Trojans (RATs) that depend on TLS channels.

TLS Scanner tools such as Qualys SSL Labs, Pentest-Tools SSL/TLS Scanner, testssl.sh, SSLyze [16], and Nmap, are frequently used for scanning servers applications, and networks looking for SSL/TLS

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

vulnerabilities, misconfigurations, and weak cryptographic settings. They can provide information related to the scanned host(s): a) identify supported TLS/SSL versions and cipher suites; b) identify where the target is vulnerable to TLS attacks including Heartbleed, ROBOT, weak ciphers, and risks associated with protocol fallback; c) retrieve the certificate (and the related chain) of the target, validate the chain, checks the expiration, and issuer trust; d) Detect weak keys, insecure renegotiation, or fallback mechanisms.

TLS-Anvil is designed to assess the compliance of server or client implementations with the specifications of the protocol. It serves as a valid tool for penetration testers to evaluate the robustness of a TLS stack, as well as for developers to test their applications that utilize TLS.

3. Exploiting TLS-Anvil for Anomaly Detection with Suricata

3.1. TLS-Anvil and Suricata: Presentation

3.1.1. TLS-Anvil

TLS-Anvil, implemented in Java, is a test suite based on combinatorial testing [23,24] that aim to detect violations of the protocol specifications by TLS servers or clients by systematically testing parameter value combinations. TLS-Anvil exploits software tests to stimulate a System Under Test (SUT) with inputs and observe the reaction of the system. To create test templates, the authors carefully analyzed TLS-related RFCs for absolute requirements which are marked with the MUST, SHALL, or REQUIRED keywords. Additionally, the tool integrates known state machine vulnerabilities from related works [14]. From the practical point of view, TLS-Anvil uses the following libraries: JUnit 5 (test framework), TLS-Attacker (TLS stack), TLS-Scanner (scanner based on TLS-Attacker), coffee4j (combinatorial test library).

Figure 1 shows the general architecture of TLS-Anvil, including the different phases (detailed in [4]) that are performed during the execution of a test. Each test template is a JUnit test function that verifies a given requirement and defines the desired outcome for all test cases derived from it. It basically has two functions:

1. Defines which TLS messages are sent and received by the test suite.
2. Defines when a test case succeeds or fails.

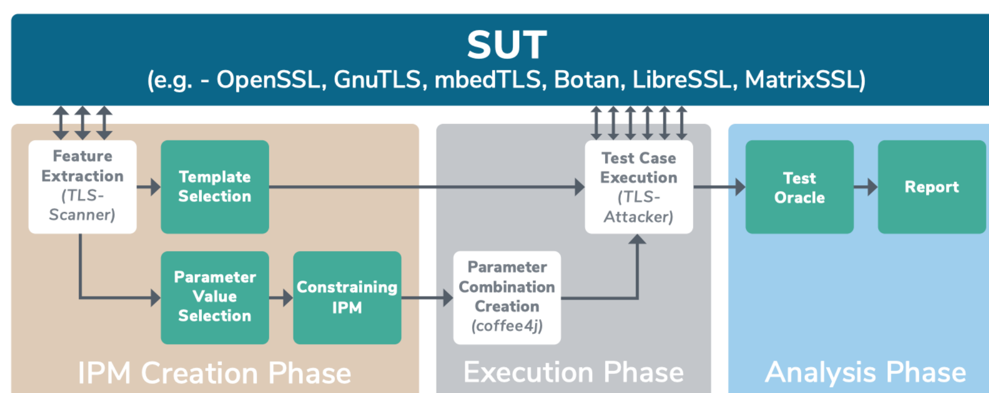

Figure 1. TLS-Anvil Architecture (source: [4]).

The template has additional Java annotations that define an IPM (*Input Parameter Model*), which contains all relevant test parameters and their values. The IPM is used to generate test inputs (a value is assigned to each parameter) using t-way combinatorial tests. Different IPMs are defined for each test template, depending on the requirement it verifies.

TLS-Anvil is structured into two Java modules:

- **TLS-Testsuite**: This is the main module, containing all the templates found within the `de.rub.nds.tlstest.suite.tests` package; this package contains three test types: `server`, `client`, and `both` (tests that work for both peers). The tests are then further broken down by RFC.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

- **TLS-Test-Framework**: This module contains all the JUnit extensions and test execution logic.

