Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

Automated Assistance for Actionable Security: Security and Compliance of TLS Configurations

by

Salvatore Manfredi

Theses Series **DIBRIS-TH-2023-XX**

DIBRIS, Università di Genova
Via Opera Pia, 13 16145 Genova, Italy https://www.dibris.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica, Bioingegneria,

Robotica ed Ingegneria dei Sistemi

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum

Automated Assistance for Actionable Security: Security and Compliance of TLS Configurations

by

Salvatore Manfredi

March, 2023

Dottorato di Ricerca in Informatica ed Ingegneria dei Sistemi
Indirizzo Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Univ. di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
https://www.dibris.unige.it/

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum
(S.S.D. INF/01)

Submitted by Salvatore Manfredi
DIBRIS, University of Genoa, Genoa (Italy)
Center for Cybersecurity, FBK, Trento (Italy)

Date of submission: December 2022

Title: Automated Assistance for Actionable Security: Security and Compliance of TLS Configurations

Advisor: Silvio Ranise
Director, Center for Cybersecurity, FBK, Trento (Italy)
Full professor, Department of Mathematics, University of Trento, Trento (Italy)

Co-Advisor: Giada Sciarretta
Researcher, Center for Cybersecurity, FBK, Trento (Italy)

Ext. Reviewers: Stefano Calzavara[⋆], Juraj Somorovsky[†]
[⋆] Associate Professor, Department of Environmental Science, Informatics and Statistics, Università Ca' Foscari Venezia, Venezia (Italy)
[†] Associate Professor, Faculty of Computer Science, Electrical Engineering and Mathematics, Paderborn University, Paderborn (Germany)

Abstract

Since its first version was published in 1999 as an RFC, Transport Layer Security (TLS) has rapidly become the de facto standard for providing confidentiality and integrity to communications exchanged in an insecure environment. Despite the fact that the protocol has undergone a number of structural and logical revisions over the past 23 years, its implementations still contain a significant flaw. In essence, the lack of a default secure configuration reduces the usability of TLS because system administrators, who are responsible for its configuration, must study the state of the art in terms of technologies and new vulnerabilities, and understand how to secure their deployment.

While TLS cipher suites and extensions have evolved over the past decade, increasing the protocol's overall security, supporting software has been slow to drop support for insecure ciphers, broken hash algorithms, and outdated protocol versions [KRA+18]. All components that system administrators and app developers may accidentally make available on their systems. In addition to managing the webserver itself, administrators must also manage cross-cutting elements such as certificate management, the TLS library, and the system on which these applications are installed. Administrators can take advantage of the availability of state-of-the-art tools, TLS analyzers capable of detecting a wide variety of vulnerabilities on a target system. Some are downloadable, while others can be run directly through the browser, but they all share one drawback: they cannot guide system administrators and app developers through the process of fixing the discovered vulnerabilities. This partially defeats the purpose of these tools, as administrators will still need to spend time researching scientific literature and determining how to fix the discovered misconfiguration.

To assist system administrators and app developers with the crucial task of configurating their deployments, we contribute with:

- *a study of the state-of-the-art in terms of TLS analyzers capable of inspecting webservers' configuration, performing vulnerability detection and compliance analysis, with a focus on their features, limitations and application domains;*

- *the design and development of TLSAssistant, a modular and extendable open source framework that can assist system administrators and app developers in correctly configuring TLS deployments. The strengths of the tool include its ability to generate actionable reports, containing accurate, concise information and code fragments that can be copied and pasted into the webserver configuration in order to mitigate the detected vulnerabilities;*
- *the development of a methodology that can assist users in assessing the compliance of their systems against a single or multiple agency-issued security guidelines;*
- *the application of TLSAssistant in a variety of corporate scenarios and projects, describing its impact and the lessons learned from its deployment.*

Acknowledgements

English

First of all, I want to thank Professor Might who was able to formalize what "I wanted to do when I grew up" [Mig] and Professor Cavallaro, who made me discover it [WSF]. I want to thank Professor Ranise, who gave me the opportunity to pursue this goal, and Dr. Sciarretta, whose only fault was that she had the most interesting internship topic among those proposed. I sincerely thank Professor Merlo and Dr. Fabiano for their great support when I most needed it.

I also want to express my gratitude to Dr. Sharif, the best colleague/helpdesk I could have asked for, as well as all the friends and coworkers who have helped me during these challenging years.

Finally, I want to thank my parents and my sister. If I have come this far, it is also your fault.

Italiano

Per prima cosa, voglio ringraziare il Professor Might che è riuscito a formalizzare ciò che "avrei voluto fare da grande" [Mig] e il Professor Cavallaro che me lo fece scoprire [WSF]. Ringrazio il Professor Ranise, che mi ha dato la possibilità di perseguire questo obbiettivo e la Dottoressa Sciarretta, la cui unica colpa è stata quella di aver proposto l'argomento di tirocinio più interessante. Ringrazio sentitamente il Professor Merlo e la dottoressa Fabiano per il loro importantissimo supporto quando ne avevo più bisogno.

Ringrazio il Dr. Sharif, il miglior collega/helpdesk che avrei potuto sperare di trovare, e tutte le persone, i colleghi e gli amici che mi hanno supportato in questi anni impegnativi.

Voglio infine ringraziare i miei genitori e mia sorella. Se sono arrivato fin qui è anche colpa vostra.

Table of Contents

List of Figures **4**

List of Tables **6**

Chapter 1 Introduction **8**

1.1 Goals and Contributions 9

1.2 Thesis Structure 10

Chapter 2 Background **13**

2.1 Historical Notes 13

2.2 Transport Layer Security 14

2.2.1 TLS Handshake 14

2.3 Misconfigurations and Vulnerabilities 19

Chapter 3 Actionable Hints for Configuring TLS **23**

3.1 Related Work: TLS Analyzers 23

3.1.1 Mobile Clients 27

3.1.2 Report Snippet 28

3.2 Actionable Mitigations 28

3.2.1 Mitigations Identification 29

3.2.2 Initial Tool Design and Assumptions 29

3.2.3 Attack Trees 33

3.3 Usability and Impact of Assisted Mitigations . . . 36
3.3.1 Related work: Usability Studies . . . 38
3.3.2 User Study Design . . . 39
3.3.3 User Study Results . . . 48
3.4 Lessons Learned and Discussion . . . 53

Chapter 4 TLS Vulnerabilities and Threat Intelligence 55
4.1 Context and Motivation . . . 55
4.2 FINSEC Project . . . 57
4.3 Planning and Integration . . . 59
4.3.1 STIX Output in TLSAssistant . . . 60
4.3.2 FINSEC Connector . . . 62
4.3.3 API and Queue Handling . . . 63
4.4 Integration with Risk Assessment . . . 65
4.5 Lessons Learned and Discussion . . . 66

Chapter 5 From Standalone Tool to Collaborative Framework 68
5.1 Discussion: Challenges and Limitations . . . 68
5.2 Architecture Definition . . . 69
5.2.1 Modules Characterization . . . 71
5.2.2 Standards . . . 80
5.3 Discussion . . . 84

Chapter 6 An Assisted Methodology to Evaluate Security Compliance 87
6.1 Banking Standards . . . 88
6.1.1 PSD2 . . . 88
6.1.2 PCI-DSS . . . 89
6.1.3 Discussion . . . 90
6.2 National TLS Guidelines . . . 90

6.3 Challenges . . . 92
6.4 Related Work: Tools for Compliance Analysis . . . 94
6.5 Compliance Methodology . . . 97
6.5.1 Recommendations Collection . . . 97
6.5.2 Single Guideline . . . 99
6.5.3 Multiple Guidelines . . . 100
6.6 Prototyping . . . 103
6.6.1 SAT . . . 104
6.6.2 JSON Schema . . . 106
6.6.3 Reference Use Cases . . . 108
6.7 Discussion . . . 109

Chapter 7 Impact on Collaborations **112**
7.1 eIDAS Authentication Scheme . . . 112
7.2 Sensitive SaaS Configuration . . . 113
7.3 Continuous Monitoring of Enterprise Infrastructures . . . 115
7.4 Feedback on Agency-issued TLS Guidelines . . . 116

Chapter 8 Conclusions and Future Work **118**
8.1 Future Work . . . 119

Bibliography **120**

Appendix A Survey Questionnaires' Content **136**

Appendix B Leaves Content of TLS Attack Trees **138**
B.1 Break Confidentiality . . . 138
B.2 Break Authentication . . . 140

List of Figures

2.1 TLS 1.2 full handshake 15
2.2 TLS 1.2 *Client Hello* content 16
2.3 TLS 1.2 *Server Hello* content 17

3.1 testssl.sh report snippet 26
3.2 TLSAssistant v1.0 architecture 32
3.3 TLSAssistant v1.0 report (excerpt) 33
3.4 An example of an attack tree in the context of TLS 34
3.5 Attack tree depicting the goal of breaking TLS message confidentiality (simplified) 35
3.6 Attack tree depicting the goal of breaking TLS message integrity 36
3.7 Attack tree depicting the goal of breaking TLS authentication (simplified) 37
3.8 TLSAssistant v1.1 architecture 38
3.9 Time (in minutes) to fix a security issue 50

4.1 FINSEC reference architecture 56
4.2 FINSEC *Dashboard* 57
4.3 A simplified STIX SDO ecosystem with its possible relationships 58
4.4 A simplified FINSEC architecture 59
4.5 STIX output for the Bar Mitzvah attack 61
4.6 TLSAssistant v1.2 architecture 62
4.7 Integration of TLSAssistant in the FINSEC architecture 64

4.8 Integration of TLSAssistant in a Risk Assessment Engine model 66

5.1 TLSAssistant v2 architecture 69
5.2 TLSAssistant v2 report (excerpt) 78
5.3 TLSAssistant v2 scoreboard 79
5.4 TLSAssistant v2 logo 85

6.1 sslyze compliance output (excerpt) 94
6.2 TLS-Scanner output fragment 96
6.3 Defined partial orders 101
6.4 Comparison approaches 102
6.5 Compliance module architecture 103
6.6 generate-after-one Apache output fragment 110
6.7 compare-to-many Apache report fragment 111

7.1 CIE ID infrastructure (simplified) 113
7.2 Examples of SaaS integrations 114
7.3 Grafana output for a single host (excerpt) 115
7.4 Grafana output summed per-host (excerpt) 117

List of Tables

2.1 Main differences introduced with TLS 1.3 19
2.2 List of known protocol-related TLS attacks 20

3.1 TLS analyzers comparison - Server side 24
3.2 TLS analyzers comparison - Mobile side 28
3.3 List of suggested mitigations for webservers 30
3.4 List of suggested mitigations for Android apps 30
3.5 Demographics: Participants' academic background 40
3.6 Demographics: Participants' technical background 41
3.7 Experimental design 44
3.8 Correctness in fixing an incorrect TLS configuration 48
3.9 Analysis of correctness (GLMM) 48
3.10 Time (in minutes) to fix a security issue 50
3.11 Analysis of time (GLMM) 51
3.12 Analysis of survey questionnaire (Fisher's test) 51
3.13 Analysis of survey questionnaire 52
3.14 Analysis of survey questionnaire (Mann-Whitney test) 53

6.1 RFC 2119 requirement levels meaning 91
6.2 TLS protocol compliance across guidelines (excerpt) 92
6.3 TLS extensions recommendations across guidelines (excerpt) 97

6.4 Supported groups recommendations across guidelines (excerpt) 98
6.5 Single Guideline - Compliance evaluation 99
6.6 SAT solvers evaluation 104

A.1 Questions in the survey questionnaire 136

B.1 Break Confidentiality - Full Leaves 138
B.2 Break Authentication - Full Leaves 140

Chapter 1

Introduction

Since its first version, published as an RFC in 1999, Transport Layer Security (TLS) has established itself as a powerful suite of protocols, able to protect insecure communications by providing both confidentiality and integrity to the communicating parties. Despite the fact that TLS has undergone multiple updates and sometimes even significant protocol changes, its implementations still contain the protocol's biggest flaw: they do not provide strong security out-of-the-box and must be configured to provide the desired security benefits. The problem is worsened by the fact that TLS is not a part of a single software, but rather a collaboration between a webserver, a TLS library, and a host operating system. All three must be handled properly in order to provide secure communications.

Kotzias et al. [KRA+18] observe that the TLS ecosystem has undergone significant changes over the past decade, with notable shifts in the cipher suites and TLS extensions offered by clients and accepted by servers. In particular, they report that clients, particularly web browsers, are quick to adopt new algorithms but slow to drop support for older ones. They also encounter a substantial amount of client software that likely provides insecure ciphers by accident. Backward compatibility is especially dangerous and impactful when we notice that almost 5,000,000 websites still support SSLv3, a protocol no longer considered secure since 2015 [Sho22a, BTPL15]. More than 233,000 webservers are still vulnerable to a six years old attack called Sweet32 [Sho22c, BL16] and approximately 203,000 still use an out-of-date version of OpenSSL that can be tricked into leaking the victim's memory contents. [Sho22b, Syn14]. All these settings, that become "misconfigurations" when new attacks and vulnerabilities are discovered, can be secured by editing the webserver's configuration (for SSLv3 and Sweet32) or by upgrading the installed TLS library (for Heartbeat).

In order to provide the best possible service, a system administrator must comprehend the repercussions of each configuration element and make decisions in accordance with this knowledge. Each of the customizable preferences has nontrivial implications that can only be fully grasped

through an in-depth examination of the current state of the art, a task that a system administrator may not have the time or desire to complete. Moreover, a user study conducted by Tiefenau et al. [THKvZ20a] demonstrated that even experienced administrators find it difficult to predict the immediate effects of applying an update and are extremely concerned about potential downtimes. It also appears that they prefer suggestions from peers who have experienced a similar situation to vendor-provided information. This causes (potentially) valid configurations to become obsolete and insecure over time due to the lack of upgrades.

An alternative to conducting a literature review would be to rely on cutting-edge tools that are able to identify incorrect configurations and provide the administrator with information regarding potential dangers to the system's safety. However, relying on these tools does not absolve the administrator of his or her responsibility to comprehend how system components interact with one another, what security issues may result from misconfigurations and how to actually mitigate the detected vulnerabilities. Thus, the system administrator must weigh the risk of overlooking a security vulnerability by not studying best practices against the risk of unintentionally introducing security flaws due to a lack of knowledge and understanding. In addition, when shifting the focus from a single webserver to a broader perspective, additional factors such as security standards (e.g., [PCI18]) and national guidelines (e.g., [Age20]) must be taken into account.

1.1 Goals and Contributions

This thesis focuses on the state-of-the-art regarding the practical applications of TLS, ranging from configuration difficulties to interoperability issues in various scenarios. Our goal is to bridge the gap between cutting-edge research and the system administrators and app developers that could benefit from it but who either lack the time or the knowledge to access it themselves. The work begins with the observation that a subset of TLS analyzers share a significant limitation: they provide little to no guidance on how to actually mitigate the detected vulnerabilities. We sought to address this deficiency by providing actionable hints, a set of literature-based instructions that would assist system administrators in deploying secure instances of the TLS protocol. We chose to expand the use of these hints, demonstrate their effectiveness and usability in the context of a user study, and evaluate their impact in a real-world scenario. As a result of our work over the years, we have decided to broaden our risk assessment efforts. Indeed, the ability to provide an "overall assessment" of a system, analyze evolving risks, and monitor new vulnerabilities is of the extreme significance, given that TLS serves as a central component of many modern technologies. Finally, we worked on the development of a methodology that would permit the evaluation of a system's compliance with a set of guidelines. Thus going beyond the protection of a single system and aiming to secure a service as part of a larger, more regulated ecosystem.

To summarize, this work provides the following main contributions:

- a study of the state-of-the-art in terms of TLS analyzers capable of inspecting webservers' configuration, performing vulnerability detection and compliance analysis, with a focus on their features, limitations and application domains;
- the design and development of TLSAssistant, a modular and extendable framework that can assist system administrators and app developers in correctly configuring TLS deployments. The tool - freely available on GitHub [Sec] - is designed to be easily extensible (to simplify the upgrade process, allowing third-party and internal contributors to rely on a reliable core system that can automatically integrate new modules) and interoperable with other cybersecurity tools or platforms. The strengths of the tool include its ability to generate actionable reports, containing accurate, concise information and code fragments that can be copied and pasted into the webserver configuration in order to mitigate the detected vulnerabilities and ensure the compliance with agency-issued guidelines;
- the development of a methodology that can assist users in assessing the compliance of their systems against a single or multiple agency-issued security guidelines;
- the application of TLSAssistant in a variety of corporate scenarios and projects, describing its impact and the lessons learned from its deployment.

1.2 Thesis Structure

The thesis is structured as follows.

Chapter 2 - Background

This chapter introduces the foundational concepts and definitions of this work. It will focus on the TLS suite, including its history, protocols, functions, and vulnerabilities.

Chapter 3 - Actionable Hints for Configuring TLS

This chapter introduces the concept of "actionable hints" and describes our evaluation of their applicability and impact. To achieve this, we will *(i)* present the state-of-the-art in TLS analyzers, highlighting their features and a shared limitation, *(ii)* illustrate the concept of actionable hints and the work of collecting known mitigations to create reliable actionable reports, *(iii)* describe the design of an automated tool called TLSAssistant, and *(iv)* the user study designed to evaluate its usability.

Chapter 4 - TLS Vulnerabilities and Threat Intelligence

This chapter presents TLSAssistant integration into FINSEC, a framework for predictive and collaborative financial infrastructure security. Integrating multiple security intelligence sources

and vulnerability evaluation and scoring could improve the other integrated services. The sections will *(i)* describe the context in which this integration took place, *(ii)* provide an overview of the FINSEC platform by describing its structure, how it works, and the parties involved, *(iii)* demonstrate how we overcame the integration challenges, *(iv)* describe the information exchange with the Risk Assessment Engine as implemented in FINSEC, and *(v)* discuss the lessons learned.

Chapter 5 - From Standalone Tool to Collaborative Framework

In this chapter, we will *(i)* discuss the challenges and limitations encountered while working on TLSAssistant over the years, *(ii)* introduce a new architecture for TLSAssistant v2, designed to be modular and easily extendable, and *(iii)* discuss the lessons learned during the redesign phase.

Chapter 6 - An Assisted Methodology to Evaluate Security Compliance

In this chapter, we will *(i)* describe the circumstances that led to the study of agency-issued compliance guidelines, *(ii)* introduce the existing guidelines, *(iii)* present the state-of-the-art in terms of compliance assessment tools, *(iv)* illustrate the defined use cases, the proposed methodology to implement them, and *(v)* describe the prototyping work.

Chapter 7 - Impact on Collaborations

This chapter will focus on the scientific, technical, and social impact of a series of collaborations that have taken place over the past three years.

Chapter 8 - Conclusions and Future Work

In this chapter, we conclude the thesis work and offer some insight into potential future directions.

The content of this thesis is based on the following peer-reviewed conference and journal articles:

1. Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. 2019. *Lost in TLS? No More! Assisted Deployment of Secure TLS Configurations.* In: Foley, S. (eds) Data and Applications Security and Privacy XXXIII. DBSec 2019. Lecture Notes in Computer Science, vol 11559. Springer, Cham. [MRS19]

2. Andrea Bisegna, Roberto Carbone, Mariano Ceccato, Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, Alessandro Tomasi, and Emanuele Viglianisi. 2020. *6. Automated Assistance to the Security Assessment of API for Financial Services.* In Cyber-Physical Threat Intelligence for Critical Infrastructures Security: A Guide to Integrated Cyber-Physical Protection of Modern Critical Infrastructures. Now Publishers. [BCC+20]

3. Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. 2021. *Do Security Reports Meet Usability? Lessons Learned from Using Actionable Mitigations for Patching TLS Misconfigurations.* In Proceedings of the 16th International Conference on Availability, Reliability and Security (ARES 21). Association for Computing Machinery, New York, NY, USA, Article 141, 1–13. [MCSR21a]

4. Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, and Alessandro Tomasi. 2020. *TLSAssistant Goes FINSEC A Security Platform Integration Extending Threat Intelligence Language.* In Cyber-Physical Security for Critical Infrastructures Protection: First International Workshop, CPS4CIP 2020, Guildford, UK, September 18, 2020, Revised Selected Papers. Springer-Verlag, Berlin, Heidelberg, 16–30. [MRST20]

5. Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. 2022. *Empirical Validation on the Usability of Security Reports for Patching TLS Misconfigurations: User- and Case-Studies on Actionable Mitigations.* Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, 13(1), 56–86. [MCSR22]

6. Matteo Rizzi, Salvatore Manfredi, Giada Sciarretta, and Silvio Ranise. (2022). *A Modular and Extensible Framework for Securing TLS.* In Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy. CODASPY '22: Twelveth ACM Conference on Data and Application Security and Privacy. ACM. [RMSR22b]

7. Matteo Rizzi, Salvatore Manfredi, Giada Sciarretta, and Silvio Ranise. 2022. *Demo: TLSAssistant v2: A Modular and Extensible Framework for Securing TLS.* In Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies (SACMAT '22). Association for Computing Machinery, New York, NY, USA, 271–272. [RMSR22a]

Chapter 2

Background

This chapter presents the fundamental concepts and definitions upon which this work is founded. Specifically, it will cover the Transport Layer Security (TLS) suite of protocols, including *(i)* its history, *(ii)* the protocols that constitute it, their functioning and *(iii)* their vulnerabilities.

2.1 Historical Notes

TLS (short for "Transport Layer Security") originated as a compromise between Netscape [Inc22] and Microsoft [Mic22a] proposals for how to secure HTTP [IET22]. Netscape created Secure Sockets Layer (SSL) 2.0 in 1994 and released it alongside its browser, Navigator 1.1, in 1995 [Ris22]. Due to the protocol's severe cryptographic and practical flaws, two companies decided to develop an alternative. Netscape introduced SSL 3.0 in 1995 [Net], while Microsoft, its competitor, revised SSL 2.0 and developed the PCT protocol. PCT v1.0 was backward compatible with SSL 2.0, but this protocol never gained traction on outside of the Microsoft ecosystem [Ris22].

Diverse industry and community members opposed a protocol fork, so Consensus Development (already responsible for the SSL 3.0 reference implementation) arranged a meeting between Netscape and Microsoft [Die14]. After Microsoft and Netscape agreed to support the IETF's open process for standardizing the SSL protocol, the standardization process began. As part of the agreement, SSL was renamed to avoid giving the impression that the IETF supported Netscape's protocol. TLS 1.0 (which was actually SSL 3.1) was published as an official RFC in 1999 [DA99].

2.2 Transport Layer Security

The main goal of the TLS protocol is to ensure the confidentiality and integrity of data exchanged between two communicating applications. It is composed of two layers: the Handshake protocol and the Record protocol [DR08].

Record protocol deployed on top of a transport protocol (such as TCP) and encapsulates the messages coming from higher levels. It ensures confidentiality by employing symmetric encryption algorithms with keys generated for each connection and integrity by calculating the hash of the messages being sent. The used keys and algorithms are those agreed upon during the handshake.

Handshake protocol deployed on top of the Record protocol, negotiates the shared secret (i.e. Master Secret) from which all required keys will be derived. The identity of the connecting peers is authenticated by using public key cryptography and it is generally required for at least one entity. The handshake protocol supports two special messages: *(i) Change Cipher Spec*, which changes the session's ciphering strategy, and *(ii) Alert*, which broadcasts potential alerts.

Four versions of the TLS protocol have been released over the past 24 years:

- **TLS 1.0** published in 1999 as a standardization of the SSL 3.0 protocol by IETF [DA99] (see Section 2.2);
- **TLS 1.1** introduced in 2006 to enhance protection against CBC attacks [DR06];
- **TLS 1.2** published in 2008 with significant security enhancements (such as supporting authenticated encryption and AES cipher suites) and extensions support [DR08];
- **TLS 1.3** published in 2018 with modifications to the handshake, removal of legacy features, and simplification of cipher suite negotiation [Res18].