3.1.2. Suricata

Suricata is a high-performance intrusion detection and prevention system, which is widely used also as network security monitoring engine. It is open source and owned by a community-driven nonprofit, the Open Information Security Foundation (OISF). Like other IDSs, Suricata provides threat detection capabilities. It also allows traffic filtering and monitoring. Suricata can detect common types of attacks such as port scanning and denial-of-service attacks. This software uses a set of rules to perform threat detection and analysis and offers network administrators the ability to write and apply their own detection rules. Additionally, Suricata is multi-threaded, meaning it runs multiple threads at the same time, if the system allows it.

Additionally, by setting the `max-pending-packets` parameter in the `suricata.yaml` file, it is possible to decide the number of packets Suricata can process simultaneously; this value can range from one to tens of thousands of packets and prevents packets from being lost due to saturation.

Since in this work we will explain how we derived some rules, we explain first briefly the semantics of such rules with the help of an example Suricata rule (rule/signature).

A Suricata rule is made up of three parts: action, header, and options.

Action. This is the first word of the rule, shown in red in the Figure 2. The possible actions that can be indicated are:

- `alert` - generates an alert.
- `pass` - blocks packet inspection.
- `drop` - drops the packet and generates an alert.
- `reject/rejectsrc` - sends an RST/ICMP unreachable error to the packet's sender.
- `rejectdst` - like reject, but to the recipient.
- `rejectboth` - like reject, but to both peers.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (msg:"HTTP GET Request
Containing Rule in URI"; flow:established,to_server; http.method;
content:"GET"; http.uri; content:"rule"; fast_pattern; classtype:bad-
unknown; sid:123; rev:1;)
```

Figure 2. Example Suricata Rule.

Header. This part is shown in green in the Figure 2 and consists of:

1. Protocol - the protocol the rule refers to; the main ones are TCP, UDP, ICMP, and IP.
2. Source/Destination - IP address (or range of addresses) of the packet's sender/recipient.
3. Source/Destination Port - port (or range of ports) of the sender/recipient.
4. Direction - direction of the traffic to be analyzed, usually `->`.

The address and port fields can be replaced with `any` if you don't want a specific restriction.

Rule Options. This part is shown in blue in the Figure 2 and represents the rule's specifications. It contains keywords from various categories, such as Meta and Payload.

Meta keywords do not have a direct effect on network traffic inspection, but they do affect how events/alerts are reported. The main ones are:

- `msg` (message) - provides information about the rule and the potential alert, such as the RFC it references or the potential attack in progress.
- `sid` (signature ID) - gives each rule its ID (a number greater than 0).
- `rev` (revision) - often appears after `sid`; represents the version of the rule, i.e., a number that is incremented each time it is modified.

The **Payload** keywords are used to inspect the contents of a packet's payload. The keywords used in deriving the Suricata rules in our work are:

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

- content - indicates in quotation marks what you want to be present in the packet; if the content is ASCII, it is simply written, while hexadecimal is delimited by two slashes, "|", for example, content:"|16 03 03|";.
- offset - indicates the byte of the payload from which a match will be sought; for example, offset:3; checks from the fourth byte onwards.
- depth - indicates how many bytes from the beginning of the payload will be checked. offset and depth can be combined and are often used together; for example, offset:3; depth:3; checks the fourth, fifth, and sixth bytes of the payload.

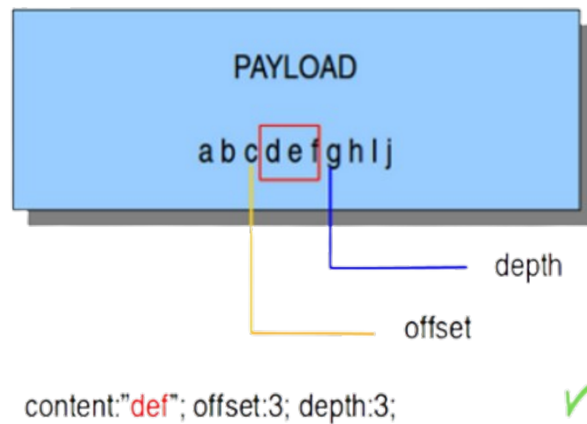

Figure 3. Example for specifying an "offset" and "depth" in a Suricata rule.

- distance - unlike the previous modifiers, it is a **relative** modifier, meaning it refers to the previous content. The value given to distance determines how many bytes away from the previous match to start searching for the new match.