According to the SSL Pulse scan conducted by Qualys [Qua22a], 99.8% of the most popular websites [Ser22] support TLS 1.2, making it the most widely supported protocol to date. Due to its wide availability and the number of protocol-related vulnerabilities discovered in recent years (which will be discussed in Section 2.3), from here on out we will focus on TLS 1.2 and its handshake.

2.2.1 TLS Handshake

The purpose of TLS 1.2 handshake is to allow the parties to exchange all information required to establish a secure session. Figure 2.1 depicts the entire handshake (with asterisks indicating

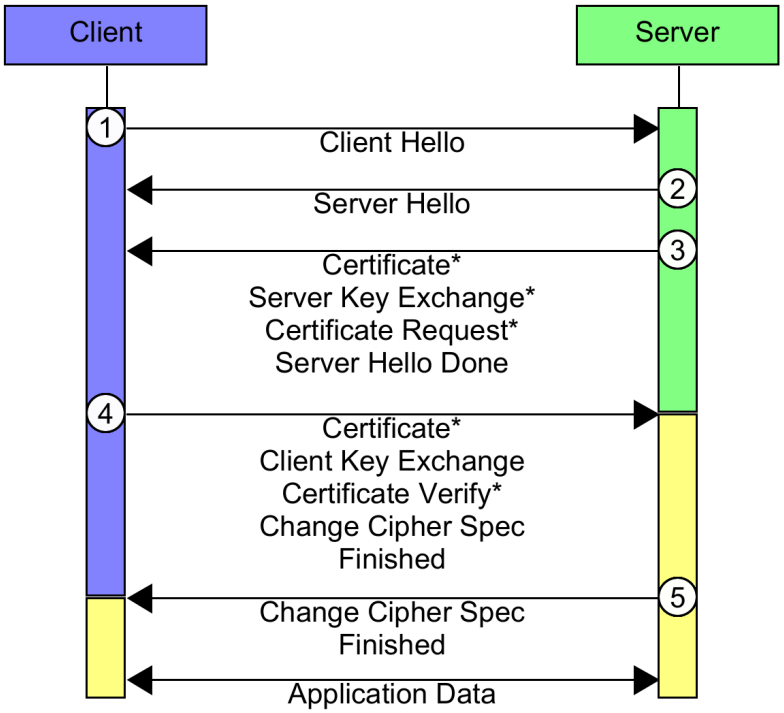


Figure 2.1: TLS 1.2 full handshake

optional data and the yellow segments showing the encrypted transmission). Messages 1-2 start the handshake and exchange information that will be used to identify the session and the current connection. Messages 3 and 4 are used to initiate the authentication process (i.e., by sending and verifying X.509 certificates) and to exchange the keying material necessary to construct the Master Secret, from which all required keys will be derived. Theoretically, both client and server authentication are optional, but the most frequent authentication scenarios involve either unilateral (server-authenticated) or mutual authentication (both parties authenticated). The *Change Cipher Spec* message, sent by both parties immediately before the handshake's final message, instructs the Record protocol to use the negotiated algorithms and keys (upon reception). Finally, the *Finished* is the first encrypted message sent over the secure channel and is used to verify the handshake's validity.

In the following, we will discuss the content of each message:

Client Hello transmitted by the client to initiate the handshake. As shown in Figure 2.2, it is composed of:

Figure 2.2: TLS 1.2 *Client Hello* content

client_version - the version of TLS that the client wishes to use to communicate with;

random - random value obtained by chaining the timestamp (32 bit in UNIX time) and a randomly generated nonce (28 bytes);

session_id - the ID of a session the client wishes to use;

cipher_suites - a list of supported cipher suites. The list of cipher suites contains the client's preferred cryptographic algorithms. Each element specifies a unique combination of a key exchange, bulk encryption, and MAC algorithm;

compression_methods - a list of supported compression methods;

client_hello_extension_list - a list of requested additional functionalities.

```
Handshake Protocol: Server Hello
   Handshake Type: Server Hello (2)
   Length: 76
   Version: TLS 1.2 (0x0303)
 > Random: 0fae1b7db5e15efd9e660d00fa4c466c2edb3ae63cd4c52c130c305064401d71
   Session ID Length: 0
   Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
   Compression Method: null (0)
   Extensions Length: 36
 > Extension: server_name (len=0)
 > Extension: renegotiation_info (len=1)
 > Extension: ec_point_formats (len=4)
 > Extension: session_ticket (len=0)
 > Extension: application_layer_protocol_negotiation (len=11)
```

Figure 2.3: TLS 1.2 *Server Hello* content

Server Hello is sent by the server once it receives a *Client Hello*. It contains, as shown in Figure 2.3:

server_version - this field will contain the lower protocol version suggested by the client and the highest supported by the server;

random - as for the client, it is obtained by chaining the timestamp and a randomly generated nonce;

session_id - the ID of the session corresponding to this connection;[1]

cipher_suite - the chosen cipher suite from those proposed by the client;

compression_method - the compression method chosen among those proposed by the client;

server_hello_extension_list - the required set of extensions.

Certificate

server-side transmitted if the client requests an authenticated connection (via cipher suite proposal). The server must send a list of X.509v3 certificates (from the host one up to the root CA);

client-side transmitted if the client has received a *Certificate Request* message. It must send a list of X.509v3 certificates (from the host one up to the root CA) complying with the requirements contained in the message itself.

[1] A session consists of a secure channel that can contain multiple connections. Multiple connections within a session share a subset of the terms of the initial handshake (e.g., session ID and cipher suite).

Server Key Exchange sent if the delivered certificate lacks sufficient information to establish a secure channel between the server and client. This occurs when the chosen key exchange algorithm is `dh_anon`, `dhe_rsa`, or `dhe_dss` (obsolete). The content of the message depends on the chosen algorithm:

dh_anon (anonymous Diffie-Hellman)

dh_p DH prime modulus;

dh_g DH generator;

dh_Ys the server's public value ($g^X \mod p$);

dhe_rsa (Ephemeral Diffie-Hellman with RSA)

- the server signs the DH parameters using its private key to create a pre-master secret.

Certificate Request transmitted by a non-anonymous server to request a client authentication. It contains the following values:

certificate_types the accepted certificate types (e.g., rsa_sign, rsa_fixed_dh);

supported_signature_algorithms a set of supported signatures (e.g., SHA512, SHA1 DSA or ecdsa_sha1);

certificate_authorities a list of acceptable authorities.

Server Hello Done is an empty message that signals the end of the server's hello messages.

Client Key Exchange configures the premaster secret that will later be used to generate the *Master Secret*.[2] This message contains one of the following structures based on the agreed key exchange algorithm selected (as specified in the *Server Hello*):

RSA-Encrypted Premaster Secret Message a 48-byte premaster secret is generated by the client and encrypted using the server's public key (acquired from the received certificate);

Ephemeral Diffie-Hellman public exponent (dhe) the client sends dh_Yc, the remaining DH public value;

Static Diffie-Hellman exponent (fixed_dh) the message will be empty because the public exponent has already been sent within the certificate.

Certificate Verify sent to provide client certificate verification. It is a signed concatenation of all handshake messages (sent and received) prior to *Client Key Exchange*.

[2]The *Master Secret* is a 48 bytes long string used as a source to derive all session-related keys.

Table 2.1: Main differences introduced with TLS 1.3

Status	Component	Reason
Removed	not-AEAD ciphers	avoid attacks on legacy ciphers
	RSA key exchange	always provide forward secrecy
	broken hash algorithms (MD5, SHA-1)	avoid SLOTH and similar attacks
	Change Cipher Spec message	streamline the handshake
	data compression	avoid CRIME attack
	session renegotiation	avoid renegotiation attacks
Added	0-RTT mode for a quick resumption	increase resumption speed
	`EncryptedExtensions` msg	avoid transmitting preferences in plaintext
Changed	msg encryption starts after the *Server Hello*	allow Client certificate encryption
	`Hello` content (structure unchanged)	extend Handshake capabilities

Change Cipher Spec its reception signifies the beginning of the agreed-upon encryption strategy. It is transmitted by both the client and the server and consists of a single "1" byte.

Finished both parties generate a MAC by hashing the entire handshake. It is used to indicate the algorithm's completion.

2.2.1.1 TLS 1.3

In August 2018, the IETF published RFC 8446 as a proposed standard after four years of work and 28 drafts [Dat18]. The document explains how TLS 1.3 operates and, in the first chapter, outlines the main differences between TLS 1.2 and 1.3. Table 2.1 provides a summary of the major differences, which include profound changes to the handshake, the elimination of support for several legacy components (such as cipher suites and hash algorithms), a simplified key exchange, and a redesign of key derivation functions. The presence of these improvements fixed at their source a number of vulnerabilities exploitable by some still-viable attacks for TLS 1.2.

2.3 Misconfigurations and Vulnerabilities

TLS protocol versions prior to 1.3 are prone to a wide range of vulnerabilities. While some of the attacks are the result of flaws in the logic of the protocol, other exploit the support of weak cryptographic aspects (e.g., weak ciphers and hash functions) or (in)voluntary weakening of security properties (such as accepting self-signed certificates [Devc]).

In Table 2.2 we detail a set of well-known TLS attacks, each line contains: *(i)* the name given by the authors; *(ii)* the feature or weakness exploited and *(iii)* a brief description on how the

attack can be mounted. Among all the attacks, here we provide some details on four that will be referenced throughout the thesis: CRIME [NIS12], BREACH [GHP12], Bar Mitzvah [Man] and ROBOT [BSY18].

CRIME and BREACH are related to the availability of DEFLATE, a compression algorithm that reduces an input's size by replacing duplicate strings with a reference to their last occurrence. Given that neither TLS nor HTTP conceal the size of each message, this information leakage can be exploited by an attacker to steal sensitive data. Assuming a desire to steal session cookies, the attack is carried out by injecting different characters into the client's messages in an attempt to guess the cookie (e.g., using a controlled JavaScript loaded by the victim). Thanks to DEFLATE, the response will be larger if the guess is incorrect and the characters are not part of the cookie. Alternatively, if the attacker guessed correctly, the size will not change. This attack is classified as CRIME if it exploits TLS compression; otherwise, it is classified as BREACH;

Bar Mitzvah utilizing the invariance weakness of the RC4 stream cipher, an attacker can retrieve the session cookie by attempting to guess the least significant bits of the keystream. While the attack has been known since 2013, nearly 4.2% of the world's most popular websites continue to support it (according to the Qualys SSL Pulse dashboard [Qua22a]);

ROBOT due to the availability of the PKCS#1v1.5 padding algorithm in RSA, an attacker is able to extract the private key of the session and breaking the message confidentiality. By using an adaptive chosen-ciphertext attack, based on Daniel Bleichenbacher's chosen-ciphertext attack [Ble98], the victim is forced to leak information that help the attacker in guessing the key. The key may then be employed to decrypt HTTPS traffic transmitted between the TLS server and the user's browser.

Table 2.2: List of known protocol-related TLS attacks

Name	Vulnerability	Attack
3SHAKE [MI14]	Renegotiation feature	Complete three handshakes with incorrectly placed certificates
ALPACA [BDM+21]	Availability of multiple protocols on the same server	Redirect traffic from one subdomain to another, resulting in a valid TLS session
Bar Mitzvah [Man]	RC4 steam cipher	Extract weak keys by targeting the first 100 bytes of the ciphertext

Continued on next page

Table 2.2 – *Continued from previous page*

Name	Vulnerability	Attack
BEAST [Gre11]	Initialization vector in cipher block chain	Guess the plaintext to retrieve the symmetric key
BREACH [GHP12]	HTTP compression mechanism	Request data from the server in order to guess the response body (note: without downgrading the SSL/TLS connection)
CRIME [NIS12]	TLS header compression mechanism (DEFLATE)	Continuously request data from the server in order to decrypt the session cookies (inferring the encryption)
DROWN [Por]	SSLv2 weakness due to the use of export ciphers	Decrypt intercepted TLS connections by connecting to an SSLv2 server that uses the same private key
Lucky 13 [AP13]	CBC-mode weakness due to HMAC-SHA1 decryption failure information leakage	Replace the last bytes with chosen bytes and monitor the transmission time
RC4 NOMORE [VP]	Bias in the generation of the "random" keys of the RC4 stream cipher	Statistically analyze the Fluhrer-McGrew biases
Raccoon [MBA+21]	Premaster secret management	Construct an oracle by exploiting timing measurements
Renegotiation attack [MIT09]	Renegotiation feature	Inject plaintext within an existing data stream
ROBOT [BSY18]	PKCS#1v1.5 padding in RSA	Force the victim to leak information that help the attacker in guessing the key
SLOTH [Duc]	Availability of weak hash functions	Request an RSA-MD5 certificate signature and looking for collisions
SSL POODLE [MDK]	SSLv3 weakness due to the missing validation of padding bytes	Downgrade to SSLv3 and guessing the padding in order to slowly recover plaintext
Sweet32 [BL16]	64-bit block ciphers	Mount a birthday attack which creates collisions

Continued on next page

Table 2.2 – *Continued from previous page*

Name	**Vulnerability**	**Attack**
TLS POODLE [MIT14b]	Availability of CBC-mode for ciphers	Exploit the non-determinism during decryption

Improving the accessibility of TLS libraries can help mitigate all of these issues, as the availability of legacy or insecure features whose use is not discouraged is what leads to misconfigurations and servers susceptible to attacks that date back a decade or more. Providing a strong default configuration for general-purpose use, streamlining TLS software configuration, and making backwards compatibility features opt-in can significantly reduce the likelihood of misconfiguration resulting from the use of default settings [KRA+18].

Chapter 3

Actionable Hints for Configuring TLS

In this chapter, we introduce the concept of actionable hints and describe the work we performed to evaluate their applicability and impact. To this aim, we will *(i)* present the state-of-the-art in terms of TLS analyzers, highlighting their features and a shared limitation, *(ii)* illustrate the concept of "actionable hints" and the work of collecting known mitigations to create reliable actionable reports, *(iii)* describe the design of an automated tool, able to assist system administrators and app developers in securing their TLS deployments and *(iv)* describe the user study conducted to evaluate the tool's usability when patching defective TLS configurations and the effectiveness of reports containing actionable reports.

3.1 Related Work: TLS Analyzers

There exists a wide amount of TLS analyzers freely available online. With the intention of creating an open source downloadable tool, we decided to filter the available analyzers based on the ability to download and run them locally (i.e., excluding powerful webapps such as Qualys SSL Server Test [Qua22b] and Cryptosense Discovery [Cry]), read, study, and redistribute their source code (i.e., excluding closed source or obsolete tools), and with the greatest number of available vulnerability checks.

Table 3.1: TLS analyzers comparison - Server side

Checks	3Shake_chk	sslscan	sslyze*	testssl.sh	TLS-Scanner*	tlsfuzzer*
Last updated	Nov 2015	Jul 2022	May 2022	Sep 2022	Jul 2022	Sep 2022
Language	Python 2	C	Python 3	Bash	Java	Python 3
Available protocols	◗	●	●	●	●	●
Available ciphers	○	●	●	●	●	●
Cipher preference	○	●	○	●	◗	○
Forward secrecy	○	◗	●	●	●	○
Elliptic curves	○	●	●	●	●	○
DH groups	○	●	●	●	●	○
Negotiated protocol	◗	○	○	●	○	○
Negotiated cipher	○	○	○	●	○	○
Certificate details	○	●	●	●	●	○
Certificate validity	○	●	●	●	●	○
Chain of trust	○	●	●	●	●	○
CRL	○	○	○	●	○	○
OSCP	○	●	●	●	●	○
Cert. transparency	○	●	●	●	●	○
HSTS	○	○	○	●	●	○
HPKP	○	○	○	●	●	○
Webserver detection	○	○	○	●	○	○
CRIME	○	◗	◗	●	●	○
BREACH	○	○	○	●	●	○
Sweet32	○	◗	◗	●	●	◗
POODLE	○	◗	◗	●	●	○
POODLE variants	○	◗	◗	○	●	○
ROBOT	○	◗	●	●	●	●
Drown	○	◗	○	●	●	●
RC4	○	●	●	●	○	◗
CCS Injection	○	●	●	●	○	◗
Lucky 13	○	◗	◗	◗	○	●
Heartbleed	○	●	●	●	●	●
Logjam	○	◗	●	●	●	◗
Freak	○	◗	◗	●	○	◗

Continued on next page

Table 3.1 – *Continued from previous page*

Checks	3Shake_chk	sslscan	sslyze*	testssl.sh	TLS-Scanner*	tlsfuzzer*
BEAST	❍	◗	◗	●	❍	◗
3SHAKE	●	❍	❍	◗	●	◗
SLOTH	❍	❍	❍	❍	❍	●
ALPACA	❍	❍	❍	❍	●	❍
Raccoon	❍	❍	❍	❍	●	❍

Table 3.1 shows the comparison between the six most peculiar ones, comparing them to find the best state-of-the-art standalone tools with a wide amount of available checks (elements marked with an asterisk were not part of the original comparison). [1] Each check is identified depending on the type of information resulting in their output. In particular, ●, ◗ and ❍ mean a complete, partial (which can be inferred using other explicit detections) or total lack of detection, respectively. The evaluated tools are:

TLS Extended Master Secret Extension Checker [You] (abbreviated as "3Shake_chk" for spacing purposes): is a simple script that checks if the target server supports the only available mitigation for the 3SHAKE attack [MI14]: the *extended_master_secret* TLS extension [BDLP+15]. The output shows, for each available version of TLS (up to 1.2), if the extension request has been accepted by the server;

sslscan [rbs17]: is tool capable of performing a variety of checks without focusing on specific deployment aspects. Despite its features, it does not implement many vulnerability checks directly;

sslyze [Diq22]: is another powerful tool, similar to sslscan in terms of available checks and similarly limited;

testssl.sh [Wet22]: is a fully-featured tool able to analyze a server's configuration. The tool is mainly focused on detecting weaknesses and various configuration issues while being able to perform a wider set of tests. Among these, testssl.sh can provide detailed insights on both certificate (e.g., chain of trust, revocation data, OSCP settings, HSTS and HPKP status) and webserver (e.g., security headers, fingerprinting, key usage) status. The generated report contains the results for all the performed analysis, associated with a color that signals the severity of the detected result (going from green to red);

[1] Since our first comparison in 2018, many available tools have undergone significant enhancements, but their focus (and corresponding feature gap) have remained unchanged.

```
 Testing vulnerabilities

 Heartbleed (CVE-2014-0160)                not vulnerable (OK), no heartbeat extension
 CCS (CVE-2014-0224)                       not vulnerable (OK)
 Ticketbleed (CVE-2016-9244), experiment.  not vulnerable (OK), no session tickets
 ROBOT                                     not vulnerable (OK)
 Secure Renegotiation (CVE-2009-3555)      not vulnerable (OK)
 Secure Client-Initiated Renegotiation     not vulnerable (OK)
 CRIME, TLS (CVE-2012-4929)                VULNERABLE (NOT ok)
 BREACH (CVE-2013-3587)                    no HTTP compression (OK)  - only supplied "/" tested
 POODLE, SSL (CVE-2014-3566)               not vulnerable (OK)
 TLS_FALLBACK_SCSV (RFC 7507)              Downgrade attack prevention supported (OK)
 SWEET32 (CVE-2016-2183, CVE-2016-6329)    VULNERABLE, uses 64 bit block ciphers
 FREAK (CVE-2015-0204)                     not vulnerable (OK)
 LOGJAM (CVE-2015-4000), experimental      common prime with 4096 bits detected: RFC3526/Oakley Group 16
                                                 (4096 bits), but no DH EXPORT ciphers
 BEAST (CVE-2011-3389)                     TLS1: ECDHE-RSA-AES256-SHA
                                                 DHE-RSA-AES256-SHA
                                                 DHE-RSA-CAMELLIA256-SHA AES256-SHA
                                                 CAMELLIA256-SHA
                                                 ECDHE-RSA-AES128-SHA
                                                 DHE-RSA-AES128-SHA
                                                 DHE-RSA-SEED-SHA
                                                 DHE-RSA-CAMELLIA128-SHA AES128-SHA
                                                 SEED-SHA CAMELLIA128-SHA
                                                 IDEA-CBC-SHA
                                                 ECDHE-RSA-DES-CBC3-SHA
                                                 EDH-RSA-DES-CBC3-SHA DES-CBC3-SHA
                                                 EDH-RSA-DES-CBC-SHA DES-CBC-SHA
                                           VULNERABLE -- but also supports higher protocols
                                                 TLSv1.1 TLSv1.2 (likely mitigated)
 LUCKY13 (CVE-2013-0169), experimental     potentially VULNERABLE, uses cipher block chaining (CBC)
                                                 ciphers with TLS. Check patches
 RC4 (CVE-2013-2566, CVE-2015-2808)        VULNERABLE (NOT ok): ECDHE-RSA-RC4-SHA
                                                                RC4-SHA RC4-MD5
```

Figure 3.1: testssl.sh report snippet

TLS-Scanner [Ruh22]: is a state-of-the-art vulnerability scanner able to assist pentesters and security researchers in the evaluation of TLS Server configurations. It is developed by the team of researchers that discovered important attacks against TLS (e.g., ROBOT and DROWN).

tlsfuzzer [Kar22] is a test suite for all SSL and TLS implementations. It comes with a multitude of scripts narrowed for specific use cases. While it uses fuzzing techniques for testing target libraries, the scripts are generally written in a way that verifies correct error handling: according to the author, it does not check only that the system under test did not crash but also that it returned correct error messages. A crucial drawback consists in the fact that tlsfuzzer lacks a proper documentation, and thus its usage is subsequent to understanding the purpose of the available scripts.

All the listed tools work by repeatedly connecting to the target webserver using specifically crafted *Client Hello* messages (see Section 2.2.1). By checking the server's responses (i.e. the *Server Hello* message), the tools are able to infer its configuration.

3.1.1 Mobile Clients

While in a browser the handling of TLS and its certificates is built-in, this is not the case for mobile native applications: a developer can either choose to use one of the many available TLS libraries (e.g., OkHttp [Squ22]) or to implement his own methods. In both cases, an incorrect certificate handling may lead to several authentication-related issues. To detect this kind of misconfigurations, there exist the need for specific tools. Table 3.1 shows the differences between five Android-related analyzers (elements marked with an asterisk were not part of the original comparison):

MalloDroid [FHM+12]: is a Python script (built on top of Androguard [Des]) that performs static analysis on the code of an Android application. It takes as input and Android app installer (`.apk` file) and uses the capabilities inherited from Androguard to decompile the application. Once the script acquires the source code, it *(i)* extracts the set of URLs the app is instructed to connect and checks the validity of their certificates, and *(ii)* identifies if the app is using a non-standard trust manager and checks the related methods;

QARK [Lin19] is a static code analysis tool designed to recognize potential security vulnerabilities. A key feature is the ability to dynamically generate a custom-built testing app, designed to demonstrate the detected issues;

SMV-Hunter [utd15] is a set of tools able to perform large-scale automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android apps. Despite being an interesting research tool, it has never been updated since its first release;

SUPERAndroidAnalyzer [SUP18] is a complete APK file analyzer for a wide set of Android vulnerabilities, including SSL/TLS. It works by decompiling the target apks and performing static analysis following the rules provided in a customizable JSON file called *rules.json*. Its report shows in great detail the detected vulnerabilities, providing the affected line of code and ordering them according to their risk level (i.e. critical, high, medium, low and warning);

Tapioca [Dor]: testing framework that performs a series of unique checks by simulating a MITM. Using different types of packet capture, the tool is able to: *(i)* validate the negotiation between server and client; *(ii)* enumerate all the URLs the app tries to connect; *(iii)* verify if the client correctly validates the received certificates; and *(iv)* (prior packet decryption) search among the messages to locate known strings. Tapioca is very peculiar as its installer permanently alters the host OS as it converts the entire system into a tool.

Table 3.2: TLS analyzers comparison - Mobile side

Checks	MalloDroid	QARK*	SMV-Hunter*	SUPER*	Tapioca
Last updated	Dec 2013	Apr 2019	Oct 2015	Dec 2018	Apr 2022
Language	Python 2	Python 3	Java	Rust	Python 3
TrustManager	●	◗	◗	◗	❍
HostnameVerifier	●	◗	◗	◗	❍
Weak algorithms	❍	●	❍	●	❍
Ignored SSL errors	❍	●	❍	●	◗
Accepting all certificates	❍	◗	❍	●	●
Certificate disclosure	❍	❍	❍	●	❍
KeyStore disclosure	❍	❍	❍	●	❍
Use of insecure SSL methods	❍	◗	❍	●	●

3.1.2 Report Snippet

Besides the wide amount of provided features, all the compared tools share a major limitation: they offer little to no explanation on how to actually mitigate the detected weaknesses. This partially defies their own purpose, as system administrators and app developers will be required to spend a lot of time and effort researching the most appropriate set of mitigations to apply.

To show the required effort, we provide a snippet of the testssl.sh's report (see Figure 3.1). It contains the full list of performed checks matched with their respective presence in the analyzed TLS deployment. The status of each vulnerability is shown with a combination of a string (e.g.; "VULNERABLE") and a color that represents the severity of the finding. Not the shown snippet nor any other part of the report provide any useful insight on how to actually mitigate the detected vulnerabilities.

3.2 Actionable Mitigations

To address the scarceness of available information on how to actively fix a misconfiguration, we decided to work on the concept of "actionable mitigation". An actionable mitigation consists of a concise yet informative report containing a practical mitigation that even an inexperienced user can apply to fix an issue. To evaluate their applicability and impact we decided to build a tool able to perform a wide set of analyses as well as generating reports containing actionable

mitigations.

3.2.1 Mitigations Identification

As a first step toward the creation of an automated analysis tool, we listed all known TLS vulnerabilities (see Section 2.3) and gathered the current best practices to mitigate each one. These mitigations, shown in Tables 3.3 and 3.4, have been collected by fetching information from both scientific literature and each vendor's technical documentation.

The vast majority of the server-related mitigations are applied by changing some lines in the server's configuration file, while the remaining ones are related to vulnerable/outdated support libraries. Android-related weaknesses are instead more strongly tied to improper debugging techniques and a lack of understanding of the certificate validation flow process. Consequently, the resolution to these problems consists in applying known best practices.

The objective was to provide system administrators with the most efficient mitigations that required the least amount of effort (and assuming the use of OpenSSL, the most widely used TLS library at the time [Dat23]). We omitted on purpose any mitigation that would have required recompilation of libraries and webservers. In an effort to enhance the comprehensiveness of the tool, as future work we intend to incorporate mitigations requiring a higher level of expertise to implement.

3.2.2 Initial Tool Design and Assumptions

To assist system administrators and Android developers in correctly configuring their TLS deployment, and close the gap between available analyzers and actual user needs, we decided to build a tool able to perform a wide set of analyses as well as generate reports containing actionable mitigations. Our target users consist of system administrators and app developers without specific knowledge of TLS or its vulnerabilities. The returned actionable mitigation should therefore be as seamless and straightforward as possible.

For the design, we consider a wide set of known TLS attacks, which can be carried out by an attacker who can add, remove, or modify messages sent between the client and the server. In addition, the attacker can control websites, inject malicious JavaScript content into the client (in order to trigger abnormal requests or behaviors), and exploit timing side-channels.

Aiming to analyze the greatest number of vulnerabilities with the smallest number of state-of-the-art tools, we selected a subset of the most suitable state-of-the-art tools. In 2018[2], the choice fell on:

[2]Chapter 5 discusses the current set of integrated tools, which has been upgraded over the years.

Table 3.3: List of suggested mitigations for webservers

Attack	Mitigation
3SHAKE [MI14]	Upgrade to OpenSSL v1.1.0+
ALPACA [BDM+21]	Use ALPN and SNI TLS extensions
Bar Mitzvah [Man]	Disable the RC4 cipher
BEAST [Gre11]	Disable TLS 1.0
BREACH [GHP12]	Disable HTTP compression
CSS Injection [MIT14a]	Upgrade to OpenSSL v1.0.1h+
CRIME [NIS12]	Disable TLS compression
DROWN [Por]	Disable SSLv2
Freak [MIT15]	Disable all export ciphers
Heartbleed [Syn14]	Upgrade to OpenSSL v1.0.1g+
Logjam [Gre]	Disable all export ciphers
Lucky 13 [AP13]	Upgrade to OpenSSL v1.0.1e+
RC4 NOMORE [VP]	Disable the RC4 cipher
Raccoon [MBA+21]	Avoid "static" DH ciphers
Renegotiation attack [MIT09]	Upgrade to OpenSSL v0.9.8m+
ROBOT [BSY18]	Avoid RSA key exchanges
SLOTH [Duc]	Upgrade to OpenSSL v1.0.1f+
SSL POODLE [MDK]	Disable SSLv3
Sweet32 [BL16]	Disable 3DES ciphers
TLS POODLE [MIT14b]	Avoid CBC mode for ciphers

Table 3.4: List of suggested mitigations for Android apps

Weakness	Mitigation
Accepting self-signed certificates [Devc]	Perform a complete certificate validation
Certificate disclosure [Devc]	Obfuscate the required certificate
Keystore disclosure [Deva]	Use a strong and secure password
SSL getInsecure method [Devd]	Substitute any getInsecure with a getDefault
Unsecure TrustManager [Now17]	Switch to an external TLS library
Weak Algorithms [Devb]	Use strong encryption algorithms

testssl.sh [Wet22] chosen among many others (as shown in Section 3.1) due to its focus on vulnerabilities and the wide amount of features;

TLS Extended Master Secret Extension Checker [You] (3SHAKE checker for spacing reasons) added to make our tool able to detect the Triple Handshake attack; [3]

MalloDroid [FHM+12] even if less powerful than Tapioca (see Section 3.1), it was more suitable for our modularity requirement. Tapioca installer's effects on the host system are incompatible with the idea of portability our tool has.

By extracting information from server responses, we are aware that false positives may be detected (e.g., vulnerabilities that cannot be exploited because not all the preconditions for mounting an attack are present). The presence of multiple analysis tools capable of identifying the same vulnerability could reduce the number of false positives. However, this is outside of our initial set of assumptions and could therefore be considered for future improvement.

The tool, called TLSAssistant - whose first iteration was written in Bash - can be invoked via command-line. Among the available parameters, the tool takes as input the target to be evaluated (e.g., the IP address of a server) and outputs a single report file. Its content depends on the detected weaknesses and on the level of verbosity the user chose. Figure 3.2 shows its initial architecture composed of two main components: ANALYZER and EVALUATOR.

ANALYZER. Takes as input a series of parameters depending on which analysis the user wants to run. By design, our tool has a flexible architecture that allows a continuous integration of newer and more sophisticated tools. In 2018, the set of integrated tools consisted of command-line scripts written either in Bash or Python. The integrated tools allow the ANALYZER to take as input: *(i)* a hostname/IP address (optionally specifying the port to scan); *(ii)* an `apk` installer or *(iii)* both of the previous. Once loaded, the module will run each of the tools related to the required scan, collect their reports and transmit them to the EVALUATOR.

EVALUATOR. Core of TLSAssistant, it is responsible for the enumeration of the detected vulnerabilities and the generation of actionable report that will guide system administrators and app developers towards the application of all needed mitigations. The EVALUATOR can be split in two dependent modules:

Vulnerability enumerator collects and analyzes the reports generated by the ANALYZER. By parsing the inputs, this module is able to compile a list containing all the discovered vulnerabilities.

Report handler takes the vulnerability list and, in accordance with the user's choice, renders the final output. While TLSAssistant has been developed to be modular, the only available

[3] In 2018, this was the only tool that provided a complete detection of the Triple Handshake attack. It has eventually been discarded in favor of testssl.sh's updated set of detections.

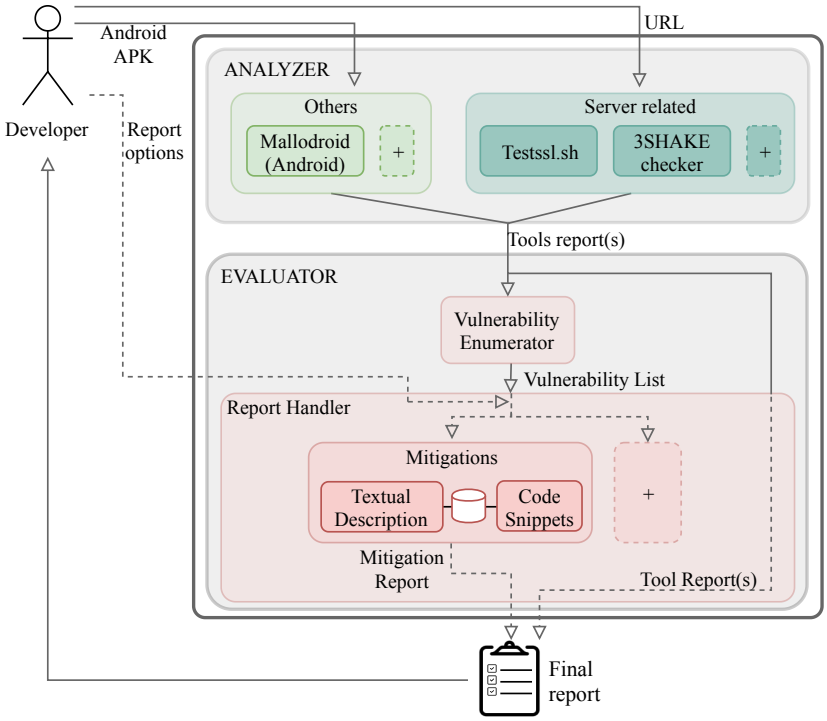


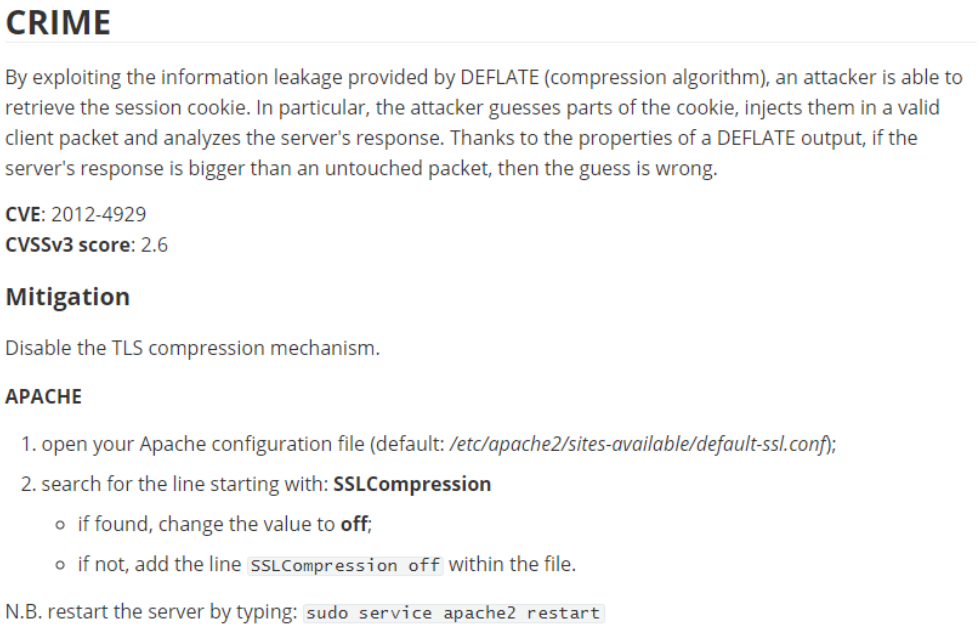
Figure 3.2: TLSAssistant v1.0 architecture

source of information is the **Mitigations** module. It consists of a shared database containing a list of all the known TLS vulnerabilities with their descriptions and related fixes (see Section 3.2.1). The Report handler offers three kind of reports, each version provides the content of the previous one and adds more technical details. For every detected weakness, the main information contained in each version of the report is the following:

v0 mitigations' description. It is the most basic form of report, as it only contains a description of how the related mitigation work;

v1 code snippet. Provides a fragment of code that can be copy-pasted into the webserver's configuration to seamlessly fix the weakness. TLSAssistant can detect any webserver but is only able to provide snippets for Apache HTTP server. We planned to extend the code coverage to more webservers available on the market. A fragment of this report can be seen in Figure 3.3;

v2 tools' individual reports. In addition to our detailed contribution, this kind of report also provides the full set of individual reports generated by each tool.



CRIME

By exploiting the information leakage provided by DEFLATE (compression algorithm), an attacker is able to retrieve the session cookie. In particular, the attacker guesses parts of the cookie, injects them in a valid client packet and analyzes the server's response. Thanks to the properties of a DEFLATE output, if the server's response is bigger than an untouched packet, then the guess is wrong.

CVE: 2012-4929
CVSSv3 score: 2.6

Mitigation

Disable the TLS compression mechanism.

APACHE

1. open your Apache configuration file (default: */etc/apache2/sites-available/default-ssl.conf*);
2. search for the line starting with: **SSLCompression**
   - if found, change the value to **off**;
   - if not, add the line `SSLCompression off` within the file.

N.B. restart the server by typing: `sudo service apache2 restart`

Figure 3.3: TLSAssistant v1.0 report (excerpt)

3.2.3 Attack Trees

We decided to model a series of attack trees in order to gain a better understanding of how each attack could affect the TLS security. An attack tree is a hierarchical organization of the software threats [Sch99]. Given a root node that identifies the objective to be attained, the tree is structured as follows: each node represents a sub-objective, and each leaf represents a different strategy for achieving the objective. Every node represents a single concept, and groups of nodes can be labeled with the condition of being satisfied simultaneously (AND nodes) or not (OR nodes). Attack trees can be expanded by introducing new edge types. One can model a tree based on various criteria (e.g., computational cost or time), each of which can provide unique insight into the difficulty of achieving a given objective.

The attack tree depicted in Figure 3.4 is obtained by applying Bruce Scheiner's 1999 original design to the TLS context. It has a limited expressive capacity because there is no way to distinguish between the leaves shown. There exist related works that apply the concept of attack trees in the context of TLS, articles like the one published in 2019 by Calzavara et al.[CFN$^+$19] which systematically evaluates the attack conditions on multiple TLS vulnerabilities. While the work in [CFN$^+$19] can benefit from the original design whereas, for our purposes, the expressive capacity of the original design prevents us from providing useful details regarding the actual application of the attack.

To increase TLS security awareness, we decided to extend TLSAssistant v1.0 adding a re-

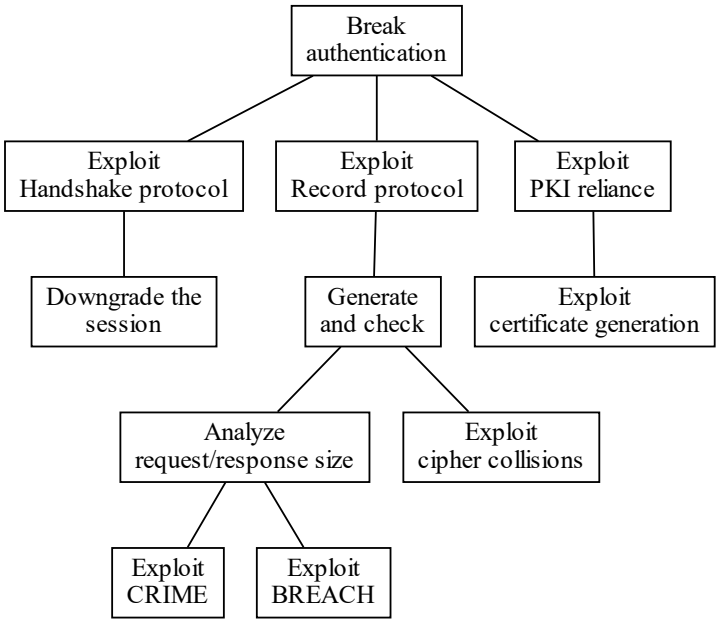


Figure 3.4: An example of an attack tree in the context of TLS

port option that can dynamically generate a forest of attack trees (namely **v3**, based on the verbosity scheme outlined in Section 3.2.2). TLSAssistant upgraded architecture can be seen in Figure 3.8. The graphical representations of the detected vulnerabilities can be utilized effectively in two situations: to teach and learn how a misconfiguration can impact a system (and its security properties) and to conduct a more accurate risk analysis by understanding how a particular attack can be mounted.

All the TLS attack trees consist of:

A goal (root) To model a tree capable of covering the entire TLS infrastructure, we decided to organize all attacks based on the security property they would violate. Consequently, our TLS attack trees have the following objectives: *(i)* Break Authentication, *(ii)* Break Confidentiality, *(iii)* Break Integrity;

Protocol/infrastructure subgoals Establish which protocols (e.g., Handshake Protocol), or infrastructure (e.g., *Public Key Infrastructure*) is being exploited to achieve the root goal. These nodes may also contain a reference to other attack trees. For example, due to the impact of the attacks, the *Break Integrity* tree can be seen as a *Break Authentication*'s subtree;

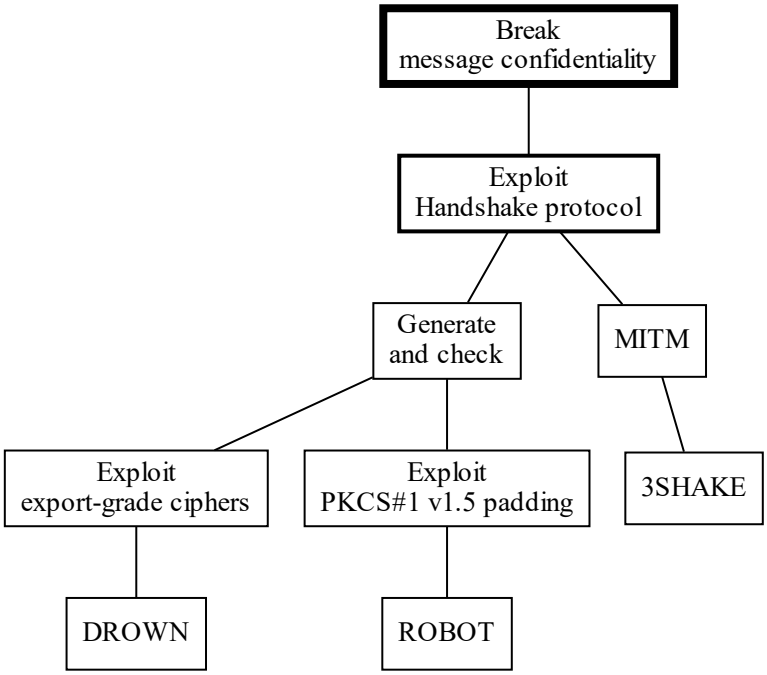


Figure 3.5: Attack tree depicting the goal of breaking TLS message confidentiality (simplified)

Technique subgoals Show the technique an attacker has to use in order to exploit the aforementioned protocol (e.g., a session downgrade attack is performed by exploiting the Handshake Protocol);

Attacks (leaves) The original leaf definition was not enough expressive for our purposes, for this reason we decided to extend them by adding some sort of "sub-leaves". Each leaf contains all the details needed to understand how the attack works divided into boxes. The first one lists the prerequisites an attacker needs, the second one describes the steps needed to exploit the vulnerability and, if needed, a third one shows how the attack is concluded. For example, an attacker who detects the presence of TLS compression may be able to launch a CRIME attack [NIS12]. By controlling the victim's browser via JavaScript, it is possible to continuously retrieve portions of the session cookie, thereby violating message confidentiality. The attacker is then able to impersonate the victim (breaking authentication) by using the same cookie on a different channel.

Figure 3.5, whose leaves can be seen in Appendix B.1, shows the tree structured on attacks that break message confidentiality. In particular, it shows three ways to exploit the Handshake pro-

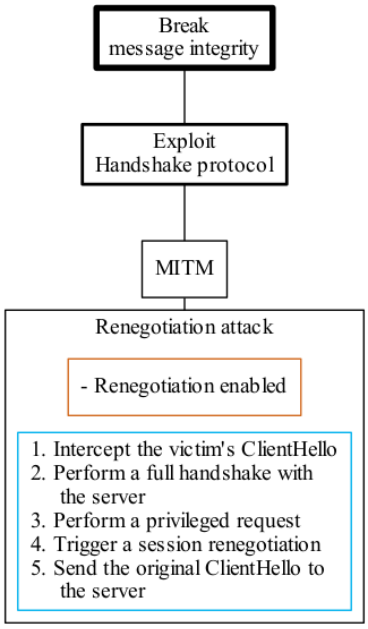


Figure 3.6: Attack tree depicting the goal of breaking TLS message integrity

tocol either by using a MITM approach or exploiting backward compatibility. Figure 3.6 shows the only attack that directly threatens message integrity while Figure 3.7 (whose leaves can be seen in Appendix B.2) contains the attacks that can break message authentication.[4] The dashed parts are references to the trees shown in the two previous figures. This graphical representation is used to indicate an intersection between the three modeled trees (except for the MITM branch within Figure 3.5).

The graphical representation of these trees can be generated by compiling two provided DOT files using a tool called `Graphviz` [Autb]. These two files, enriched with a set of hooks that allow TLSAssistant to highlight and edit them, compose its attack trees database.

3.3 Usability and Impact of Assisted Mitigations

To evaluate the usability of TLSAssistant when patching defective TLS configurations, we designed and performed a user study involving 62 participants.

[4]Some of the leaves may have an effect on more than one element of the CIA triad, but in order to avoid redundancy, they have been placed in a single subtree.

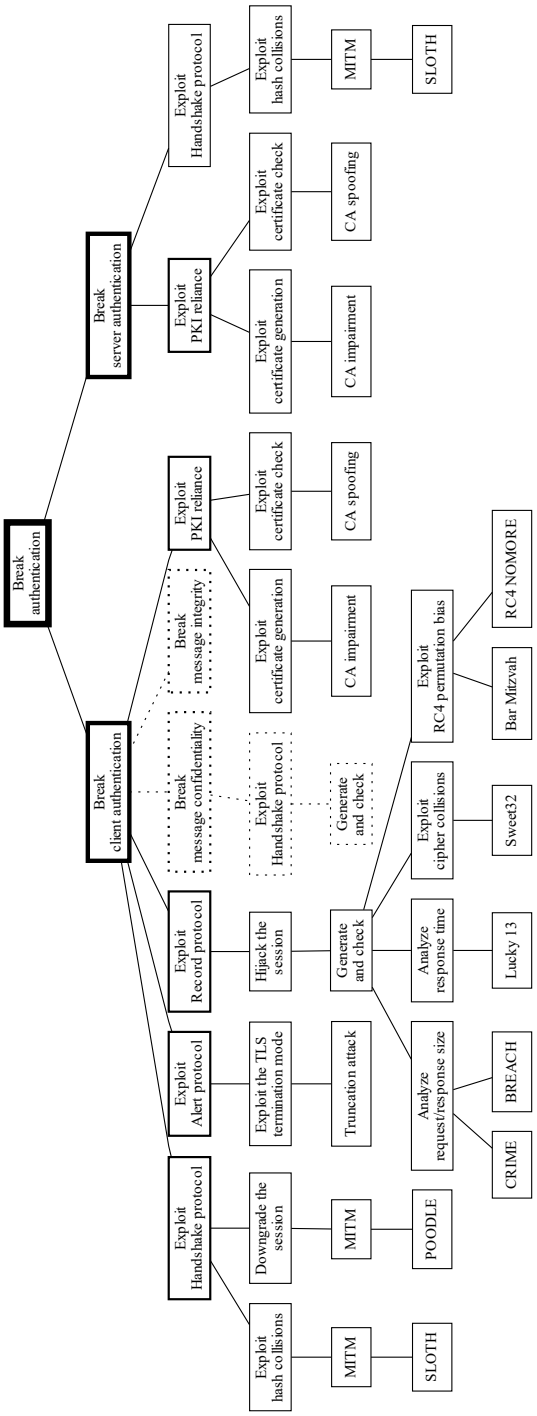


Figure 3.7: Attack tree depicting the goal of breaking TLS authentication (simplified)

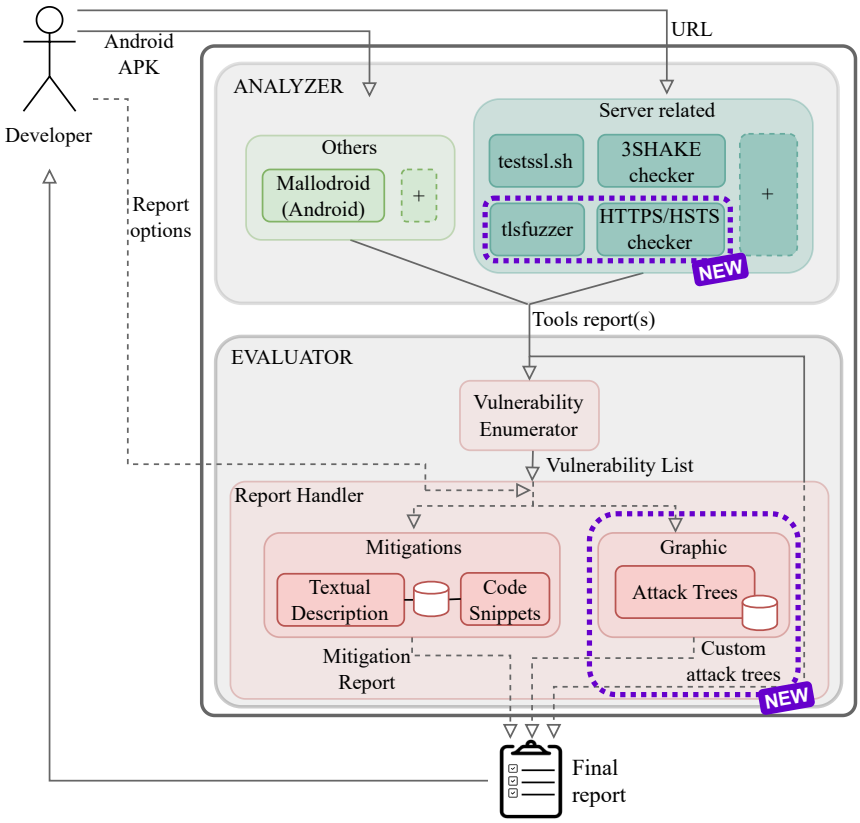


Figure 3.8: TLSAssistant v1.1 architecture

3.3.1 Related work: Usability Studies

3.3.1.1 Usability Studies in Cyber Security

Multiple usability studies have been conducted to evaluate tools and methodologies in various cyber-security domains, including password storage and penetration testing.