- within - specifies **within** how many bytes from the previous content to search for a match. distance and within are often used together to search within a specific range of bytes; in this example, distance:0; within:3; limits the search to the 3 bytes following the content "abc".

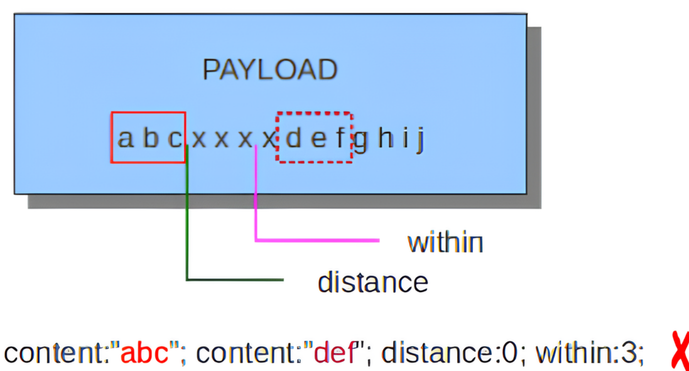

Figure 4. Example for specifying a "distance" and "within" in a Suricata rule.

- byte_test - is a keyword with several input values:
  1. <num of bytes> - the number of bytes to be examined.
  2. <operator> - is the operator used to compare the bytes with the test value; it can be <, >, =, <=, >=, & (if preceded by an exclamation point, they are negated).
  3. <test value> - value to compare the bytes against, using the operator; can be decimal or hexadecimal.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

4. `<offset>` - indicates the position in the payload of the bytes to be selected. By adding `relative`, the offset will not be absolute, but will be considered the distance from the previous content.

Finally, the keyword `flow` indicates the flow direction, that is, one between `from_server` and `from_client`.

3.2. Deriving Suricata Rules from TLS-Anvil Tests

In this section we detail some (new) Suricata rules we have derived based on selected TLS-Anvil tests and the traffic they generated, which was then analyzed using Wireshark. We indicate which TLS-Anvil test and relative file(s) we have used to generate each TLS anomaly, and afterwards we show the derived Suricata rule.

All the files related to the TLS-Anvil tests are located in the GitHub repository [33] in the tests folder. Therefore, all the file paths mentioned in this section originate from that folder.

3.2.1. TLS Anomaly: Empty TLS Records

Sending *Application Data* records with length 0 is not prohibited by the TLS protocol specification, which specifies that such empty records are potentially useful as a traffic analysis countermeasure. An adversary could, in fact, attempt to deduce the content of the communication by analyzing the packet length. On the other hand, receiving an excessive number of empty records, unaccompanied by real data, could indicate a **Denial-of-Service (DoS)** attack attempt. It is therefore advisable to generate an alert to report this suspicious message. For this reason, we derive a specific Suricata rule to individuate *Application Data* records of length length 0.

TLS-Anvil test description and files. The test sends an empty *Application Data* record, i.e., with length 0. The test is located in TLS-Anvil in the file named `Fragmentation.java` available in TLS-Anvil tests folder [34]. Figure 5 shows a Wireshark screenshot of a dump file generated by the test, where the *Application Data* record with length 0 is highlighted. The derived Suricata rule for TLS Record of type *Application Data* with length 0 is shown below:

```
alert tls any any -> any any (msg:"RFC 5246: Empty Application Data";
flow:from_client;
content:"|17 03 03 00 00|"; depth:5;
sid:1000004;)
```

Rule description: The rule searches for an empty *Application Data* record, i.e., with length 0, where the string "|17 03 03 00 00|" indicates a record of type *Application Data* (0x17) with TLS version 1.2 (0x0303) and length 0 (0x0000); this sequence must be at the beginning of the record (`depth:5;`).

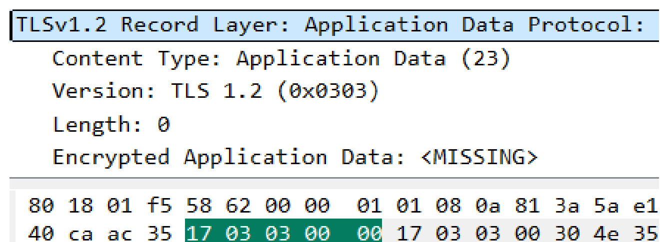

Figure 5. Wireshark dump showing the traffic generated in the test for TLS Record of type *Application Data* with length 0.

Sending TLS record fragments of length 0 for the Handshake messages is instead forbidden (by the RFC), so in this case the server should respond with a **fatal alert**.

TLS-Anvil test description and files: In this test, the client (i.e. TLS-Anvil) sends an empty *Finished* record, i.e., with length 0. Figure 6 shows a Wireshark screenshot of a dump file generated

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

by the test, where the *Finished* record with length 0 is highlighted. The test is located in the file `Fragmentation.java`.

The derived Suricata rule for an empty Finished TLS handshake record is shown below:

```
alert tls any any -> any any (msg:"RFC 5246: Empty Finished Record";
flow:from_client;
content:"|16 03 03 00 00|"; depth:5;
sid:1000005;)
```

Rule description: The rule searches for a message of type Handshake with length 0, where |16 03 03 00 00|: indicates a *Handshak*e message (0x16) with TLS version 1.2 (0x0303) and length equal to 0 (0x0000); this sequence must be found at the beginning of the TLS record (`depth:5;`).

Figure 6 illustrates the screenshot captured with Wireshark for a dump file generated in this test, where it can be noticed the *Handshake* message of length 0; Wireshark also flags the record as "not allowed" due to its length of 0.

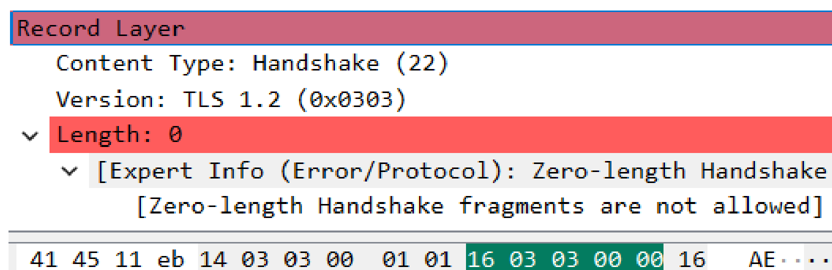

Figure 6. Wireshark dump showing the traffic generated in the test for TLS Record of type *Handshake* with length 0.

3.2.2. TLS Anomaly: Undefined TLS Record Type

TLS Record Content type: According to the TLS 1.2 specification [12], any TLS record must contain information about the type of the higher-level protocol carried inside the TLS record. The type, corresponding to application data, TLS handshake message, change cipher spec message, or alert messages, can be one of the following:

```
enum {
change_cipher_spec(20), alert(21), handshake(22),
application_data(23), (255)
} ContentType;
```

In hexadecimal these values correspond to `0x14, 0x15, 0x16` and `0x17`.

We consider the following anomaly: the presence of an undefined TLS Record type within the **ClientHello** handshake message. We run the following tests with TLS-Anvil.

TLS-Anvil test description and files: TLS-Anvil sends a **Client Hello** with an incorrect `Content Type`, specifically set to `0xFF` instead of `0x16` (as shown in Figure 7). The server (implementing the TLS 1.2 specification) should detect invalid content types and should respond with a **fatal alert** of type `unexpected_message`. The test (called *sendNotDefinedRecordTypesWithClientHello*) is found in the file `server/tls12/rfc5246/TLSRecordProtocol.java` located at [34]. Another test (named *sendNotDefinedRecordTypesWithCCSAndFinished* test), found in the same file, performs a similar operation, but with **ChangeCipherSpec** and **Finished** handshake messages. In Figure 7, we show a Wireshark screenshot of a dump file generated by these test(s), where the first 3 octets of the record are marked, representing the *Content Type* and the version.

The derived Suricata rule for an undefined TLS Record type is shown below:

```
alert tls any any -> any any (msg:"RFC 5246: Not Defined ContentType";
flow:from_client;
```

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