Naiakshina et al. conducted qualitative usability studies with either students [NDT+17] or remote freelance developers [NDG+19], requesting that they create a password storage mechanism. These studies demonstrated that participant knowledge of security does not guarantee the delivery of secure software.

In the domain of risk assessment, Allodi et al. measured the accuracy [ACMS20] and the difficulty [ABC+17] of students (with different technical education) in assessing the severity of software vulnerabilities using the Common Vulnerability Scoring System. Labunets et al. conducted a series of empirical evaluations to compare the efficacy of two classes of threat-analysis techniques [LMPT13] and the clarity of two risk model representations [LMP+17].

Scandariato et al. conducted a series of controlled experiments to compare static analysis and penetration testing tools based on how well they assist developers in accurately detecting vulnerabilities [SWJ13] and then fixing the code [CS16].

Compared to the described studies, our focus is different: we tried to understand how a system administrator could be guided toward a correct configuration of vulnerable webservers using actionable hints.

3.3.1.2 Impact of Providing Hint Suggestions

The following articles focus instead on how awareness and documentation affect the usage and related maintenance of specific technologies; they start from hypothesis similar to ours but focus on the difficulty rather than proposing a solution.

Acar et al. [ABF+16] conducted a systematic investigation into how the documentation available to developers directly affects both security and privacy properties. They discovered that the majority of developers use search engines and StackOverflow to address issues, resulting in poor implementation outcomes. Gorski et al. [GIW+18] evaluated the impact of providing security advice in the event of API misuse, demonstrating that the advices had a positive impact on code security and had no effect on the interface's overall usability.

Krombholz et al. investigated the mental models of both users and sysadmins; their findings reveal a large number of misconceptions regarding threat models, protocol components, and the benefits of using HTTPS [KBP+19]. In [KMSW17], they also conducted a series of controlled experiments to demonstrate the complexities of deploying HTTPS, demonstrating that it is far too complicated even for experts. Bernhard et al. have partially validated the findings of the latter as they conducted two usability studies focusing on the certificate acquisition [BSA+19].

The research conducted by Tiefenau et al. reveals that even skilled administrators struggle to keep their systems up-to-date because the decision-making process preceding the application of patches is time-consuming and requires careful consideration [THKvZ20b]. To overcome this limitation, Li et al. propose that assisting system administrators during information collection would simplify updating efforts and increase the likelihood of prioritizing updates for managed systems [LRM+19].

3.3.2 User Study Design

This study aims to examine the impact of providing system administrators with a set of mitigations in order to evaluate the assistance provided by actionable reports in patching a flawed TLS configuration. The *quality focus* regards how mitigation hints increase the developer capability to correctly and quickly patch a defective TLS configuration. We thus formulate the following

Table 3.5: Demographics: Participants' academic background

Course	✗	✓
Introduction to Computer and Network Security	12	50
Security Testing	28	34
Cryptography	41	21
Network Security	52	10
Cyber Security Risk Assessment	57	5
Offensive Security	58	4
Complexity, Crypto and Financial Technology	59	3

two research questions:

RQ1. Does a textual description of the mitigation and the corresponding code snippet *increase* the likelihood of a *correct* patch to a defective TLS configuration by a system administrator?

RQ2. Does a textual description of the mitigation and the corresponding code snippet *decrease* the time required by a system administrator to patch a defective TLS configuration?

The main perspective from which our experiment should be evaluated is how actionable information can improve the speed and accuracy of identifying and patching insecure TLS configurations. Other interesting perspectives include *(i)* a researcher interested in empirically evaluating the value of hints for patching defective TLS configurations; or *(ii)* a project manager, who has to make a decision of which development/maintenance tools and procedures to adopt, in order to ensure the successful deployment of a correctly configured infrastructure.

The experimental settings have been designed following the template and guidelines by Wohlin et al. [Car01] to select participants and present their demographics (Section 3.3.2.1), to define the experimental design and select appropriate metrics and dependent/independent variables (Section 3.3.2.2), to identify the most appropriate statistical tests (Section 3.3.2.3) and to identify the threats to the validity of our findings (Section 3.3.2.4).

3.3.2.1 Demographic Statistical Sample

We involved 62 participants in this study. They are Bachelor and Master students from the departments of Computer Science and Mathematics of the University of Trento playing the role of inexperienced system administrators who should patch defective TLS configuration files.

The study has been conducted as part of laboratory lectures in two courses of cybersecurity offered in at the University of Trento.

Table 3.6: Demographics: Participants' technical background

Technical skill	✗	✓
Configure TLS servers	47	15
Configure Apache HTTP instances	29	33
Create/edit UNIX configuration files	26	36
Change working folder	1	61
Create/remove folder	2	60
Edit file	3	59
Install package	5	57

Participants were aware that they could abandon the experiment at any time without consequences, as their performance would have not been evaluated. There was no compensation for their participation in the study, neither monetary nor an exam bonus.

A profiling survey has been used to collect demographic data from the participants.

Seniority. The first question splits participants according to their seniority: 22 participants are Bachelor and 40 are Master students.

Year. 11 participants attend the 2nd and 11 the 3rd year of the Bachelor program, while 26 participants attend the 1st year and 14 the 2nd year of the Master program.

Academic background. We filled a list of the related University courses, whose content might have been relevant to influence the result of a corrective task on TLS configuration. The answers are shown in Table 3.5. Participants background was collected in terms of which related courses they already attended or not (column marked with ✓ and ✗, respectively, in the table). Most of the participants (i.e., 50 over 62) already attended the course *Introduction to Computer and Network Security*, while almost half of the participants (i.e., 34 out of 62) attended the course about *Security Testing*. Less participants attended *Cryptography* (21 students) and *Network Security* (10 students). A smaller group attended *Complexity, Crypto and Financial Technology* (3 participants), *Cyber-Security Risk Assessment* (5 participants) and *Offensive Security* (4 participants).

Technical background. Additionally, we collected the technical background of participants, in terms of which tasks they conducted in the past (data shown in Table 3.6). Only 15 are expert in manually configuring a TLS server because they already did it (column marked with ✓), while half of them already configured Apache HTTP servers (33 participants) and created or edited other Unix configuration files (36 participants). The large majority of the participants are fluent in basic Unix administration tasks, such as navigating in the file system (61 participants), working with folders (60 participants), editing files (59 participants) and installing system packages (57 participants).

Number of participants. Establishing the right number of participants to a user study is a

difficult task that can be completed only ex-post, after the experimental data are collected (by estimating the *power* of the test, as done in [CMM+15]). However, a replication with a large number of participants is mandatory for inconclusive studies, where the observed difference in independent variables was not statistically relevant (*power* was low). As a matter of fact, between 15 and 20 participants is considered reasonable to draw conclusions from statistical analyses of the results [SM16]; in our case we have 4 times this number of participants.

Concerning the profiles of participants, we are aware that the expertise of students may be different from that of professionals. However, finding professionals available to conduct a demanding experiment as the one we designed is not easy. We mitigated this limitation by considering students with different levels of education (Bachelor and Master) and by making sure that participants had enough knowledge on TLS and its related vulnerabilities. All in all, the use of undergraduate students as a proxy of junior developers to draw conclusions is a common practice in empirical software engineering that is largely accepted and validated [HRW00, SAW08, SMJ15].

Ethical considerations. Participation was voluntary and students could have chosen not to attend the experiment without negative results on the final evaluation of the exam. Those students who opted to participate were aware that they would not be evaluated based on their performance and that they would receive no compensation (neither money nor bonus in the exam mark) for the participation in the study; they were aware that they could drop off at any time with no consequence. During all the experimentation, we strive to adhere to the general ethical principles stated in the ACM code of conduct [Ass18] with particular attention to trustworthiness, fairness, privacy, and confidentiality.

3.3.2.2 Experimental Setup and Execution

Systems. The systems used to conduct the experiment are two web servers with faulty TLS configurations, running Apache HTTP Server v2.4.37 and OpenSSL v1.0.2. Each incorrect configuration exposes the corresponding system to one specific attack. The two systems are:

- S_1: a defective webserver vulnerable to BREACH; and
- S_2: a defective webserver vulnerable to CRIME.

They are packaged as two distinct VirtualBox machines. Instead of using a random pair of detectable misconfigurations (e.g., POODLE [MDK], Sweet32 [BL16] or others) we selected two vulnerabilities that are comparable in terms of complexity of the operations required to patch. In addition, we ensured that they could be fixed in two hours, taking into account the student's technical background. In particular, both are prone to the same type of information leakage caused by DEFLATE, but exploited using two different attacks (as discussed in Section 2.3). In addition, it is important to note that these systems are representative of realistic TLS configurations

as both Apache and OpenSSL are respectively the most popular webserver [Dat22] and TLS library [Dat23]. To make the corrective tasks independent, only one vulnerability is present in each system.

Metrics. To measure the support of mitigation actions to conduct a corrective maintenance on TLS configurations (i.e., vulnerability detection and fix), we identified the following variables. The main factor of the experiment—that acts as an independent variable—is the presence of the mitigation hints during the execution of the task. In our experiment, the *base* treatment case TR_{list} consists of the bare list of vulnerabilities, as it is provided by the analysis tool testssl.sh; and TR_{hint} consists of the actionable reports generated by TLSAssistant, that include not only the list of vulnerabilities, but also a textual description of the mitigations and a code snippet to apply the mitigation.

Moreover, by adopting an approach similar to the one described in the ISO 9241 (Part 11) standard [ISO18], we instrumented the experimental settings to measure the following dependent metrics:

- **Correctness** of each corrective task performed by participants, which corresponds to the *System Effectiveness* in [ISO18] and thus examines the participants' ability to complete a task. Participants could repeat the scans as many times as they like during the experimental session. However, to consider a task correct, the participants were supposed to run a final scan and to show the experimenter that the freshly generated report contains no vulnerability.
- **Time** taken to perform a corrective task on a defective TLS configuration, which corresponds to the *System Efficiency* in [ISO18]. We collected such information by asking participants to fill in—while performing the experimental tasks—start and end time of each task.

Finally, the *System Satisfaction* in [ISO18] to evaluate the overall usability of the TLS analyzer report is measured by a survey questionnaire (available in [MCSR21b] and analyzed in Section 3.3.3.3).

Experimental Design. We employ a counterbalanced experimental design intended to fit into two 40-minute lab sessions. Each participant is randomly assigned to one of four groups based on his or her seniority, with each group working in two laboratories on different systems with different treatments. The design allows for considering different combinations of *Systems* and *Treatments* in different order across *Labs* (see Table 3.7).

Experimental Procedure. Before the experiment, participants were properly trained with lectures and exercises on TLS, to recall the required background [MCSR21b]. The purpose of training is to make participants confident about the kind of tasks they are going to perform and the environment they will have available.

Table 3.7: Experimental design

	Group A	**Group B**	**Group C**	**Group D**
Lab 1	S_1 + TR$_{hint}$	S_2 + TR$_{list}$	S_2 + TR$_{hint}$	S_1 + TR$_{list}$
Lab 2	S_2 + TR$_{list}$	S_1 + TR$_{hint}$	S_1 + TR$_{list}$	S_2 + TR$_{hint}$

The experimental setting has been designed to be as realistic as possible. Participants could therefore run scanning tools as frequently as desired, inspect scan reports, and search the Internet for additional information. Furthermore, by conducting a dry run, we ensured that the entire experiment could be completed in two hours. The dry run also helped in refining the survey questionnaires. Participants have been delivered the following material:

- two virtual machines: S_1 and S_2;
- two digital documents containing the instructions for each treatment (i.e., TR$_{list}$ and TR$_{hint}$);
- a printed page containing a recap of the lessons learned during the training phase (e.g., the commands used).

The experiment was carried out according to the following procedure. Participants had to:

1. Complete a pre-experiment profiling survey questionnaire;
2. For Lab 1: *(i)* mark the start time; *(ii)* perform the corrective task; *(iii)* mark the stop time;
3. Complete a survey questionnaire on the first lab;
4. For Lab 2: *(i)* mark the start time; *(ii)* perform the corrective task; and *(iii)* mark the stop time;
5. Complete a survey questionnaire divided in three parts: a *1st part* on the second lab, a *2nd part* on a comparison between the two labs, and a *3rd part* to collect feedback on TR$_{hint}$ (the treatment with mitigation hints).

The pre-experiment profiling survey collects demographic data about the participants, such as their previous experience with Apache HTTP Server and their knowledge of the Bash command language; the complete survey is included in the replication package available online [MCSR21b] and we have described the collected data in Section 3.3.2.1.

Each lab can be considered over, and, thus, a participant can mark the stop time, only after proving that the task was successfully completed or because the available time has expired.

We provide the list of questions for the survey questionnaire in [MCSR21b], report them in Appendix A and discuss the answers in Section 3.3.3.3. The survey questionnaires deal with cognitive effects of the treatments on the behavior of the participants and perceived usefulness of the provided report.

3.3.2.3 Statistical Tests

We are interested in determining if the presence of mitigation hints influences the *Correctness* and *Time* required to fix defective TLS configurations. However, observed differences in the correctness of corrective tasks and the time spent on them could be due to random variation or measurement errors. To test if the observed difference is statistically significant, we use sound statistical tests. As a common practice, we accept a 5% probability of committing type-I error, i.e., assessing that the difference is significant when it is actually due to random error. Practically, this setting defines the threshold $\alpha = 0.05$, for considering the result of a statistical test significant.

The lack of statistical significance may indicate that there was an effect, but it was not observable (possibly due to a small number of observations), rather than that there is no effect, i.e., we risk to commit a type-II error (nullification fallacy [KFLS15]). To quantify the probability of this problem, when the significance threshold is not reached, we can estimate the probability π of committing a type-II error as *1 - Power*, where *Power* is the statistical power of the adopted statistical test. As common practice, assume a threshold $\beta = 0.20$ and we consider the power adequate when $\pi < \beta$.

The decision of which statistical tests to use was based on test applicability conditions and best practices recommended or commonly accepted in authoritative literature.

There are two distinct data points for each participant, one for the first lab and another for the second lab, so we never conduct multiple pairwise comparisons with overlapping data. Consequently, there is no risk of inflating the family-wise error rate, and no correction factor (such as Bonferroni or Holm) is needed.

Correctness. To analyze the differences in terms of *Correctness*, we looked at the frequencies of correct/wrong tasks and we used a test on categorical data, because the tasks can be either correct (completed successfully) or incorrect (completed unsuccessfully). In particular, we used Fisher's exact test [Dev07] that is applicable to categorical data (correct/wrong answers). Fisher's exact test is more accurate than the χ^2 test for small sample sizes, which is another possible alternative to test the presence of differences in categorical data. The same analysis was conducted in [CDPF+14].

Time. To test the differences in *Time*, we perform the two-tailed Mann-Whitney U test on all samples [She07]. This test is applicable to compare (time duration) samples of two populations. As a non-parametric test, Mann-Whitney U test does not require data to be normally distributed.

Effect size.To quantify the magnitude of differences among the two treatments, we used two kinds of effect size measures, the *odds ratio* for the categorical variable *Correctness* and the Cliff's delta effect size [GK05] for *Time*. An odds ratio of 1 indicates that the condition or event under study is equally likely in both groups (participants using TR_{list} and those using TR_{hint}). An odds ratio greater than 1 indicates that the condition or event is more likely in the first group. An odds ratio less than 1 indicates that the condition or event is less likely in the first group. For independent samples, Cliff's delta provides an indication of the extent to which two (ordered) data sets overlap, i.e., it is based on the same principles of the Mann-Whitney test. Cliff's Delta ranges in the interval $[-1, 1]$. It is equal to $+1$ when all values of one group are higher than the values of the other group and -1 when the opposite is true. Two overlapping distributions would have a Cliff's Delta equal to zero. The effect size is considered small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$ and large for $d \geq 0.474$ [Coh88].

Co-factors. The analysis of other factors (participants' background, the system, the lab) that could have influenced the *Correctness* and *Time* is performed using the *Generalized Linear Mixed Model* [Jia07] (GLMM for short). GLMM extends the Generalized Linear Model (GLM) by adding random effects to the linear predictor (GLM only supports fixed effects). Random effects are particularly appropriate with repeated measures design, i.e., when different data points are collected for the same participant (in our design, each participant worked at two tasks). GLMM incorporates a number of different statistical models: *ANOVA*, *ANCOVA*, *MANOVA*, *MANCOVA*, ordinary linear regression, t-test and F-test. It consists in fitting a linear model of the *dependent* output variables (*Correctness* or *Time*) as a function of the *independent* input variables (all factors, including the treatment, i.e., the vulnerability detection tool). GLMM is capable of testing a dependent output variable (experiment outcome) on many input variables (factors) and it allows to test the statistical significance of the influence of each factor separately.

GLMM requires to specify the *exponential-family* distribution based on the domain of the outcome. We have chosen:

- the `binomial` family with `logit` link function to fit the *Correctness*, as it corresponds to a logistic regression that is appropriate for a binary outcome (correct/wrong task);
- a `Gamma` family for fitting the *Time*, as it is appropriate for fitting a time duration that can be a positive decimal value.

As other models could have been used to fit the *Time*, we use the *Akaike Information Criterion* (AIC for short) to check that the chosen model (i.e., the `Gamma` exponential-family distribution) was the most appropriate to fit our data [SKvW+14]. AIC is founded on information theory and it entails balancing the trade-off between the goodness of fit of the model and the size of the model.

Surveys. Two statistical tests have been used on the survey questionnaire. Fisher's exact test is applied to categorical data in order to compare the frequency of yes/no responses among

participants who utilized various tools. The Mann-Whitney U test is used to analyze answers to general questions (not specific to a lab) that were formulated using a Likert scale, checking for the null-hypothesis that the average answer was negative or neutral.

3.3.2.4 Threats to Validity

The main threats to the validity of this experiment belong to the internal, construct, conclusion and external validity threat categories [Car01].

Internal validity threats concern external factors that may affect the independent variable. The chosen design allowed us to control a number of factors, namely participants background, system and learning across experimental sessions. Participants were not aware of the experimental hypotheses, not rewarded for the participation in the experiment and not evaluated on their performance in doing the experiment.

Construct validity threats concern the relationship between theory and observation. As described in Section 3.3.2.2, we considered real vulnerabilities and we used a sound procedure to objectively evaluate whether the fixes were correct.

The background of participants was estimated according to their academic background and their technical knowledge.

Conclusion validity threats concern the relationship between treatment and outcome. We used statistical tests to draw our conclusions on the correctness and the time required to fix TLS misconfigurations. The adopted statistical tests are particularly robust (i.e., they do not give false rejections of the null hypothesis) under deviations from normality.

External validity concerns the generalization of the findings. In our experiments we considered two major attacks related to a TLS configuration, namely BREACH and CRIME. Although different attacks might occur, the results obtained with these already support well our interpretations.

Our experiment exploited one real-world web application running in a web server (i.e., Apache HTTP). Despite we consider that this web server is representative of other web servers (e.g., NGINX), in principle different results could be obtained for different web servers.

The study was performed in an academic environment, which may differ substantially from an industrial setup. However, we mitigate this threat by using subjects with different background and different seniority, including Bachelor and Master students, some of which with experience with TLS, Apache HTTP and Unix. Moreover, we considered their seniority as a factor to detect any influence on the results.

Table 3.8: Correctness in fixing an incorrect TLS configuration

	TR_{list}	TR_{hint}
Correct	40 (67%)	61 (98%)
Wrong	20 (33%)	1 (2%)

Table 3.9: Analysis of correctness (GLMM)

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	15.26	14.49	1.05	0.2923
Treatment	-14.12	4.28	-3.30	**0.0010**
System	-0.72	2.34	-0.31	0.7596
Lab	0.43	2.34	0.18	0.8534
Seniority	-6.07	13.78	-0.44	0.6598
Year	3.66	7.07	0.52	0.6048

3.3.3 User Study Results

This section presents the data collected during the experimental validation. After analyzing data with reliable statistical methods, we formulate answers to the two research questions presented in Section 3.3.2.

3.3.3.1 Analysis of Correctness

Table 3.8 shows the distributions of correct/wrong answers when using testssl.sh's reports (TR_{list}) and the actionable reports (TR_{hint}).

When provided with mitigations and code snippets, almost all participants were able to correctly patch the vulnerability; only one participant was unsuccessful. Conversely, when participants were provided only with the list of vulnerabilities,[5] just 40 were able to complete a correct vulnerability patch.

We apply Fisher's test to check if the observed trend is statistically significant. The difference is statistically relevant (p-value< 0.001, and we recall that we assume significance when p-value$< \alpha$, with $\alpha = 0.05$) with a *large* effect size (odds ratio = 30).

Table 3.9 reports the analysis of correctness with GLMM. The model takes into account not only the effect of the main treatment (i.e., availability of mitigation hints) but all the other factors that

[5]The number of participants who worked with TR_{list} does not sum to 62, because two participants attended only the first lab.

we considered in our experimental design, i.e., the *System*, the *Lab*, the profile of the participants (the *Year* attended and their *Seniority*). The random-effects term is the participant who took parts in the two labs.

Statistically significant cases are in boldface. Consistently with the Fisher's test, we can observe that the availability of mitigation hints significantly influences the correctness of a task related to fixing a TLS configuration file, as Pr($>|z|$)$= 0.0010$. However, it is the only significant factor.

The probability of committing type-II error obtained from GLMM power analysis is 0.05, which is smaller than 0.20 So, we can claim that the missing significance is not due to insufficient experimental data points, but to missing causal correlation between independent and dependent variables.

System has no significant effect on the *Correctness* of tasks, so we can conclude that the two applications and the two corrective tasks were well-balanced, with neither being more difficult nor simpler to fix. Moreover, the *Lab* is not a significant factor, so there is no discernible learning effect between the two experimental sessions. This means that performing a first lab does not improve the accuracy in the second lab, and we only measure the actual difference caused by the independent variable (i.e., the presence of mitigation hints).

Based on these results, we can answer the research question RQ1 (see beginning of Section 3.3.2) as follows:

> *Providing a text description together with a code snippet of the mitigation increases the capability of a system administrator to patch the defect in the TLS configuration. In fact, we observed that participants deliver correct fixes in 98% of the cases when this additional information has been included in the security reports, while the rate of correct fixes drops to 67% when no mitigation is provided.*

3.3.3.2 Analysis of Time

We now analyze the time taken to fix a TLS configuration. We only consider time information for those participants who correctly fixed TLS configurations, and we discard data for incomplete and wrong tasks.

Figure 3.9 shows two box-plots of the time (in minutes) taken to fix a TLS configuration. Descriptive statistics (number of data points, mean, median and standard deviation) are summarized in Table 3.10. On average, when testssl.sh is used (TR_{list}) fixing a wrong configuration takes 23 minutes. When using the actionable reports (TR_{hint}) the amount of time, on average, is reduced to less than 8 minutes.

According to the result of the Mann-Whitney test, this difference is statistically significant (p-value< 0.001) with a *large* effect size (Cliff's Delta = 0.8819).

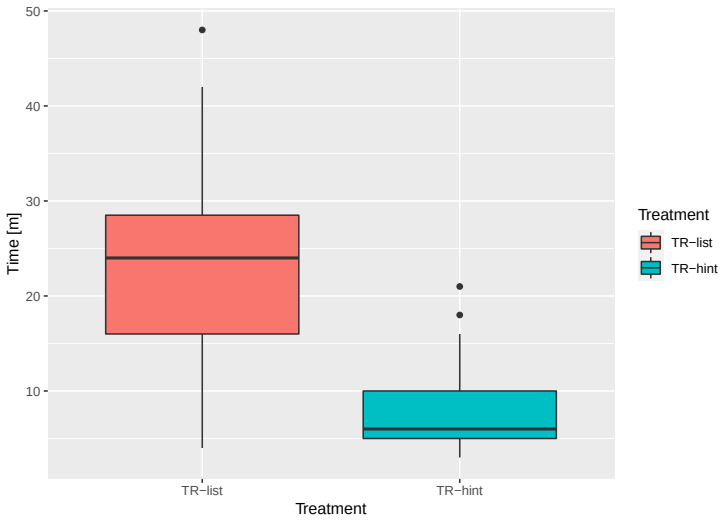


Figure 3.9: Time (in minutes) to fix a security issue

Table 3.10: Time (in minutes) to fix a security issue

	TR_{list}	TR_{hint}
n	40	61
Mean	23.2	7.9
Median	24	6
SD	9.51	3.66

Table 3.11 reports the analysis of *Time* with GLMM, the statistically significant cases are shown in boldface. Consistently with the results of the Mann-Whitney test, GLMM confirms that the availability of mitigation hints significantly influences the time needed to fixing a wrong TLS configuration file. Similarly to what previously observed for the *Correctness*, other factors have no significant influence on the *Time* to fix.

In this case, however, the probability of a type-II error obtained from GLMM power analysis is larger than 0.20. So, differently than the analysis of *Correctness*, the missing significance of cofactors influence (e.g., *System* and *Lab*) on *Time* cannot be interpreted as considerations on the experimental design.

Eventually, considering that this GLMM model could have been computed with different exponential family distributions, we need to check that our choice was the most appropriate to fit our data. To this aim, we used the Akaike Information Criterion (AIC). This consists in fitting our data with other models and compare their AIC value. AIC values for other models are the following: 681.33 for Gaussian, 614.64 for Gamma and 631.43 for Poisson. As we can see, our

Table 3.11: Analysis of time (GLMM)

	Estimate	Std. Error	t value	Pr(>|z|)
(Intercept)	0.04	0.03	1.39	0.1637
Treatment	0.09	0.01	12.55	<**0.0001**
System	-0.01	0.01	-1.57	0.1176
Lab	0.01	0.01	1.02	0.3077
Seniority	-0.02	0.02	-0.79	0.4274
Year	0.00	0.01	0.39	0.6954

Table 3.12: Analysis of survey questionnaire (Fisher's test)

Question	TR_{list}		TR_{hint}		P-value
	Yes	No	Yes	No	
Enough time	40	20	61	1	<**0.0001**
No difficulty	18	42	57	5	<**0.0001**
Online search	53	7	3	59	<**0.0001**
	<60%	≥60%	<60%	≥60%	
Configuration	53	7	41	21	**0.0048**
Documentation	24	36	61	1	<**0.0001**

choice is confirmed, because the AIC value for the *Gamma* family is much lower than those of the other models.

Providing a text description together with a code snippet of the mitigation decreases the time needed by a system administrator to patch the defect in the TLS configuration. In fact, we observed that in average it took 8 minutes to fix a misconfiguration when this additional information has been included in the security reports, while on average it took 23 minutes when no mitigation is provided.

3.3.3.3 Analysis of Survey Questionnaire

The survey questionnaire (available in [MCSR21b]) is composed of three parts and is meant to collect the participants opinion on the experiment.

3.3.3.3.1 First Part The objective of the first section is to collect participant feedback on the security reports in order to compare them indirectly. Answers to this first part are reported

Table 3.13: Analysis of survey questionnaire

Question	TR_{list}	TR_{hint}
Most useful	18	41
Most easy to read	7	52
Most complex to understand	53	6

in Table 3.12. For the first three questions, the table reports the number of yes/no answers for participants who worked just with the list of vulnerabilities (2nd and 3rd columns, respectively) and with the mitigation actions within the actionable reports (4th and 5th columns).

The fourth and fifth questions asked participants to report the percentage of their lab time spent on specific tasks. Answers are in Likert scale ("$<$ 20%", "$\geq$ 20% and $<$ 40%", "$\geq$ 40% and $<$ 60%", "$\geq$ 60% and $<$ 80%" and "$\geq$ 80%"). The table reports the number of participant who answered when a task took less than 60% or more than 60% of the lab time. The last column of Table 3.12 reports the significance (i.e., p-value) computed by applying the Fisher's test to each question, to reveal statistical significance for the various answers. Significant cases are highlighted in boldface.

According to the first question, participants working with the actionable reports (i.e., TR_{hint}) considered that the time allocated to the task was enough, while time was short for participants assigned TR_{list}. This result is consistent with the analysis of *Time*, of Section 3.3.3.2, where participants who worked with no mitigation hints took longer to complete their tasks.

Consistently, the responses to the second question reveal that only participants who worked with actionable reports had no trouble completing the tasks. The difficulty of completing tasks increased when participants were only provided with a list of security flaws.

Considering the responses to the third question, we see that the majority of participants who worked solely with the list of vulnerabilities conducted online searches (53 positive answers versus 7 negative answers). In contrast, when the actionable reports were utilized, the majority of participants did not conduct online searches (3 positive answers versus 59).

Moving on to the next two time-related questions, we observe a different strategy for completing the assigned task. 53 participants assigned to TR_{list} spent less than 60% percent of the lab time examining TLS configuration code, indicating that they did not attempt to comprehend how TLS was configured and instead focused on looking for the solution online or in the TLS report. In contrast, almost none of the participants who utilized the actionable reports searched online for TLS documentation.

3.3.3.3.2 Second Part The second section of the survey is a more direct comparison between the two alternative approaches. In fact, we asked participants to make an explicit decision about

Table 3.14: Analysis of survey questionnaire (Mann-Whitney test)

Question	Strongly disagree	Disagree	Neutral	Agree	Strongly agree	P-value
Mitigations hints useful	0	3	15	21	20	<**0.0001**
Code snippets useful	1	1	10	15	32	<**0.0001**

the two reports. For each question, Table 3.13 reports the number [6] of decisions that participants formulated about tasks supported by TR_{hint} and TR_{list}.

The first question asked which report was the most useful when correcting the configuration error. The majority of respondents (41) found the actionable reports to be the most beneficial.

The second question investigated the readability of security reports. The reports with mitigations were considered easier to read than the bare list of vulnerabilities.

The third question dealt with complexity of understanding. Consistently with the previous answers, the report not containing mitigations (i.e., TR_{list}) was considered more complex to understand than when mitigations were included (in TR_{hint}).

Table 3.14 reports the answers to two other direct questions about mitigation hints and code snippets. Many participants strongly agree (20) or agree (21) that the textual mitigation hints were useful to complete the corrective task. A similar trend can be observed for the next question, participants strongly agree (32) or agree (15) that code snippet were useful to complete the corrective task. The result of Mann-Whitney test (null-hypothesis mean answer $\leq$ *"Neutral"*) confirms that this trend is statistical significant.

3.3.3.3.3 Third Part This last section, with only open questions, let participants write free text as feedback to the experiment. Its analysis would require a fundamentally different approach, mostly bases on *grounded theory* [GS67, SC90] and is thus left for future work.

3.4 Lessons Learned and Discussion

We observed that mitigation hints help to patch defects in TLS configurations, by reducing the probability of error by 30 times and the time to complete the fix by 3 times.

In the following, we report the implications and general observations that we can formulate, based on the objective and quantitative results presented in the previous section.

[6] We consider only 59 participants as two did not attend the second lab and a third one did not answer the 2nd part of the survey questionnaire.

Limited information: *Automated security tools convey insufficient information.* A TLS configuration is quite complex as it contains many properties that may not be immediately understandable; therefore, a simple list of security issues is insufficient to allow a system administrator to fix it. Indeed, additional information is required to fill in the gaps of a security report and assist the system administrator in determining the necessary changes. In our experiment, the participants who received the list of vulnerabilities had to search online to understand how to fix security defects, while this was not required to those who worked with actionable hints (see Section 3.3.3.3 and Table 3.12).

Correctness and Time: *Actionable maintenance hints improve correctness and time to fix.* Despite the fact that multiple tools identify the same vulnerabilities, it is crucial how these are communicated to system administrators. When actionable hints are available, a security report is more usable, user-friendly and able to guide system administrators towards a mitigation (see Table 3.9) in a timely manner (see Table 3.11 and Figure 3.9). Researchers and practitioners should keep this result in mind when developing new automated security tools. Scan results should be complemented with explanations and operational suggestions on how to solve the security problem or, at least, where to find additional information for guidance towards the solution.

Perceived effect: *Mitigations hints are easier to read and more useful than the list of vulnerabilities*. When performing corrective maintenance, a flat list of detected vulnerabilities is not regarded as particularly useful, likely because it is neither informative nor simple to read. This leaves system administrators with no idea where to look for the required information or how to differentiate between relevant and irrelevant data gleaned from, for example, online searches. Consequently, additional information must be gathered by spending time on the code or by reading additional documentation. Conversely, system administrators are aware of the benefits of actionable mitigation hints because they are considered easier to read, more useful to support a fixing task and do not require additional time to fill knowledge gaps (see Survey Questionnaire [MCSR21b] and Table 3.13).

Chapter 4

TLS Vulnerabilities and Threat Intelligence

In this chapter, we present the integration of TLSAssistant [Sec] in the FINSEC platform [FIN], which is a framework for predictive and collaborative security of financial infrastructures. The goal of the integration was to potentially improve the functionality of the other integrated services by combining multiple sources of security intelligence and providing vulnerability evaluation and scoring. The following sections will *(i)* describe the context in which this integration took place, *(ii)* provide an overview of the FINSEC platform by describing its structure, how it works and the parties involved, *(iii)* illustrate how we overcame the challenges posed from the integration, *(iv)* further detail the information exchange with the Risk Assessment Engine as implemented in FINSEC and *(v)* discuss the lessons learned.

4.1 Context and Motivation

Under the revised *Payment Services Directive* (PSD2) [Eur], *Account Servicing Payment Service Providers* (ASPSP) are required to provide an interface for third parties to access account information and perform operations (e.g., payments) on behalf of the account holder.

The "Access to Account (XS2A) Framework" [Ber] provides a detailed description of RESTful *Application Programming Interfaces* (APIs) and their usage for authentication of involved parties and authorization to access service resources, including account information, payment initiation, and confirmation of funds. Both the transport and application layers contribute to the APIs' security. The first core technology identified explicitly by the guidelines is the TLS protocol: specifically, "the communication between the Third Party Provider (TPP) and the ASPSP is always secured by using a TLS-connection using TLS version 1.2 or higher." [Ber]. Recommen-

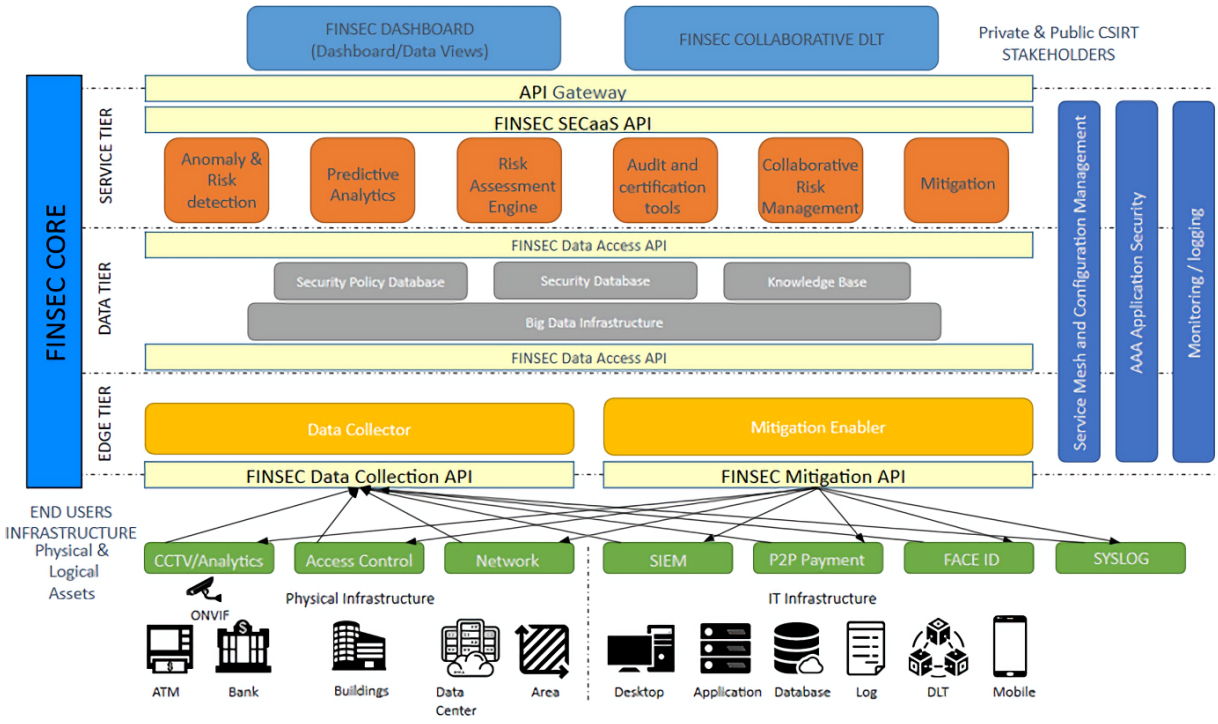


Figure 4.1: FINSEC reference architecture

dations for security and identification standards in XS2A have been published by Open Banking Europe [PRE], which explicitly assume the use of TLS to ensure confidentiality and integrity. The use of TLS is also assumed by the Financial-grade API (FAPI) specification [Ope] for authentication and authorization. While PSD2 APIs are one of the most recent examples of online financial services that rely on the security of TLS, there are several other examples such as home banking web and mobile applications; is also explicitly cited as an example of how to comply with the *Payment Card Industry Data Security Standard* (PCI-DSS) [PCI22] Requirement 4.1, to "Use strong cryptography and security protocols to safeguard sensitive cardholder data during transmission over open, public networks".

Beyond online services, TLS is deployed in a wide range of IoT devices [SM17] and client-end TLS proxies (e.g., in anti-virus products) [WMY18]. Vulnerabilities in their TLS configuration and implementation can significantly downgrade the overall cyber and physical security, affecting many different enterprise sectors. For instance, in the financial sector, CCTVs are crucial for ATM surveillance.

Figure 4.2: FINSEC *Dashboard*

4.2 FINSEC Project

FINSEC (Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures) is a Horizon 2020-funded project developed by a consortium of 23 international partners [FIN]; its objective is to provide an integrated service platform for the cyber-physical security of critical financial infrastructures.

Figure 4.1 shows a version of the FINSEC architecture [FIN19a] updated to October 2019. It can be seen as a three-layered set of components with different roles:

Presentation Layer composed by a *Dashboard* and any *External Services* with access to the APIs. In particular, the *Dashboard* is an interactive interface developed within the FINSEC project capable of showing the overall security status of the infrastructure and its assets (see Figure 4.2);

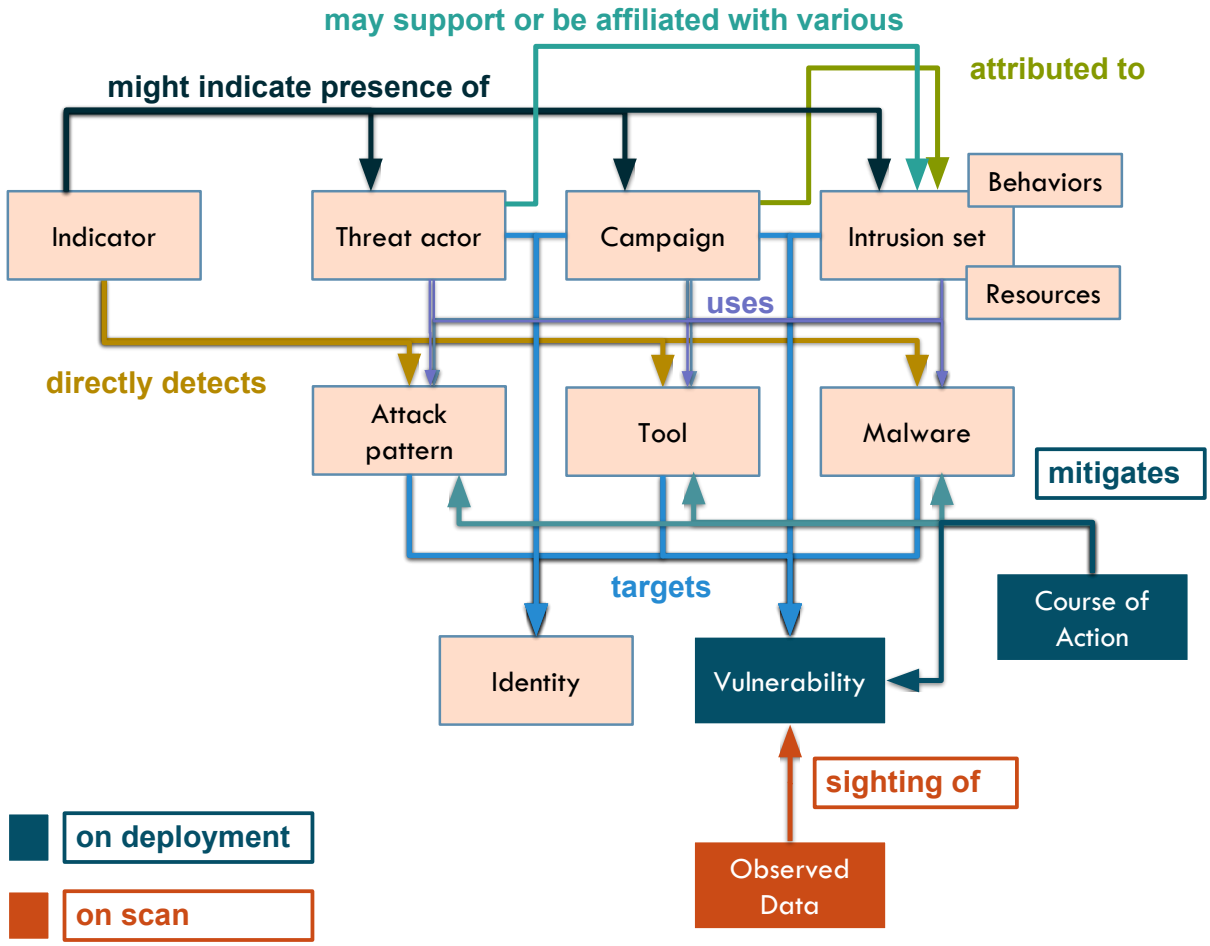


Figure 4.3: A simplified STIX SDO ecosystem with its possible relationships

Service Layer contains the services integrated in the FINSEC platform. These can either be called on-demand or have an ongoing monitoring activity (e.g., the Risk Assessment Engine);

Data Layer hosts a collection of security policies, vulnerabilities (i.e. imported CVE), system logs, and additional intelligence. It is where all knowledge is stored and retrieved by entities in higher layers. Its content is written in FINSTIX, a proprietary extension of the STIX language (see [FIN19b]). STIX (Structured Threat Information eXpression) is a standard language, created by the OASIS Cyber Threat Intelligence Technical Committee to enable organizations to share threat intelligence data in a machine-readable and consistent format [OAS23]. Each data can be represented with a combination of objects and their descriptive relationships. The *STIX Domain Objects* (SDOs) and *STIX Relationship Objects*

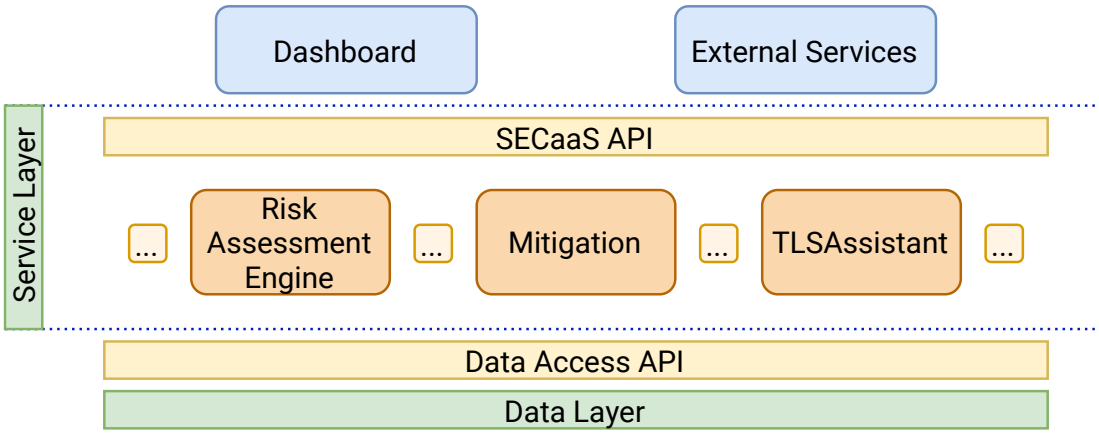


Figure 4.4: A simplified FINSEC architecture

(SROs) are visually summarized in Figure 4.3.

From a developer's perspective, the FINSEC platform has a microservice architecture. This means that each integrated service is fully virtualized, independent of the others, and must manage its own dependencies and deployment. In FINSEC, this has been accomplished by leveraging the containerization technology provided by Docker. Due to the high level of independence, this method aims to simplify both the overall management and the work of each individual maintainer, provided they handle their containers as if they were partially cloud-deployed. This is because, despite the fact that each service can maintain its own private state, it must store all pertinent data in the shared database and manage its own access queue.

The communication among containers is managed through the use of REST APIs that each maintainer must provide. The set of APIs exposed by the services is called *SECaaS API* while the one provided to access the *Data Layer* is called *Data Access API*.

4.3 Planning and Integration

TLSAssistant was developed outside of the FINSEC context, with design decisions driven by the use case of a system administrator who wants to check his own TLS deployment by downloading TLSAssistant and using it offline. Unlike the FINSEC platform, this specific use case resulted in *(i)* the creation of an internal database containing mitigations for vulnerabilities and other information that helps address the issues, *(ii)* the ability to run a single analysis at a time and *(iii)* the generation of a human-readable report.