```
byte_test:1,!=,0x14,0;
byte_test:1,!=,0x15,0;
byte_test:1,!=,0x16,0;
byte_test:1,!=,0x17,0;
content:"|03 03|"; offset:1; depth:2;
sid:1000009;)
```

Rule description: This rule searches for a TLS record with an undefined content type, which refers to any content type that deviates from the four permissible values specified in RFC 5246 [12]. More in detail:

- The `byte_test` functions check the first byte of the record (`offset 0`), verifying whether it is different from the four *Content Type* values defined in the RFC.
- |03 03|: This sequence corresponds to the *version* field of the record and indicates TLS 1.2 (0x0303); the two bytes are the second and third of the record, i.e., 2 bytes deep with an offset of 1 (`offset:1; depth:2;`).

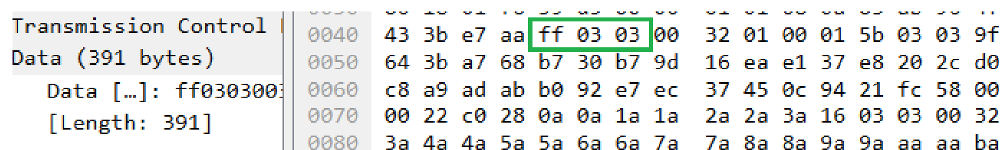

Figure 7. Wireshark dump showing the traffic generated in the tests for undefined TLS Record type.

3.2.3. TLS Anomaly: TLS Record with Excessive Size (Too Large CipherText)

According to the TLS specification, each TLS record fragment containing encrypted Application data should be at most $2^{14} + 2048$ bytes.

We consider the following anomaly: TLS Record fragment (with encrypted application data) whose size is bigger that $2^{14} + 2048$ bytes. We run the following test with TLS-Anvil.

TLS-Anvil test description and files: TLS-Anvil sends a fragment with encrypted application data (`TLS_Ciphertext`) whose size is bigger than $2^{14} + 2048$ bytes allowed by the RFC. Il server should respond with a fatal alert of type `record_overflow`. Figure 8 shows a Wireshark screenshot of a dump file generated by the test, highlighting the *length* field as being greater than `0x4800`; Wireshark also reports the prohibited length of the record. The test is found in the file `Fragmentation.java`.

The derived Suricata rule for a TLS Record of type *Application Data* with excessive size (Too Large CipherText) is shown below:

```
alert tls any any -> any any (msg:"RFC 5246: Too Large CipherText
(>2^14+2048 bytes)";
flow:from_client;
content:"|17 03 03|"; depth:3;
byte_test:2,>,0x4800,0,relative;
sid:1000007;)
```

The above rule detects a TLS record of type *Application Data* with a *length* field greater than $2^{14} + 2048$. The meaning of the fields in the Suricata rule is explained next:

- |17 03 03|: indicates a record of type *Application Data* (0x17) for the version TLS 1.2 (0x0303); this sequence is found at the beginning of the TLS record (`depth:3;`).
- the function `byte_test` verifies if the number represented by the immediately following 2 bytes (`0,relative`) is greater than $2^{14} + 2048$, or 4800 in hexadecimal (`>,0x4800`).

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1


```
TLSv1.2 Record Layer: Application Data Protocol: Application Data
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
  ⌄ Length: 18433
      > [Expert Info (Error/Protocol): TLSCiphertext length MUST NOT exceed 2^14 + 2048]
 17 03 03 48 01 00 00 00  00 00 00 00 00 00 00 00   ···H··· ········
```

Figure 8. Wireshark dump showing the traffic generated in the test for TLS Record with excessive size (Too Large CipherText).

3.3. Testing the Derived Suricata Rules

We have created a testbed both to run the tests with TLS-Anvil and thus to derive the new Suricata rules, as well to to validate the rules. We have set up a local TLS-enabled server, used as SUT. In the examples we have used the OpenSSL library.

We used the TLS-Docker-Library [35], which provides Docker images for various TLS libraries with different versions. A dedicated Docker network was established for the TLS-Anvil containers and the server. We created tags to enable running individual tests with TLS-Anvil. An OpenSSL server was launched with an RSA certificate, and TLS-Anvil (acting as a TLS client) was initiated for each specific test using custom tags like S7507incFallSCSV. The specific commands are indicated below, followed by a brief explanation:

```
1 $ docker run \
2 -d \
3 --rm \
4 --name openssl-server \
5 --network tls-anvil \
6 -v cert-data:/certs/ \
7 ghcr.io/tls-attacker/openssl-server:1.1.1i \
8 -port 8443 \
9 -cert /certs/rsa2048cert.pem \
10 -key /certs/rsa2048key.pem
```

Lines 2-6 are command flags related to Docker, line 7 specifies the Docker image from the library, and lines 8-10 contain the flags passed as parameters to the OpenSSL server.

Next, TLS-Anvil is started. The current directory is mounted in the Docker container and is used to store the results.