We decided to integrate our tool within the *Service Layer* (see Figure 4.4) to offer an on-demand analysis to both other services and the dashboard. To maintain TLSAssistant's independence

while being able to satisfy all the FINSEC requirements, the integration was separated in three phases:

1. extension of TLSAssistant's output capabilities to provide an option for STIX-generated reports;
2. creation of a *Connector* able to translate TLSAssistant's STIX output in FINSTIX and to link our tool with the FINSEC's *Data Layer*, avoiding data redundancy and to maintain consistency across multiple analysis;
3. creation of a set of REST APIs and of a queue manager to allow concurrent requests to be served sequentially.

The following sections will detail these phases.

4.3.1 STIX Output in TLSAssistant

Sharing intelligence with automated services required producing structured data that are consumable by other services through a persistent data store. For this reason, starting from version v1.2, TLSAssistant is able to export the analysis result in STIX (see Section 4.2). Figure 4.5 illustrates, for the Bar Mitzvah attack, how the TLSAssistant report has been modified to map the vulnerability and mitigation output to STIX objects and relationships. After each scan and for each vulnerability discovered, TLSAssistant generates a JSON file containing the following entries:

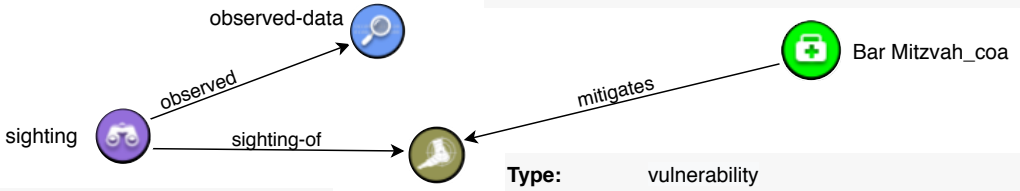
`vulnerability` is an SDO that indicates a weakness that can be used by an attacker to compromise a system. The CVE ID [MIT] that provides a common name for known vulnerabilities is usually present in the `external_references` property. In TLSAssistant, the `vulnerability` SDO contains the name and description of the detected vulnerability.

`course of action` is an object used to suggest actions that may be taken in response to a CTI. It describes technical responses (such as patches) or actions at a higher level (e.g., policy changes). In TLSAssistant, the `course of action` SDO contains the textual description of the mitigation (in the `description` field) and the actionable mitigation in the form of code snippets (in the `x_actions` custom field).

`relationship` is an SRO used to link together two SDOs or STIX Cyber-observable Objects (SCOs) in order to describe how they are related to each other. STIX defines many relationship types, e.g. `uses`, `targets`, `mitigates`. In TLSAssistant, a `course of action` is linked with a `vulnerability` through the `mitigates` relationship type.

Type:	observed-data
Id:	observed-data--4c487cd2-c9d8-4592-af05-f3f36b2ffdbc
Created:	2020-04-20T13:16:49.836Z
Modified:	2020-04-20T13:16:49.836Z
First observed:	2020-04-20T09:16:49.834124Z
Last observed:	2020-04-20T09:16:49.834124Z
Number observed:	1
Objects:	{"0": { "type": "url", "value": "www.example.com" }}

Type:	course-of-action
Id:	course-of-action--655febbb-2063-4a01-87c0-
Created:	2020-04-20T13:19:25.347Z
Modified:	2020-04-20T13:19:25.347Z
Name:	Bar Mitzvah_coa
Description:	Disable the RC4 stream cipher.
X actions:	1. open your Apache configuration file (default: */etc/apache2/sites-available/default-ssl.conf*); 2. find the line starting with: ***SSLCipherSuite***; 3. add the string `:!RC4` at the end. N.B. restart the server by typing: `sudo service apache2 restart`.



Type:	sighting
Id:	sighting--c892b89a-af13-4efe-ad16-851949831e47
Created:	2020-04-20T13:16:49.837Z
Modified:	2020-04-20T13:16:49.837Z
Sighting of:	vulnerability--1a2c1635-6923-472d-8d32-1a90d80decd8
Observed data:	observed-data--4c487cd2-c9d8-4592-af05-f3f36b2ffdbc

Type:	vulnerability
Id:	vulnerability--1a2c1635-6923-472d-8d32-1a90d80decd8
Created:	2020-04-20T13:19:25.347Z
Modified:	2020-04-20T13:19:25.347Z
Name:	Bar Mitzvah
Description:	By exploiting the invariance weakness of the RC4 stream cipher, an attacker is able to retrieve the session cookie by guessing the LSBs (least significant bits) of the keystream. After a phase in which the attacker sniffs the connection between two parties, it detects a weak key usage and tries to exploit the weakness.

Figure 4.5: STIX output for the Bar Mitzvah attack

`observed data` is an SDO that contains information about entities (e.g., files and systems) using the SCOs to provide supporting context. It is not an intelligence assertion, it is simply the raw information without any context for what it means. In TLSAssistant, the `observed data` SDO contains info about the asset observed (e.g., the URL analyzed and the timestamp of the scan).

`sighting` is an SRO that denotes the belief that something in CTI was seen. In TLSAssistant, the `sighting` SRO links the `vulnerability` SDO and the `observed data` SDO through the `observed` and `sighting of` relationship types.

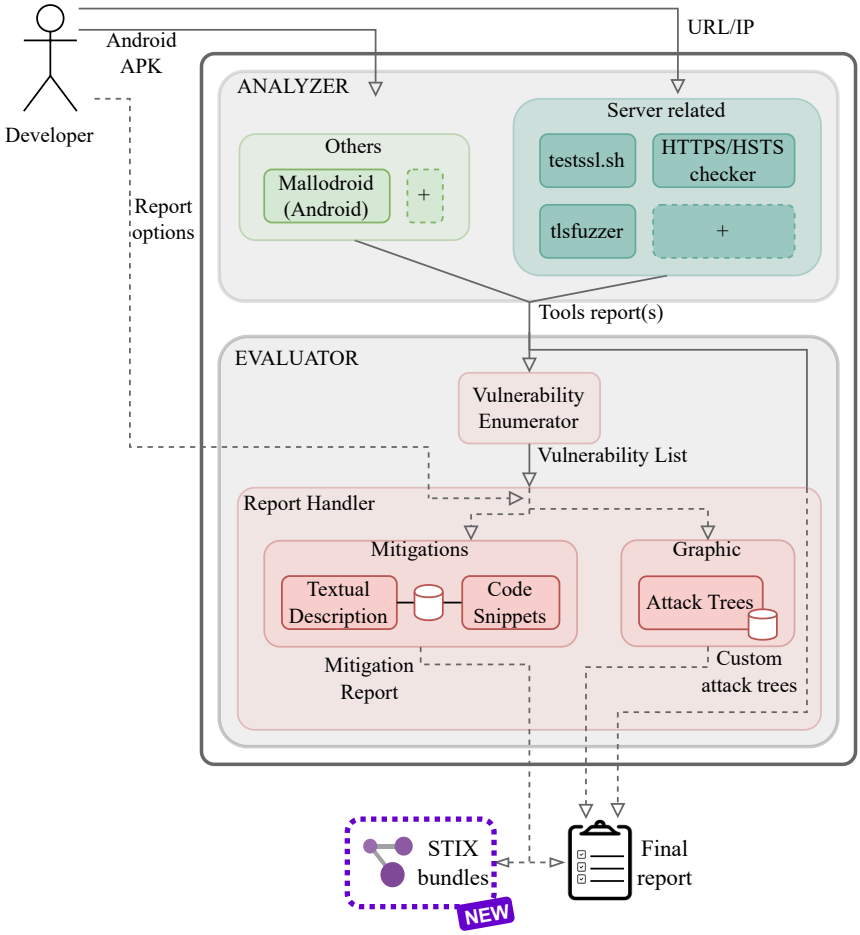


Figure 4.6: TLSAssistant v1.2 architecture

4.3.2 FINSEC Connector

The *Connector* is a Python script written to integrate TLSAssistant in the FINSEC platform. Figure 4.3 shows its two modes of operation: `on_deployment` and `on_scan`.

on_deployment the *Connector* is invoked once during the deployment phase of the container. In order to avoid data inconsistencies or duplicates, it checks if a deployment has already occurred upon invocation. In this mode, the connector's responsibility is to connect TLSAssistant 's intelligence to *Data Layer* content. In particular:

1. it exports TLSAssistant's internal database. This will create a set of STIX bundles (see Section 4.3.1) containing three objects: a `vulnerability`, a `course of action`, and a `relationship` of type `mitigates`;

2. using the exported vulnerabilities, it retrieves their IDs from the *Data Layer* then edits all the `relationship` objects. By doing this, each `course of action` will be linked with the proper object within the shared database. Two edge cases can occur:
   - if a single `course of action` is able to mitigate more than one vulnerability, the connector will create an "aggregated" `vulnerability` object and link it to the mitigation (to avoid SRO redundancy);
   - if a vulnerability extracted from TLSAssistant does not have a CVE (hence the *Data Layer* will not contain its object), a new `vulnerability` will be created and uploaded;
3. it extends the structure of each `course of action` by adding FINSTIX properties (see Section 4.2). These (e.g., `'x_subtype' = 'to_dashboard'`) are used to extend the STIX language and manage the integration with the *Dashboard*;
4. lists all the created and linked vulnerabilities in a file stored locally (this is the file whose existence will be checked on startup) and uploads all `course of action` and `mitigates` objects.

on_scan the *Connector* is called after every completed scan. Once started, the connector retrieves the JSON files generated by TLSAssistant (see Section 4.3.1), and performs the following operations (for each file):

1. extracts the `sighting`, the `observed data` (see Figure 4.5) and the name of the detected `vulnerability`;
2. matches the name of the vulnerability and links the `sighting` to its ID in the *Data Layer* (value retrieved or generated during the deployment);
3. tags each `sighting` object with a custom `scan_id` field so that the initiating service could retrieve the results;
4. finally, it uploads both `sighting` and `observed data` to the *Data Layer*, allowing the *Dashboard*, the initiating service, and any other entity to retrieve the scan results, now available within the shared database.

4.3.3 API and Queue Handling

Each TLSAssistant scan can take anywhere between 1 and 5 minutes, and an installation of TLSAssistant was never designed to handle concurrent requests. Making the service available therefore required not only the definition of an API to initiate scans, but also a message queue for requests to be passed to worker threads, and a data store for states and results to be queried.

Figure 4.7 shows a component diagram of TLSAssistant integrated in the FINSEC platform, with our contribution (colored) composed of the following:

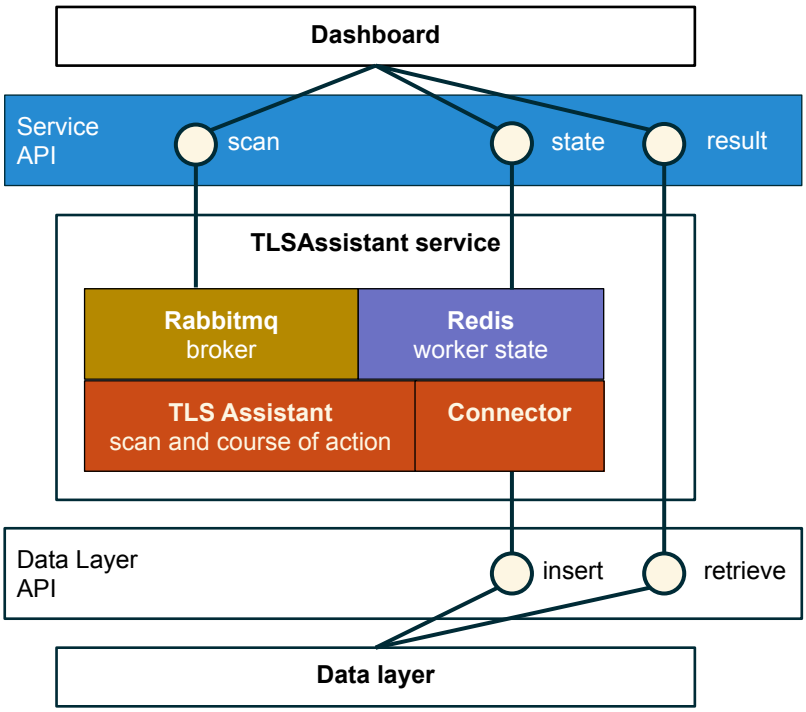


Figure 4.7: Integration of TLSAssistant in the FINSEC architecture

- TLSAssistant+ *Connector* to analyze the deployments and interact with the *Data Layer* (see Section 4.3.2);
- a set of Service APIs exposed using Flask [Pal] and leveraging the Celery [Sol] task queue;
- `rabbitmq` broker for scan requests;
- `redis` to store scan states.

In detail, our service exposes the following Service APIs:

`POST /scan` a JSON object with a `url` key on which to initiate a vulnerability scan; returns a `scan_id` - a UUID as per RFC 4122.

`GET /state/{scan_id}` returns the current scan state as reported by Celery (e.g. `pending`, `success` and `failure`).

`GET /result/{scan_id}` retrieves all `sighting` for the given `scan_id`, and all associated objects such as `vulnerability` and `course of action`, from the *Data Layer*.

Finally, enhancing the functionality of other services to achieve a synergy beyond their consumption of the generated STIX objects required individual integration, e.g. Risk Assessment - see Section 4.4.

4.4 Integration with Risk Assessment

As mentioned at the beginning of the chapter, one of the most significant benefits of integrating a tool such as TLSAssistant into the FINSEC platform is that it improves the functionality of other services. Providing a comprehensive assessment of the security risks posed by an infrastructure or system is crucial for any security platform. We thus consider how TLSAssistant can contribute information to refine the calculation of the Risk Assessment Engine (RAE) available in the FINSEC platform in order to support a continuous risk evaluation process.

The RAE is designed to support a continuous monitoring of assets; this allows a Security Officer at a financial institution to maintain a list, for instance, of TLS servers in their infrastructure, including but not limited to Online Banking or PSD2 API servers, and have them regularly scanned for vulnerabilities. While changes to server configurations should only occur infrequently and as a result of manual intervention by a system administrator, a regular scan mitigates against undetected malicious changes, unintended consequences e.g. of upgrades to dependencies, and automated changes. Regular scans may reveal vulnerabilities that necessitate a reassessment of the current risk level. If the vulnerabilities are associated with a particular CVE, this can be cross-referenced with those potentially present on the asset by manufacturer and model, and its impact can be evaluated using CVSS scores.

The RAE integrated in the FINSEC platform [FIN20] takes an approach based on graphical risk modeling described in [uEG+18]. This entails the creation of a CORAS model [LSS11] based on an understanding of the overall risk pattern to be modeled, as well as the definition of risk assessment algorithms - Bayesian networks in R and decision diagrams in DEXi [Boh22] - for an automated quantitative and qualitative risk assessment.

The RAE has been adapted to consume STIX `sighting` objects to trigger re-evaluation of risk models. As noted in Section 4.3.1, TLSAssistant produces `sighting` objects linked to cyber observables; the RAE however measures risks associated with an `x-asset` object, developed specifically for the FINSTIX data model [FIN19b]. This object links specific cyber-physical entities, e.g. servers, with their cyber-physical address (e.g. LAN IP) and/or physical location, as well as parameters relevant to risk assessment. A synergy between our vulnerability scanning service and the RAE can be achieved by (a) linking each scan of an address with a known `x-asset`, on the part of the scanning service, and (b) adding `sighting` of the relevant `vulnerability` objects to the RAE risk models.

The following modifications were made specifically to integrate with the RAE:

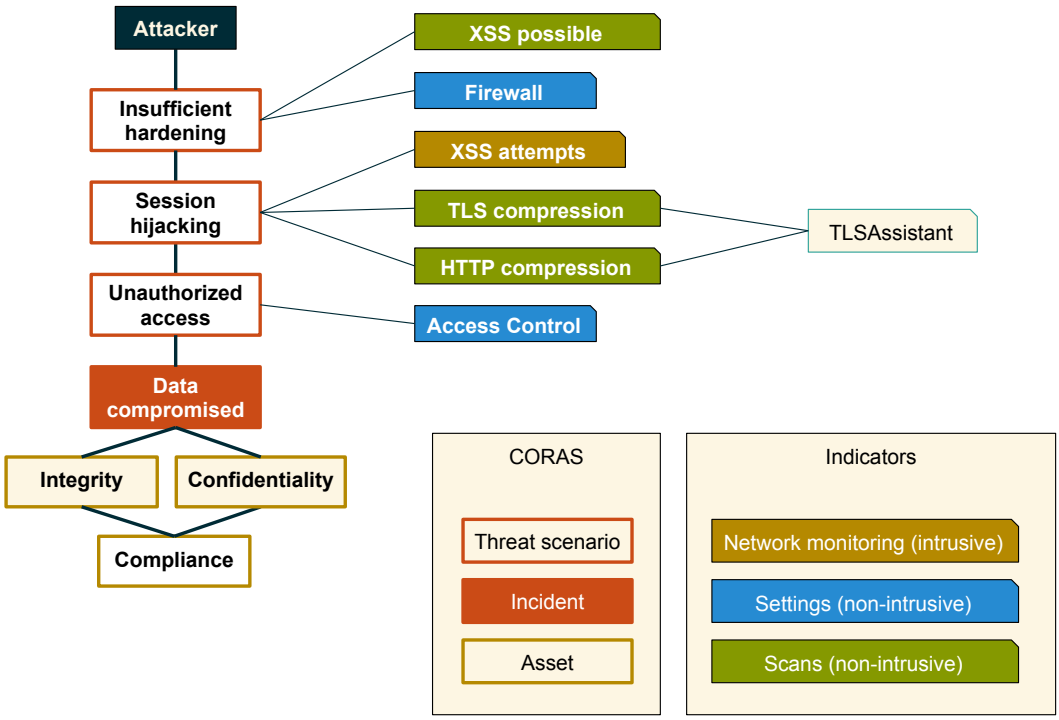


Figure 4.8: Integration of TLSAssistant in a Risk Assessment Engine model

`x-asset` objects in the *Data Layer* had a `url` property added;

`scan/` API endpoint calls can be made with an `x-asset-id` JSON element; the API will retrieve the corresponding object from the *Data Layer* and read its `url` property;

`x-asset-id` provided to the API is added to the `x_asset_refs[]` property in `sighting` objects produced by TLSAssistant;

CORAS risk models were updated to include some of the specific CVE that TLSAssistant can produce `sighting` of. A simplified model of how `sighting` of `vulnerability` may be represented in CORAS is shown in Figure 4.8.

4.5 Lessons Learned and Discussion

Combining diverse security services and integrating their results into a single Dashboard presents a number of intriguing challenges and opportunities. We addressed these issues by extending TL-

SAssistant's capabilities and bridging its core to FINSEC's *Service Layer*, allowing it to process concurrent requests and exchange cyber threat intelligence.

We began this work with the intention of integrating our tool to combine multiple sources of security intelligence, but discovered indirectly how difficult it is to integrate a standalone tool into a distributed framework without substantially modifying the tool. This was especially difficult in our case due to the fact that TLSAssistant was written in Bash and therefore does not benefit from object-oriented programming or modularity. The difficulties we encountered during this process made clear the need to completely redesign the architecture of the tool. By considering modularity, extensibility, and the possibility of future integration, future users should be able to easily integrate our tool into their pipeline. Following upgrades and research-driven design decisions, the following chapter describes the procedure that led to the redesign of TLSAssistant.

Chapter 5

From Standalone Tool to Collaborative Framework

With TLS serving as a centerpiece for numerous modern technologies, the ability to provide an "overall assessment" of a system, analyze evolving risks, and monitor new vulnerabilities is of primary importance. This conflicted with the limitations imposed by the original TLSAssistant architecture.

In this chapter, we will *(i)* discuss the challenges and limitations encountered while working on TLSAssistant over the years, *(ii)* introduce a new architecture for TLSAssistant v2, designed to be modular and easily extendable, and *(iv)* review the main changes, including a summary of the available features.

5.1 Discussion: Challenges and Limitations

As anticipated in Section 4.5, TLSAssistant development started in 2018 with a specific set of features in mind and to cover a specific set of use cases. With each new piece of work, new research questions arose, and the desire to improve its capabilities and research impact led to an unanticipated increase in code size and complexity. As TLSAssistant was created using the scripting language Bash, it could not take advantage of the decoupling capabilities that an object-oriented programming language could offer. This resulted in frequent micro-redesigns of subsets of the code for each incremental change. The integration into the FINSEC platform was the final straw when we realized that our original design for a standalone tool prevented us from sharing threat intelligence with other systems.

With the emergence of new research lines and the desire to extend TLSAssistant's capabilities to aid in their study, we decided to build the tool from scratch. The goal was to develop a

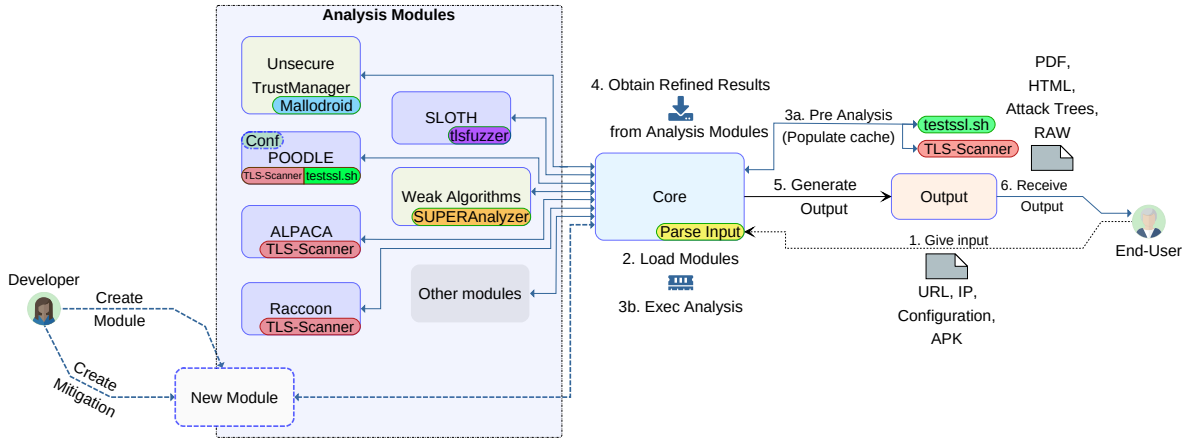


Figure 5.1: TLSAssistant v2 architecture

modular, extensible framework for the creation of new analysis approaches and tools that would enable us to perform various checks not only on the protocol itself, but also on other aspects such as the analysis of: *(i)* client-side applications *(ii)* TLS libraries (such as OpenSSL), *(iii)* certificates' validity, and *(iv)* TLS enforcement policies (e.g., HTTP Strict Transport Security). Thus, we established a primary structural objective: to make TLSAssistant modular, with the ability to add new features without substantially modifying the source code. As non-functional requirements, we desired a portable tool that is versatile, quick, and simple to operate.

The benefits of a modular and extensible framework are twofold: *(i)* the ability to streamline the upgrade process, allowing third-parties and internal contributors to rely on a reliable core system capable of automatically integrating new modules; and *(ii)* the ability to easily integrate with other cybersecurity tools or platforms.

5.2 Architecture Definition

Originally, the TLSAssistant architecture consisted of two main components that exchanged information by creating local files and invoking different scripts to generate a report that highlights detected vulnerabilities and provides information for their mitigation. As the code was written in Bash, the addition of new capabilities required extensive refactoring. As a result, we decided to redesign TLSAssistant as a modular system (see Figure 5.1), with components encapsulated by type: *Analysis* modules to check for vulnerabilities, *Core* modules for information exchange between modules, and *Output* modules to provide properly formatted output for the user.

The users of our tool are the *third-party developer* who will create new modules and the *end-*

users who will analyze TLS-related vulnerabilities using the tool.

Flow for the Developer To add a new $Analysis$ module (see left of Figure 5.1), the developer must create it according to the provided specifications. By doing so, the $Core$ module will recognize and incorporate the new feature. In addition to providing the code implementing the checks for detecting a vulnerability, if a mitigation is known, the developer must create a JSON file describing it according to the mitigation's standard (see Section 5.2.2 for details).

Flow for the End-User The usage of the tool can be split in two steps. In Step 1 (see right of Figure 5.1), the end-user chooses which modules to employ in the analysis by supplying a configuration file or command line list. Each configuration file contains a list of modules that conduct a particular kind of analysis (e.g., checking for vulnerabilities related to weak TLS ciphers). Step 2 involves the $Core$ loading the configuration (if provided) and modules (from the command line list or configuration) to ensure they are relevant to the requested type of analysis. TLSAssistant v2 currently supports four types of analyses.

Single Host TLSAssistant v2 selects and executes the necessary modules. Every tool repeatedly connects to the target webserver by sending multiple specially crafted *Client Hello* messages. By analyzing the webserver's response, each tool is capable of passively recognize the active configuration. The $Output$ module then parses and collects the results collected from each module;

Multiple Hosts We perform a *Single Host analysis* on each one of the webservers specified in an input list. Each result is concatenated and provided as a single output to the $Output$ module;

TLS Configuration and Fixes If a configuration file is provided, its content is analyzed by all available modules. Alternately, if a configuration file and a valid hostname are supplied, a *Single Host analysis* is conducted, and the fixes are then incorporated into the provided TLS configuration;

Single APK If the user provides an APK file, TLSAssistant v2 decompiles its source code and executes each Android-related module (e.g., *unsecure TrustManager* wrapped from MalloDroid and *KeyStore disclosure* from SUPERAndroidAnalyzer) to check for potential security flaws.

The $Analysis$ modules will return the refined results (analysis results and related mitigations, if applicable) to the $Core$ (Step 4), which will group them and supply the $Output$ module with all the merged outputs (Step 5). The latter concludes the execution of the analysis by providing the user with the requested output type, properly combined and formatted in a single file (Step 6).

To provide a more extensible, versatile, and cross-platform tool, Python 3 was chosen as the reimplementation language. In addition, we selected JSON as the primary communication language for both inter-module communications and configuration files because it is interoperable,

(almost) human-readable, and Python-compatible. We have decided to allow developers to install new dependencies by providing a link to their source code on the `dependencies.JSON` file and defining their respective types. Then, the dynamic installer will retrieve and install the necessary files.

5.2.1 Modules Characterization

To make TLSAssistant v2 more flexible, the software was organized into modules. Contextually, a module is a collection of objects and classes stored in the same file (i.e. the type of vulnerability analyzed). Each module adheres to standards[1] designed to make rewriting or adding new features quick compatible to the *Core*.

5.2.1.1 Core Module

This is the central module that receives the user's analysis request, loads the configuration (if any) along with the *Analysis* modules, and then calls the *Output* module with the newly obtained results. As with the previous version of TLSAssistant, the analysis can be performed on a single host or APK, with the added ability to analyze a list of domains obtained from a file and apply fixes to a TLS configuration. We will now discuss the three primary categories of analysis: BlackBox, WhiteBox, and Hybrid.

5.2.1.1.1 BlackBox Analysis For performing a BlackBox analysis of a given hostname (i.e. detecting vulnerabilities and various configuration issues by repeatedly connecting to the target server and evaluating its responses), the *Core* of the framework is equipped with a configuration system that permits the end-user to fine-tune the software. Each configuration includes a list of *Analysis* modules that are parsed and utilized by the *Core*. The user can provide (e.g., `--modules poodle mitzvah 3shake`) or remove (e.g., `--exclude sloth`) a set of modules via the command line. We have added the command line interface for less experienced users so that they can perform precise and selective analysis without having to modify configurations. TLSAssistant v2 will adopt the default configuration based on the type of analysis the user wishes to perform (server or APK) if the user does not provide a configuration. This configuration is called `default_[TYPE].JSON` (see Listing 5.1).

```
1 {
2   "name": "server_default_configuration",
3   "description": "Default server configuration",
4   "modules": [
```

[1] *Analysis* modules must comply with the standards (see Section 5.2.2), while Output/Wrapper modules might comply with the standards but are not required to.

```
    "3shake",
    "beast",
    "breach",
    "ccs_injection",
    "certificate_transparency",
    "crime",
    "drown",
    "freak",
    "heartbleed",
    "hsts_preloading",
    "hsts_set",
    "https_enforced",
    "logjam",
    "lucky13",
    "mitzvah",
    "nomore",
    "pfs",
    "poodle",
    "renegotiation",
    "robot",
    "sloth",
    "sweet32",
    "ticketbleed"
  ]
}
```

Listing 5.1: *default_server.json*

A configuration may contain a second one, allowing an expert user to nest and reuse configurations with minimal modifications. To illustrate the benefits of a nested approach, consider the following scenario: a system administrator is required by internal policies to check the security posture of new endpoints, and a cyber risk assessment team (CRT) provides the policy compliant configuration to be executed with TLSAssistant. This approach places the responsibility of providing a correct configuration on the team of experts, relieving the system administrator of almost all policy definition responsibilities and leaving him or her with the task of running the analysis with TLSAssistant.

If the configuration contains an `include` directive, the software loads the included configuration recursively. After the `include` chain has been completed, the presence of a `remove` field is examined. The `remove` field will have a (complete or partial) configuration portion, and all corresponding values found within the (partial) configuration in this field will be removed from the recursively included configuration. Upon completion of this final step, the presence of an `add` field is verified. Similar to the preceding step, the `add` field will contain a configuration (total or partial) that will be appended to the included configuration. It is essential to note that the execution order is as shown, i.e. `include` then `remove` and then `add`. Listing 5.3 shows a configuration including a second one (Listing 5.2).

In the previous example, a company could have two configurations: one for general use and one for specific services. If the CRT must configure an endpoint with specific security restrictions, it will only need to merge the two configurations (applying changes to the first, generic one). By using the directives[2] correctly, the CRT will be able to generate a new configuration for the specific use case without modifying either of the two source files directly. The same procedure can be repeated for each endpoint by contextually adapting the directives to the selected use case.

In terms of analysis types, this category of analysis relates to the "*Single Host*" and "*Single APK*".

```
{
    'name':'Compliance',
    'modules':['Compliance','stix'],
    'args':{
        'stix':['c://dkgd/file.out'],
        'Compliance': [ 'Generic' ]
    }
}
```

Listing 5.2: Generic configuration

```
{
  'name':'AGID Compliance',
    'include':{
    'file':'compliance.json',
    'remove':{
      'modules':['stix'],
              'args',
    },
    'add':  {
      'modules':['TLS'],
      'args':{
        'TLS':['d','192.168.1.1','compliance'],
        'Compliance': [ 'AGID' ]
      }
    }
  }
}
```

Listing 5.3: Configuration including Listings 5.2

5.2.1.1.2 WhiteBox and Hybrid Analysis with Automatic Fixing By having access to the TLS configuration, we can analyze it for misconfigurations and immediately implement vulnerability fixes. In addition, we have chosen to perform a complete analysis by default whenever a TLS configuration is provided, allowing the user to choose between applying the fix immediately

[2] `include`, `add` and `remove`

or saving it to a separate file. We proceeded in this manner because the elimination of a single vulnerability may not necessarily result in a secure configuration. The user could, for instance, run the analysis and fix only the POODLE attack, but this would not secure the configuration against a SLOTH attack [Duc] (an attack that exploits the availability of weak hash functions).

For the TLS configuration flow we have two types of analysis that can be performed:

WhiteBox: the user provides a TLS configuration as input. The provided configuration is analyzed and the fix is applied at the discretion of the user;

Hybrid: the user provides a TLS configuration and a hostname for analysis as inputs. The specified hostname is analyzed using a standard Single Host analysis (i.e. a BlackBox analysis). Then, as a configuration is provided to apply the fixes, TLSAssistant could directly (at the discretion of the user) apply the fixes related to the detected and reported vulnerabilities on the requested port.

Since we do not know what non-standard ports the user is employing, we have decided to retain the default ports for certain checks. For instance, to determine whether HTTPS is enforced during a WhiteBox TLS configuration analysis, we will examine the virtual host (i.e., the single endpoint specified in the configuration file) with port 80, if present. In the event that the analysis is Hybrid, the port can be specified and this issue will not arise.

The vulnerabilities detected by TLSAssistant v2 can be divided into two groups: those that can be resolved by updating the TLS configuration file and those that require an upgrade to the cryptographic library. We decided to differentiate the former by adding a `conf` attribute to modules that manage them (e.g., POODLE in Figure 5.1). During TLS configuration analysis, *Analysis* modules with this attribute are selected. The attribute will perform the simple virtual host analysis of a configuration for the selected vulnerability and fix it[3] by automatically replacing the misconfigured line of code with the correct version, per the *Analysis* module's instructions.

As for the latter group of vulnerabilities, many of which are no longer exploitable in newer versions of OpenSSL, we address them by requesting that the user update the cryptographic library. In addition to performing checks for each TLS configuration vulnerability, we analyzed which library version is deemed "safe" by examining the provided changelog [Ope22]. In the changelog, we determined when a given cipher and/or protocol is deemed deprecated and removed from the OpenSSL version, leaving the configuration vulnerable only if the OpenSSL library is recompiled. In order to take into account this aspect, the interface also includes an OpenSSL attribute. The user must specify on the command line whether to run an analysis for a specific OpenSSL version with `--openssl [VERSION]` or to completely ignore the OpenSSL check with `--ignore-openssl`. If the user requests to ignore OpenSSL, the most vulnerable version

[3] If requested by the user by using `--apply-fix` [path/to/conf].

of OpenSSL is assumed during analysis. The OpenSSL version check is performed if the user provides a version of OpenSSL and the provided TLS configuration is vulnerable. If the provided version is lower than the one deemed secure for the analyzed module, the TLS configuration is deemed insecure and a mitigation is displayed.

We have decided to force the user to choose between providing an OpenSSL version and ignoring OpenSSL checks, so that the user is aware of the relationship between vulnerability results and OpenSSL version. OpenSSL was selected as the cryptographic library of choice because it is included by default in the Linux kernel and is regarded as one of the fundamental building blocks of the Internet's cryptographic infrastructure [Dat23]. Individual modules can still be excluded with the proper command, allowing an expert user to fine-tune the analysis he or she is not interested in.

5.2.1.2 Analysis Modules

This type of modules conduct the analysis by invoking the integrated cutting-edge tools and then refining the results. Wrapper refers to submodules that call external software or dependencies. These "wrappers" produce a raw output that should never reach the $Output$ module directly. The $Analysis$ modules perform the scan, either by passing through a wrapper or by implementing the logic necessary to conduct the analysis themselves. They accept parameters as input, invoke the corresponding wrapper module (if any), refine the raw output of wrappers by adding mitigations, and pass the refined output to the $Core$, which in turn lends it to the $Output$ module. tlsfuzzer, for instance, performs the analysis on SLOTH. Through the "tlsfuzzer" wrapper, TLSAssistant v2 will launch the external tool. The $Analysis$ module receives the raw results and, if the webserver is vulnerable, refines them by adding mitigations and delivering them to the $Core$ module. In light of the fact that the majority of $Analysis$ modules use the same wrappers with different arguments, we created template modules through inheritance. By acting as interfaces, these modules override methods to set mitigations and parameters for wrapper methods invoked to transform raw data into contextual data for the specified module.

5.2.1.2.1 Refining the concept of wrapper An external application or API service is contained by a wrapper module. Each wrapper includes a caching mechanism that enables the tool to analyze the same domain (or IP) multiple times during the same use session. Despite the fact that invoked tools are rarely detected by Intrusion Detection Systems (IDS), they may be if they are utilized repeatedly. This effect is mitigated by the use of a cache because it is unlikely that the server configuration will change during the same usage session (which, in most cases, takes only a few minutes). The "force" parameter allows module developers to reset the cache if necessary. When the developer needs to perform multiple analyses on the same hostname in order to compare results, forcing the analysis is useful. For instance, a developer could create a module that verifies the supplied hostname, collects data, and then waits until the user certifies

that the vulnerability has been patched. The following paragraphs describe how different tools (described in Section 3.1) were integrated using wrappers.

5.2.1.2.2 tlsfuzzer tlsfuzzer is a test suite for SSL and TLS implementations. Given the number of available scripts and in an effort to simplify the integration of new modules, we chose to create an interface that allowed us to provide the names of a series of scripts to be executed with their arguments. Given the set A of input scripts for analysis by the module and the set B of scripts whose results are cached, the tlsfuzzer wrapper module performs $A \setminus B$ to identify the missing scripts. The missing scripts will be executed, and the cache will be updated by adding the scripts' new results. Finally, the module will return the result to the $Analysis$ module that requested the wrapper.

5.2.1.2.3 testssl.sh testssl.sh is a free command line tool which checks a server's service on any port for the support of TLS/SSL ciphers, protocols as well as recent cryptographic flaws and more. To be able to use this Bash-based tool, we had to create a wrapper that, when given a list of arguments to execute and a hostname and port, calls the tool with the provided arguments on the specified hostname:port.

Since testssl.sh analyzes the majority of vulnerabilities checked by TLSAssistant v2, the wrapper module will benefit most from the cache system. Due to the webserver connections opened by testssl.sh, the boot time would be a few seconds ($\simeq$ 3 sec[4]) whenever a testssl.sh-related module is executed. Therefore, we had to strike a balance between module atomicity, efficiency, and execution speed. For this purpose, we developed a method in the $Core$ that, for each provided module, determines whether or not it is related to testssl.sh. In this case, the method collects the arguments used by the module to perform the analysis with the wrapper testssl.sh, and the vulnerability analysis associated with testssl.sh is performed prior to calling the individual methods, during a single run of the tool. This populates the cache of the testssl.sh wrapper so that any module using the wrapper does not need to rerun the analysis and wait for boot time because the results are already cached. Thus, we preserve the atomicity of the individual modules and, since the $Core$ already knows which modules will be executed, we save time by having it perform the pre-analysis.

5.2.1.2.4 TLS-Scanner TLS-Scanner is a state-of-the-art vulnerability scanner that can detect a wide variety of flaws. While it was not initially included in our tool comparison, it has recently been added due to its capacity to detect newly discovered TLS vulnerabilities (i.e. ALPACA and Raccoon). We take the result printed on the console and parse it by extracting the

[4]Benchmarked using "Multipass" software from Canonical to virtualize Ubuntu 20.04 LTS with 1GB of RAM with an Intel i5-8250U (8) @ 3.400GHz processor

required information such as the result of the vulnerability scans, and the list of unsafe ciphers if we are scanning for Padding Oracle or Raccoon vulnerabilities.

TLS-Scanner set of features make it a suitable replacement for testssl.sh in future releases.

5.2.1.2.5 Certificate The wrapper module communicates with the crt.sh API. By providing the hostname to check, this method returns the JSON value of valid certificates for the domain and its subdomains. By obtaining the data of all certificates organized by domain and subdomain, we can verify their validity and signature and obtain an immediate list of valid certificates. This wrapper is also useful for subdomain enumeration because it obtains all certificates associated with a (sub)domain.

5.2.1.2.6 HSTS HTTP Strict Transport Security (HSTS) is an HTTP response header field that specifies that a website can only be accessed through secure connections. Once activated, the directive remains in effect until the specified age is not reached [HJB12].

The "HSTS" wrapper makes raw header calls to the requested hostname specifying the type of result we want to get:

- **HTTPS enforced** we can test for HTTPS enforcement by analyzing the response header of an HTTP port connection. If the status code is 301 (permanently moved) or 302 (temporarily moved) and the location field leads to an HTTPS page, no HTTP connections will be allowed;
- **HSTS Set** it checks for the presence of a valid `strict-transport-security` directive in the header;
- **HSTS Preloaded** the analysis of HSTS preloading is performed by comparing two lists: Mozilla's [Moz19] and Google's [Goo22].
  Since the Google list is included in the Chromium project's source code, the tool will automatically download a base64-encoded file from the Google API during the installation process. This file will be rendered usable and converted to JSON at runtime by the specified wrapper module. The Mozilla list, on the other hand, is easily readable and accessible via an API call made by the installer;
- **Server Info** we can request server information to determine whether the used webserver is Apache or NGINX. This is used to select customized mitigations based on the webserver type. If neither the tool nor the server supports the vulnerability, a generic mitigation is displayed.

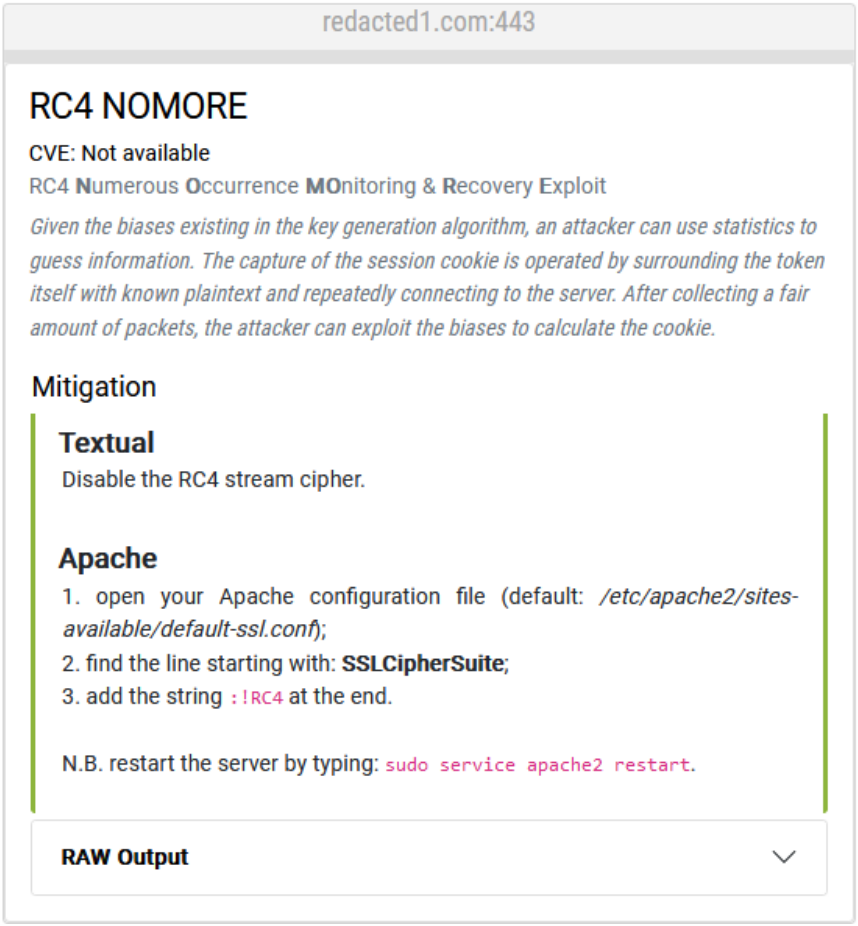


Figure 5.2: TLSAssistant v2 report (excerpt)

5.2.1.2.7 MalloDroid MalloDroid is a small tool built on top of the Androguard reverse engineering framework able to analyze Android apps for broken TLS certificate validation. To simplify the integration of this tool, we developed an internal API that allows MalloDroid to be called both as a script and as an embedded application. We then developed a wrapper to maximize its potential. The wrapper makes a call to the tool and parses the requested APK file by passing the file's path and execution arguments. When the analysis is complete, the results are cached. The results can be accessed using the APK's path as the cache is index. Given the single execution implementation (since the cache is stored in RAM, it is cleared whenever the tool is restarted), the path is regarded as unique for the tool.

5.2.1.2.8 SUPERAndroidAnalyzer SUPER is a command-line application that analyzes .apk files in search of vulnerabilities by decompressing APKs and applying a series of rules to detect a set of known vulnerabilities (see Table 3.1).

Figure 5.3: TLSAssistant v2 scoreboard

To create this wrapper, we employed a parser that removes all non-TLS-related detections (such as Android Manifest file-related checks) by formatting and loading them into the cache. To provide a failsafe mechanism, we made two calls to the tool: if SUPER partially decompresses the APK due to a decompilation error, it will analyze the partial sources (if any) instead of attempting to decompress it again. After executing the tool on the APK file, a recursive search is conducted on all subfolders of the specified temporary folder to save the results. Similar to MalloDroid, the file path is utilized using the same logic and motivations.

5.2.1.3 Output Module

This module receives the results from the *Core* module and generates the output that is readable by the end user. After performing the analysis, the *Core* provides the *Output* module with the concatenated output of each executed analysis. Following the *Core*'s instructions, this module returns the output in the format specified by the user (i.e. HTML, PDF, STIX [OAS23], RAW or Attack Tree [Sch]). Each vulnerability is accompanied by a mitigation description that will assist the user in resolving the issue, as well as a CVE value that can be used by the end-user to conduct a risk assessment (see Figure 5.2).

Given the ability to conduct a multi-domain analysis, it is probable that one or more mitigations will be repeated in the final report. To circumvent this issue, users can group their data "by module" and generate a list of vulnerable endpoints for each threat. If the user chooses to retain the default "by host" grouping, a summary scoreboard (see Figure 5.3) will be displayed to help quickly determine which hosts are susceptible to the detected vulnerabilities. In the case of automatic repair, the TLS configuration changes made by the tool will be included in the final report.

5.2.2 Standards

With the development of TLSAssistant v2, we created a set of standards capable of regulating and facilitating future integrations and extensions. The three available specifications govern module development, the formulation of new mitigations and the structuring of a configuration.

5.2.2.1 Module Definition

5.2.2.1.1 Module rules A module:

- Must
  - have a configuration file;
  - have a main class, which must have:
    - `.input(*args)` method;
    - `.output(*args)` method;
    - `.run(*args)` method;
  - be commented properly;
  - use the same file (except for submodules, these can be in different files);
  - if a module is `server` related:
    - have `hostname` parameter;
    - have `port` parameter;
  - if a module is `android` related:
    - have a `path` parameter;
  - if a module is `wrapper` and if possible:
    - have a caching system.
- Can
  - have submodules;
  - be the union of some modules;
  - have more than one class;
  - have more than one object;
  - have `conf` attribute of type `Config_base`()
    - needed to enable APACHE/NGINX config parsing

        * use inheritance to create a proper `condition()` method
    - have a `stix` attribute of type `Bundled()`
- Should
    - have a mitigation in output if it is not a wrapper module;
    - be as independent as possible.

5.2.2.1.2 Configuration file

The configuration file must define:

- Input: can have multiple inputs:
    - Short description of the input;
    - Type (file, string, integer, others);
    - `path` to the file, from the root folder of the project;
    - `class_name` as the main class name (case sensitive);
- Output: must have a single output at time, should be mutual (OR only):
    - Short description of the output;
    - Type (file, string, integer, others).

An example of these files, that will be stored in `configs/modules`, can be seen in Listing 5.4.

```
{
    'input':
    [
        {

            'name':'a',
            'type': int,
            'description': 'does x, y with z',
            'required':'False'

        },
        {
            'name':'sum',
            'type': Str,
            'description': 'Needed for...',
            'required':'True',
            'args':['sum','s']
        }
    ],
```

```
20      'description': 'Bla Bla Bla',
21      'path':'modules/android/super.py',
22      'class_name':'Super'
23      'output':
24      [
25          {
26              'name':'x',
27              'type':Str,
28              'description':'Keks'
29          }
30      ]
31  }
```

Listing 5.4: TLSAssistant v2 module configuration example

5.2.2.1.3 How to communicate between modules

- use JSON or
- use a `dict()`.

5.2.2.1.4 Output of a module

The output

- Must (unless it is a wrapper module)
  - contain the problem title;
  - contain the problem description;
  - contain a mitigation;
  - use the following structure:

```
{
    'title of the vulnerability':{
        'description':'desc',
        'mitigation':{
            [...]
        }
    }
}
```

- Can
  - have a personalized entry.

5.2.2.2 Standard Mitigations

5.2.2.2.1 Default Name

While creating a mitigation, one must:

- set the name of the mitigation coherent to the module;
- set the file name in upper case;
- use _ instead of `space`;
- use `json`;
- set the extension of the file in `.json` lowercase.