```
1 $ docker run \
2 --rm \
3 -it \
4 --name tls-anvil \
5 --network tls-anvil \
6 -v $(pwd):/output/ \
7 tlsanvil \
8 -identifier openssl-server \
9 -tags S7507incFallSCSV \
10 server \
11 -connect openssl-server:8443
```

- Lines 2-5 are Docker-related command flags.
- Line 6 sets the output directory (the current one).
- Line 7 specifies the TLS-Anvil image. The image from the offial TLS-Anvil site (`ghcr.io/tls-attacker/tlsanvil:l` cannot be used because it would be downloaded at the moment, so it would not contain user-added tags (necessary to perform an individual test) such as `S7507incFallSCSV`.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

- Line 9 defines the textbftag to use.
- Line 11 determines how TLS-Anvil connects to the server.

After deriving the Suricata rules, we validated them by first installing Suricata, then updating the rule file with the derived rules. We inserted the derived rules into a separate file named `derivedsuricatarulesTLSanomalies.rules`. We have updated the file `suricata.yaml` in the `rule-files` part as shown below:

```
rule-files:
- suricata.rules
- /path/to/derivedsuricatarulesTLSanomalies.rules
```

Afterwards, we started Suricata with the command:

```
$ sudo /usr/local/bin/suricata -c \
/usr/local/etc/suricata/suricata.yaml -i $INTERFACE$
```

Instead of `INTERFACE`, it can be indicated `any` (to analyze all traffic), a specific interface, such as `eth0`, or `docker0` to analyze traffic coming from a TLS-Anvil Docker container.

To read Suricata logs in real time and check its operation, in another terminal the following command is used:

```
$ tail -f /usr/local/var/log/suricata/fast.log
```

The path for the logs is found in `suricata.yaml`, under `default-log-dir`. Finally, the OpenSSL test server and the TLS-Anvil Docker test container are started, as explained above, to verify that the Suricata rules are triggered correctly.

Figure 9 shows a part of the file `fast.log`, which includes a default Suricata rule and one derived rule for a TLS anomaly, namely a ciphersuite containing TLS_FALLBACK_SCSV in the list of cipher suites contained in the Client Hello handshake message, corresponding to the TLS-Anvil test contained in the file `SCSV.java`. It can be noted that the log records the date and time the rule was triggered, the signature ID, the message, the classification, the priority level, and the sender and recipient of the message that caused the trigger.

```
06/16/2025-06:55:39.096462  [**] [1:2230010:1] SURICATA TLS invalid record/traffic [**] [Classifica
tion: Generic Protocol Command Decode] [Priority: 3] {TCP} 172.17.0.2:50016 -> 10.0.2.15:4433
06/16/2025-06:55:39.096462  [**] [1:1000001:0] RFC 7507: TLS 1.0 Client Hello with Cipher Suite TLS
_FALLBACK_SCSV [**] [Classification: (null)] [Priority: 3] {TCP} 172.17.0.2:50016 -> 10.0.2.15:4433
06/16/2025-06:55:39.483699  [**] [1:1000001:0] RFC 7507: TLS 1.0 Client Hello with Cipher Suite TLS
_FALLBACK_SCSV [**] [Classification: (null)] [Priority: 3] {TCP} 172.17.0.2:50022 -> 10.0.2.15:4433
```

Figure 9. Example of Suricata log indicating the test for Cipher Suite TLS with Fallback SCSV.

4. Conclusions

The objective of this work was to explore in depth the identification of vulnerabilities within implementations of the TLS protocol out from intercepted network traffic. This capability enables network administrators and security managers to receive notifications regarding flaws or anomalies that may lead to performance deterioration, compromise system integrity, or result in communication errors. Our work exploits TLS-Anvil, a famous tool that can be employed to conduct TLS compliance evaluations. In particular, this tool executes tests that enable to identify TLS implementations that deviate from the protocol standard specified in several different RFCs.

Using the information collected in a set of selected TLS-Anvil tests, we created rules for the Suricata engine, modeled to recognize patterns for software anomalies in the intercepted TLS messages. These rules were designed to send alarm messages in the presence of a connection anomaly related to an RFC standard. They are therefore intended as a support tool that can be integrated into an IDS or similar applications. The primary objective was achieved because the rules proved effective in detecting irregularities introduced by the testing tool, reporting alerts promptly.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

One potential advancement of this work includes the definition of rules for all 400 tests performed with TLS-Anvil for the various versions of TLS libraries. Once defined, such rules can be integrated into a more complex IDPS system to identify possible violations in nodes or systems executing the vulnerable TLS software. In this way, it is possible to subsequently conduct further investigations about the underlying causes and any potential malicious intent associated with the connection.

Additionally, strategies can be explored to optimize the Suricata code to further reduce the number of computational operations performed. This can be particularly advantageous for systems with low computational capacity, such as IoT devices. Finally, other compliance checking tools and new RFCs can be explored to create new rules, in addition to updating existing ones in the event of protocol changes.

Funding: This work was supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. CVE-2016-0800 Detail. Available online: https://nvd.nist.gov/vuln/detail/CVE-2016-0800
2. Heartbleed bug. Available online: https://owasp.org/www-community/vulnerabilities/Heartbleed_Bug
3. Durumeric, Z.; Li, F,; Kasten, J.; Amann, J.; Beekman, J.; Payer, M.; Weaver, N.; Adrian, D.; Paxson, V.; Bailey, M.; Halderman, J. A. The Matter of Heartbleed. In Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14). Association for Computing Machinery, New York, NY, USA, 475–488. https://doi.org/10.1145/2663716.2663755
4. Maehren, M.; Nieting, P.; Hebrok, S; Merget, R.; Somorovsky, Juraj; Schwenk, J. TLS-Anvil: Adapting Combinatorial Testing for TLS Libraries. In Proceedings of the 31th Usenix Security Symposium 2022, August 10–12, , Boston, MA, USA 2022, pp. 215-232. Available online: https://www.usenix.org/system/files/sec22-maehren.pdf
5. Suricata. Available online: https://suricata.io/
6. Berbecaru, D.; Lioy, A. (2007). On the Robustness of Applications Based on the SSL and TLS Security Protocols. In: Lopez, J., Samarati, P., Ferrer, J.L. (eds) Public Key Infrastructure. EuroPKI 2007. Lecture Notes in Computer Science, vol 4582. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-73408-6_18
7. Duong, T.; Rizzo, J. Here Come The ⊕ Ninja. May 13, 2011, Available online: https://nerdoholic.org/uploads/dergln/beast_part2/ssl_jun21.pdf
8. Rizzo, J.; Duong, T. Crime (Compression ratio info-leak made easy). Available at: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit
9. Vanhoef, M.; Piessens, F. 2015. All your biases belong to us: breaking RC4 in WPA-TKIP and TLS. In Proceedings of the 24th USENIX Conference on Security Symposium (SEC'15). USENIX Association, USA, 97–112. Available at: https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-vanhoef.pdf
10. Holz, R; Hiller, J.; Amann, J.; Razaghpanah, A; Jost, T.; Vallina-Rodriguez, N.; Hohlfeld, O. 2020. Tracking the deployment of TLS 1.3 on the web: a story of experimentation and centralization. SIGCOMM Comput. Commun. Rev. 50, 3 (July 2020), 3–15. https://doi.org/10.1145/3411740.3411742
11. CVE-2018-12404. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-12404
12. Dierks T.; Rescorla E. (2008). The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246. Available online: https://datatracker.ietf.org/doc/html/rfc5246
13. Rescorla, E. The Transport Layer Security protocol Version 1.3. August 2018, RFC 8446. Available online: https://www.rfc-editor.org/rfc/rfc8446
14. de Ruiter, J.; Poll, E. Protocol State Fuzzing of TLS Implementations. In 24th USENIX Security Symposium 2015, August 12-14, 2015, Washington D.C, USA, Available online: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter
15. Möller, B; Duong, T; Kotowicz, K. This POODLE Bites: Exploiting The SSL 3.0 Fallback. Google, September 2014. Available online: https://openssl-library.org/files/ssl-poodle.pdf
16. SSLyze - Fast and powerful SSL/TLS scanning library. Available online: https://github.com/nabla-c0d3/sslyze

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 2 December 2025 doi:10.20944/preprints202512.0139.v1

17. Nmap Security Scanner. Available online: https://nmap.org/
18. CVE-2013-0169: Understanding the "Lucky Thirteen" Attack on TLS and DTLS Protocols. Available online: https://www.cve.news/cve-2013-0169/
19. CVE-2022-20940 Detail. Available online: https://nvd.nist.gov/vuln/detail/CVE-2022-20940
20. Böck, H.; Somorovsky, J.; Young, C. Return Of Bleichenbacher's Oracle Threat (ROBOT). In Proceedings of the 27th Usenix Security Symposium. Available online: https://www.usenix.org/conference/usenixsecurity18/presentation/bock
21. What Is the FREAK Vulnerability? How to Prevent SSL FREAK Attacks. Available online: https://certpanel.com/resources/what-is-the-freak-vulnerability-how-to-prevent-ssl-freak-attacks/
22. Hunting Anomalous Connections and Infrastructure With TLS Certificates, January 20, 2022. Available online: https://www.databricks.com/blog/2022/01/20/hunting-anomalous-connections-and-infrastructure-with-tls-certificates.html
23. Simos, D.E.; Bozic, J.; Garn, B.; Leithner, M.; Duan, F; Kleine, K; Lei, Y.; Wotawa, F. Testing TLS using planning-based combinatorial methods and execution framework. Software Qual J 27, 703–729 (2019). Available online: https://doi.org/10.1007/s11219-018-9412-z
24. Simos, D.E.; Bozic, J.; Duan, F.; Garn, B.; Kleine, K; Lei, Y.; Wotawa, F. (2017). Testing TLS Using Combinatorial Methods and Execution Framework. In: Yevtushenko, N., Cavalli, A., Yenigün, H. (eds) Testing Software and Systems. ICTSS 2017. Lecture Notes in Computer Science(), vol 10533. Springer, Cham. https://doi.org/10.1007/978-3-319-67549-7_10
25. Lucky 13 Vulnerability. Available online: https://brandsek.com/kb/books/ssl-vulnerability/page/lucky-13-vulnerability
26. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks, In Proceedings of the 23rd USENIX Security Symposium, 2014 - San Diego, United States, 20 - 22 Aug 2014, pp. 733-748. Available online: https://research.utwente.nl/files/24317210/revisiting.pdf
27. TLS-Attacker. Available online: https://deepwiki.com/tls-attacker/TLS-Attacker
28. Benjamin Beurdouche, Antoine Delignat-Lavaud, Nadim Kobeissi, Alfredo Pironti, and Karthikeyan Bhargavan. FLEXTLS: A Tool for Testing TLS Implementations. In 9th USENIX Workshop on Offensive Technologies (WOOT), August 10-11, 2015, Washington, D.C., USA, Available at:
29. Qualys SSL Labs - SSL Server test. Available online: https://www.ssllabs.com/ssltest/
30. Berbecaru, D.G; Lioy, A. 2024. Threat-TLS: A Tool for Threat Identification in Weak, Malicious, or Suspicious TLS Connections. In Proceedings of the 19th International Conference on Availability, Reliability and Security (ARES '24). Association for Computing Machinery, New York, NY, USA, Article 125, 1–9. https://doi.org/10.1145/3664476.3670945
31. Berbecaru, D.G; Petraglia, G. TLS-Monitor: A Monitor for TLS Attacks. 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2023, pp. 1-6, doi: 10.1109/CCNC51644.2023.10059989.
32. Coker, G.; Guttman, J.; Loscocco, P.; Herzog, A.; Millen, J.; O'Hanlon, B.; Ramsdell, J.; Segall, A.; Sheehy, J.; and Sniffen, B. 2011. Principles of remote attestation. Int. J. Inf. Secur. 10, 2 (June 2011), 63–81. https://doi.org/10.1007/s10207-011-0124-7
33. TLS-Anvil GitHub Repository. Available online: https://github.com/tls-attacker/TLS-Anvil/
34. TLS-Anvil test files. Available online: https://github.com/tls-attacker/TLS-Anvil/tree/main/TLS-Testsuite/
35. TLS-Docker-Library GitHub repository. https://github.com/tls-attacker/TLS-Docker-Library

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2025 by the author(s). Distributed under a Creative Commons CC BY license.