For example, the *rabbit hole* module will have a `RABBIT_HOLE.json` file as a mitigation. The template for all mitigation files is the following:

```
{
    "Entry": {
        "Name": "The name of the vulnerability",
        "ExtendedName": "The extended name of the vulnerability (can be the same as the Name)",
        "Description": "Description of the vulnerability",
        "Mitigation": {
            "Textual": "Textual mitigation.",
            "Snippet": {
                "ServerWeType1": "Snippet",
                "ServerWebType2": "Snippet"
            }
        }
    },
    "#comment": "a",
    "#comment1": "b"
}
```

5.2.2.3 Configuration Requirements

5.2.2.3.1 Definition

Each config file

- Must
  - have a name;
  - list all the employed modules.
- Can

- have personalized entries;
- import other configurations;
- specify the args of each module.

5.2.2.3.2 Additional Notes

- if `include` is used, `modules` and other entries are ignored, except for `name`, `include`;
- The execution priority is:
  1. `include file`
  2. `include remove`
  3. `include add`
- every item inside `add` or `remove` is optional and it follows a generic configuration:
  - `add`: adds the entries to the already included;
  - `remove`: removes the entries received with the `include` statement;
- every other unknown entry can be added in `add`;
- if an unknown entry is added and no module processes it, nothing happens.

5.3 Discussion

In 2018, TLSAssistant development began with a particular set of features in mind and to address a specific set of use cases. New research questions arose with each new piece of work, and the desire to improve its capabilities and research impact resulted in an unanticipated increase in code size and complexity. Due to this and the lack of a future-proof design, we accumulated technical debt over the years. To overcome the technical limitations that arose over time, we decided to redesign and rebuild TLSAssistant. Furthermore we switched from a monolithic to a modular structure gaining the following benefits:

Extensibility

- changes to a module have no effect on other modules, which drastically improves their maintainability and upgradeability;
- developers can design and implement modules that are automatically recognized by the *Core* and usable out-of-the-box and without any modification to other modules.

Figure 5.4: TLSAssistant v2 logo

Interoperability

TLSAssistant v2 can now be seamlessly integrated by exchanging threat intelligence using STIX or by easily developing a bridge module that can generate an output in the desired format.

Usability

- users can now choose which modules to execute on each analysis by creating custom configurations that can be combined and serve multiple use cases;
- developers that whish to create new modules can now benefit from a set of consistent standards that enable them to easily.

TLSAssistant v2 current set of features can be summarized as follows:

- vulnerability analysis on:
  - a single or multiple deployed webserver(s) (BlackBox analysis) with the generation of an actionable report;
  - a configuration file (WhiteBox analysis) with the possibility to automatically apply the suggested mitigations;
  - a single APK file, generating an actionable report containing suggestions and best practices on how to fix the detected configuration issues;
- actionable report generation containing:
  - the full list of detected misconfigurations (the full set of detectable misconfiguration can be seen in Tables 3.3 and 3.4);

  - an explanation on how the attack works and its impact;
  - a brief explanation in natural language on how to mitigate it;
  - a set of webserver-specific actionable hints able to guide both system administrators and app developers towards the securing of the target;
  - optional highlighted attack trees to show the full attack of the detected issues.
- standalone STIX bundles generation containing a:
  - `vulnerability` SDO;
  - `course of action` SDO;
  - `relationship` SRO;
  - `observed data` SDO;
  - `sighting` SRO.

Lastly, TLSAssistant v2 also received an official logo (shown in Figure 5.4).

Chapter 6

An Assisted Methodology to Evaluate Security Compliance

Administrators tasked with configuring TLS servers must make numerous decisions (e.g., selecting the appropriate ciphers, signature algorithms, and TLS extensions), and it may not be obvious, even to security experts, which decisions may expose to attacks. Heartbleed [Sho22b], an attack discovered in 2014 that exploits an insecure implementation of the Heartbeat TLS extension [Syn14], can still compromise the privacy of approximately 203,000 websites, for instance [Sho22b].

Based on RFC documents and IANA public registries for standards and parameters [IAN], (inter)national bodies such as PCI [PSSC22], European Commission [Com22], and cybersecurity agencies such as US NIST [U.S22] and Italian AgID [Pre22] issue guidelines to define the use and configuration of TLS in various contexts. Absence of support for these standards and guidelines may result in a reduction of the security threshold recommended by technical agencies, fines from supra-governmental organizations, or even exclusion from international payment systems. Therefore, it is extremely important to assist system administrators in configuring webservers to conform to these standards.

In this chapter, we will *(i)* introduce a set of banking standards and security guidelines issued to regulate TLS configurations in different scenarios, *(ii)* present the state-of-the-art in terms of TLS compliance assessment tools, *(iii)* describe a methodology that can be used to assess the compliance of a target webserver against various agency-issued guidelines, and *(iv)* describe the work done on prototyping it.

6.1 Banking Standards

Banking activities are regulated on multiple levels and by various institutions, ranging from individual states to supranational organizations. In past years, we have decided to investigate how two banking-related standards, PSD2 [Com15] and PCI-DSS [PCI18], regulate communication security.

6.1.1 PSD2

The European Directive 2015/2366, also known as the revised Payment Services Directive (abbreviated as PSD2), makes it mandatory for banks and other entities that manage the financial accounts of their customers to open and make easily accessible their information to third party service providers in a secure manner, with the customer's consent. In order for banks to comply with PSD2, they must adhere to the *Regulatory Technical Standards* (RTS) [Com17], which establish security requirements (such as the use of strong customer authentication) and guidelines for open interfaces. Section 2 of the RTS contains the following security-related articles:

- Article 30 - General obligations for access interfaces
  - *the integrity and confidentiality of the personalized security credentials and of authentication codes transmitted by or through the payment initiation service provider or the account information service provider shall be ensured*;
  - *Account Servicing Payment Service Provider (ASPSP) shall ensure that their interfaces follow standards of communication which are issued by international or European standardization organizations*;
  - *Competent authorities shall ensure that ASPSP comply at all times with the obligations included in these standards in relation to the interface(s) that they put in place*;
- Article 34 - Certificates
  - *For the purpose of identification [...] Payment Service Provider (PSP) shall rely on qualified certificates for electronic seals as referred to in Article 3(30) of Regulation (EU) No 910/2014 or for website authentication as referred to in Article 3(39) of that Regulation*;
- Article 35 - Security of communication session
  - *ASPSP, PSP issuing card-based payment instruments, Account Information Service Provider (AISP) and Payment Initiation Service Provider (PISP) shall ensure that, when exchanging data by means of the internet, secure encryption is applied between*

the communicating parties throughout the respective communication session in order to safeguard the confidentiality and the integrity of the data, using strong and widely recognized encryption techniques.

6.1.2 PCI-DSS

The Payment Card Industry Data Security Standard (PCI DSS) - created by Visa, MasterCard, Discover, and American Express in 2004 [Tea19] - is a widely accepted set of policies and procedures designed to optimize the security of credit, debit, and cash card transactions and protect cardholders' personal information from misuse. Its primary document, "Requirements and Security Assessment Procedures", contains a list of 12 requirements that must be met for a system to be deemed compliant. Each requirement is divided into sub-requirements that describe the testing procedure and the rationale for the requirement (e.g., intent or security objective). As the set of requirements is very extensive and covers various aspects of a company's assets, the following TLS-related requirements were extracted:[1]

- Requirement 1: Install and maintain a firewall configuration to protect cardholder data
  - *Identify insecure services, protocols, and ports allowed; and verify that security features are documented for each service*;
- Requirement 2: Do not use vendor-supplied defaults for system passwords and other security parameters
  - *Identify any enabled insecure services, daemons, or protocols and interview personnel to verify they are justified per documented configuration standards*;
  - *Inspect configuration settings to verify that security features are documented and implemented for all insecure services, daemons, or protocols*;
  - *Examine the system configuration standards to verify that common security parameter settings are included*;
- Requirement 4: Encrypt transmission of cardholder data across open, public networks
  - *Select and observe a sample of inbound and outbound transmissions as they occur*;
  - *Examine keys and certificates to verify that only trusted keys and/or certificates are accepted*;
  - *Examine system configurations to verify that the protocol is implemented to use only secure configurations and does not support insecure versions or configurations*;

[1] Even though PCI-DSS v4.0 was published in March 2022, we refer to v3.2.1, the most recent version available when conducting this analysis. PCI-DSS v3.2.1 will remain active for two years after v4.0 is published.

  - *Examine system configurations to verify that the proper encryption strength is implemented for the encryption methodology in use*;
  - *For TLS implementations, examine system configurations to verify that TLS is enabled whenever cardholder data is transmitted or received*;

- Requirement 6: Develop and maintain secure systems and applications
  - *For public-facing web applications, address new threats and vulnerabilities on an ongoing basis and ensure these applications are protected against known attacks by either of the following methods [...]*;

- Requirement 11: Regularly test security systems and processes.
  - *Run internal and external network vulnerability scans at least quarterly and after any significant change in the network*:
  - *Review output from the four most recent quarters of external vulnerability scans and verify that four quarterly external vulnerability scans occurred in the most recent 12-month period.*

6.1.3 Discussion

As PSD2 articles cannot be implemented as a set of technical checks, the directive creates a gray area between an ideal security level and a webserver's actual configuration. The stated safety requirements are ambiguous due to the absence of specific security threshold indications. In addition, the articles' legalese language makes it easy for system administrators to make subjective configuration choices that cannot be supported or discouraged by third parties.

In contrast to PSD2, PCI-DSS provides a document mapping the "Requirements and Security Assessment Procedures" to the NIST Cybersecurity Framework [NIS22] and other standards. This does not, however, facilitate the translation of each requirement into a technical test (as it does not, for example, map to SP 800-52 Rev. 2, a set of guidelines that will be introduced in the next section). We decided to abandon these standards in favor of national TLS recommendations for this reason. While their impact diminishes, they can be translated into defined checks, bringing the work closer to our previous field of work.

6.2 National TLS Guidelines

TLS guidelines are documents that establish a level of security by enumerating a list of requirements. These requirements specify how a deployment should be configured to avoid insecure

Table 6.1: RFC 2119 requirement levels meaning

Keyword	Meaning
MUST / REQUIRED / SHALL	Absolute requirement
MUST NOT / SHALL NOT	Absolute prohibition
RECOMMENDED / SHOULD	There may exist valid reasons to ignore a particular item
NOT RECOMMENDED / SHOULD NOT	There may exist valid reasons when the particular behavior is acceptable
MAY / OPTIONAL	Truly optional

configurations. Theoretically, each guideline can regulate every configurable element of a webserver, indicating a level of requirement for each element (e.g., for the NIST guideline the TLS version 1.2 `MUST` be supported). The level is assigned by considering published standards, deprecated features, and known attacks.

To comply with a given guideline, a system administrator must examine each element covered by the set of requirements and verify that its value corresponds to the expected value (e.g., if the deployment has to be made compliant with the French guidelines, the Heartbeat TLS extension `SHOULD` be disabled).

We manually reviewed five technical guidelines issued by four states (i.e., US, Germany, France and Italy):

NIST SP 800-52 Rev. 2 The United States' SP 800-52 Rev. 2 [NIS19b], released in August 2019, is the most recent revision of a special publication that was first issued in 2005 [CEFR05]. It recommends the selection and configuration of TLS implementations employing Federal Information Processing Standards [NIS19a] and NIST-recommended cryptographic algorithms. In addition, it specifies its recommendations for *government-only* and *citizen-facing* applications and offers guidance on TLS extensions and certificates with security implications.

It uses specific keywords associated with RFC 2119 [Bra97], a technical standard that defines the meaning of a set of words in the context of IETF documents (see Table 6.1). The RFC defines two absolute requirements that cannot be ignored (i.e. `MUST` and `MUST NOT`), two relative requirements that can be ignored if the context of their use requires it (i.e. `SHOULD` and `SHOULD NOT`), and a fifth requirement that expresses complete optionality;[2]

BSI TR-02102-2 and TR-03116-4 German BSI TR-02102-2 ver.2022-01 [BSI22b] and TR-03116-4 [BSI22a], both of which were last updated in January 2022 and existing since

[2]While RFC 8174 [Lei17] specifies that only all capital words have the meaning specified by RFC 2119, this chapter will indiscriminately use uppercasing and sentence casing.

Table 6.2: TLS protocol compliance across guidelines (excerpt)

	NIST (user-facing)	BSI (user-facing)	ANSSI	AgID
TLS 1.0	Not recommended	Not recommended	Must not	Not recommended
TLS 1.1	Not recommended	Not recommended	Must not	Not recommended
TLS 1.2	Must	Recommended	Optional	Must
TLS 1.3	Recommended	Recommended	Recommended	Must

2020 [BSI20], are two documents that cover roughly the same number of features as SP 800-52, while maintaining the separation between federal and citizen-facing applications. The first document (TR-02102-2) provides general recommendations for the use of TLS cryptographic protocol, while the second (TR-03116-4) narrows them in the case of authentication within federal projects;

ANSSI 1.2 French ANSSI 1.2 [ANS20], published in March 2020, is the second version of the French security recommendations for TLS use (first published in 2016 [ANS16]). ANSSI, unlike US and German guidelines, defines a set of "target" recommendations referred to as Rx. Any deviation from these recommendations will result in a Rx-, a configuration with a reduced level of security.

AgID ver.2020-01 AgID ver.2020-01 [Age20] is the first document covering TLS guidelines issued by the technical agency of the Presidency of the Italian Council of Ministers, published in November 2020.[3] It is intended to provide general guidance on the proper use of the TLS suite and heavily relies on a second set of guidelines, the Mozilla recommendations for server-side TLS v5.6 [Moz20]. In fact, whereas AgID itself only regulates the protocol version and a subset of the available TLS extensions, the remaining features (e.g, ciphers, curves, signatures) are covered by Mozilla guidelines. Mozilla defines three "Profiles" based on the amount of legacy compatibility provided by the webserver: `Modern`, `Intermediate` and `Old`.

Furthermore, Italian guidelines employ a translated set of the RFC 2119-defined keywords, enabling a more precise comparison and definition of the requirement levels.

6.3 Challenges

At the international level, NIST guidelines are regarded as the primary reference standard on the subject of security. In an EU context, one must take into account the possibility that each

[3] We met with AgID in March 2021 to discuss potential inconsistencies in the guidelines. The meeting's outcome will be discussed in Section 7.4.

member state has its own recommendations, in its own language and with potentially different approaches when it comes to their application.

Furthermore, the existence of multiple binding guidelines may result in contradictory requirements when a system administrator attempts to comply with multiple sets of requirements. Observing Table 6.2, for instance, reveals how the acceptance level of TLS 1.1 varies across multiple authorities. In particular, its use may be permitted in the United States (in certain cases) but is strictly prohibited on French websites. What is the process for making a webserver compliant in both states? We must adhere to the requirements levels established by both agencies. The use of TLS 1.1 is considered both an *absolute prohibition* and *acceptable* under certain conditions; which requirement should "win"? And what if we must manage two contradictory requirement levels (e.g., `SHOULD` and `SHOULD NOT`)?

This is not only of academic interest, but may also have implications for the creation of solutions for the entire single market. We examined the state-of-the-art in compliance analysis tools to determine which tools are currently available to assist with addressing these issues, and then we developed a methodology capable of facilitating the implementation of the following use cases:

Single Guideline

- **compare-to-one**: compare an already existing configuration against a *single* guideline. The output consists of a report that highlights the differences between the current and the target configuration and guides the system administrator towards closing the gap;
- **generate-after-one**: generate a working configuration compliant with a *single* guideline;

Multiple Guidelines

- **compare-to-many**: similar to the compare-to-one but considering *multiple* guidelines;
- **generate-after-many**: similar to the generate-after-one but considering *multiple* guidelines.

The functionality of these use cases is contingent upon two conditions: the ability to translate all requirements to a common set of keywords and the capacity to compare different requirements in order to establish an ordering. The first condition can be satisfied by utilizing RFC 2119, the document on which the majority of the guidelines under consideration base their recommendations (see Section 6.2). While NIST and AgID use these keywords, ANSSI and BSI do not, therefore we must find a way to align their recommendations with RFC keywords.

Furthermore, while the definitions for each keyword is exhaustive and unambiguous, the RFC does not define any binary relationship between them, making it more difficult to satisfy the second condition. This necessitates defining an ordering among the available keywords (discussed in Section 6.5.3).

www.example.com:443: FAILED - Not compliant.

- certificate_types: Deployed certificate types are {'rsa'}, should have at least one of {'ecdsa'}.
- certificate_signatures: Deployed certificate signatures are {'sha256WithRSAEncryption'}, should have at least one of {'ecdsa-with-SHA384', 'ecdsa-with-SHA512', 'ecdsa-with-SHA256'}.
- tls_versions: TLS versions {'TLSv1.2'} are supported, but should be rejected.
- ciphers: Cipher suites {'TLS_RSA_WITH_AES_256_CBC_SHA', 'TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256', 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256', 'TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256', 'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384', 'TLS_RSA_WITH_AES_256_GCM_SHA384', 'TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA', 'TLS_RSA_WITH_AES_256_CBC_SHA256', 'TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA', 'TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384', 'TLS_RSA_WITH_AES_128_CBC_SHA', 'TLS_RSA_WITH_AES_128_GCM_SHA256', 'TLS_RSA_WITH_AES_128_CBC_SHA256'} are supported, but should be rejected.

Figure 6.1: sslyze compliance output (excerpt)

6.4 Related Work: Tools for Compliance Analysis

There are numerous online and downloadable tools whose features can assist system administrators in assessing the webserver's compliance. Here are the advantages and disadvantages of each one:

- **sslyze**, already introduced in Section 3.1 is a downloadable tool for determining whether a server employs strong encryption settings and is vulnerable to a subset of known TLS attacks. Since version 5.0.0 (released in November 2021), sslyze can compare a server's configuration (only) to Mozilla's recommended configurations [Moz20]. The output (see Figure 6.1) is a bulleted list containing a simple list of unmet requirements;
- **TLS Profiler** [Fet21] is an analysis tool based on sslyze that can compare the configuration of a server with Mozilla's profiles. While TLS Profiler is based on sslyze, its implementation predates the latter, as its development began in September of 2019. The tool, which can be downloaded or utilized as a web application, generates a list of bullet points highlighting the differences between the current Mozilla profile and the desired profile. The list is comparable to the one shown in Figure 6.1, but each divergent setting is displayed separately from its class; for instance, even though cipher suites are typically configured in bulk, TLS Profiles displays a bullet point for each one. Regardless of the output format difference, this tool shares the same limitations as sslyze;

- **Discovery** is a free online server analyzer [Cry22a] made available by Cryptosense [Cry22b], a company that focuses on secure cryptography deployments. It compares the target webserver against a set of current (i.e. ANSSI [ANS20]) and outdated guidelines (such as NIST SP 800-57 Revision 3 [BBB+12] rather than Revision 5 [Bar20], and ECRYPT 2016 report [ECR16]). Its report is clear and concise and every detected setting is rated on a scale from A to F. By selecting the unmet requirements, the website redirects the user to the technical documentation. This feature can help to pinpoint the issue, but passes the burden of understanding the issue (and related technical documentation) to the system administrator. Moreover, while it does check against multiple guidelines, it does not perform a cross evaluation. This deprives system administrators of a way to understand if the evaluated webserver can, with an appropriate set of changes, become compliant with more than one guideline at the same time;
- **testssl.sh** is a command-line tool already introduced in 3.1. A March 2016 issue on GitHub proposed the addition of a compliance check against NIST SP 800-52. The proposed changes are currently on hold because a proper integration will require a greater number of modifications [Coo21];
- **TLS-Scanner** is a research tool developed by Ruhr-Universität Bochum to assist in evaluating TLS deployments [Ruh22]. With the release of version 4.2.0 in June 2022, TLS-Scanner gained the capability to compare a webserver configuration against NIST and BSI-issued guidelines, SP 800-52r2 and TR02102-2, respectively. At the time of writing, the compliance checks only consist of a counter that shows the number of passed, skipped and failed checks:

```
--|Guideline BSI TR-02102-2
Passed: 18
Skipped: 0
Failed: 6
Uncertain: 0

--|Guideline NIST SP 800-52r2
Passed: 25
Skipped: 6
Failed: 8
Uncertain: 0
```

  Without a way to understand which tests have failed, it is impossible for a system administrator to understand which requirement has not been met and, consequently, how to fix it.

```
Passed: 18
Skipped: 0
Failed: 6
Uncertain: 0
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeR
ichtlinien/TR02102/BSI-TR-02102-2.html
----|Failed Checks:

Grundsätzlich wird empfohlen, nur Cipher-Suiten einzusetzen, die die Anforder
ungen an die Algorithmen und Schlüssellängen der [TR-02102-1] erfüllen.
        The following Cipher Suites were supported but not recommended:
        UNOFFICIAL_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
        TLS_RSA_WITH_AES_128_GCM_SHA256
        TLS_RSA_WITH_AES_256_CBC_SHA256
        TLS_RSA_WITH_AES_128_CBC_SHA256
        TLS_RSA_WITH_AES_256_CBC_SHA
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
Die folgenden Diffie-Hellman Gruppen werden empfohlen.
        The following groups were supported but not recommended:
        ECDH_X25519
        SECP521R1
Die folgenden Diffie-Hellman Gruppen werden empfohlen.
        The following groups were supported but not recommended:
        ECDH_X25519
        SECP521R1
Die folgenden Cipher-Suiten werden empfohlen.
        The following Cipher Suites were supported but not recommended:
        TLS_CHACHA20_POLY1305_SHA256
```

Figure 6.2: TLS-Scanner output fragment

However, it is possible to increase the level of verbosity and receive more information by using the `ALL/DETAIL` reporting levels, although this is not explicitly stated in the readme. Figure 6.2 shows the extended output and much more precise indications of which elements are currently available but must be modified to achieve compliance.

- **Immuniweb SSL Security Test** is an online test suite [Imm22] capable of performing PCI-DSS 3.2.1 (see Section 6.1) and NIST compliance checks, among other security-related analyses. It labels each detected cipher and protocol, indicating whether it is part of a good configuration or a *major compliance issue*. It does not describe how to correct the detected misconfigurations, but it is more comprehensive than the previously described analyzers;

- **Observatory** [Moz22a] is a Mozilla project designed to help system administrators configure deployments securely. Specifically, the "TLS Observatory" tab makes use of the Immuniweb engine and displays a summary of the proximity to a *Compatibility Level*, which is another name for the TLS profiles defined in [Moz20]. However, if the configuration deviates significantly from a Profile, the tool returns "Non-compliant". This makes it impossible to determine how much the detected configuration deviates from the target configuration and, consequently, provides no indication of how to close the gap;

Table 6.3: TLS extensions recommendations across guidelines (excerpt)

	NIST	BSI	ANSSI	AgID
extended_master_secret	Must	Recommended	Recommended	Not mentioned
heartbeat	Not mentioned	Not recommended	Not recommended	Not recommended
post_handshake_auth	Optional	Not mentioned	Optional	Must not
renegotiation_info	Must	Recommended	Recommended	Must
supported_groups	Must	Recommended	Recommended	Not mentioned
truncated_hmac	Optional	Not recommended	Not recommended	Not mentioned

- **SSL Configuration Generator** [Moz22b] developed by Mozilla. It is the only tool capable of generating working configurations for a large number of webservers, taking into account the TLS library version, the selected profile (i.e. `Modern`, `Intermediate` and `Old`), and additional configurable elements. The configuration includes blanks for the system administrator to fill in with platform-specific settings (e.g., certificate path, redirection policies).

While there exist numerous tools capable of performing compliance-related checks, each one is specialized for specific use cases. In addition, they frequently only compare against a single guideline, and the reports frequently lack clarity because they do not specify which changes are required (e.g., `MUST`) and which offer some degree of flexibility (e.g., `RECOMMENDED`). In addition, none of the presented tools cover compliance with the numerous guidelines outlined in the preceding section, which are highly recommended for transnational services.

6.5 Compliance Methodology

In this section, we describe the process of collecting a set of recommendations from multiple agency-issued guidelines, including the mapping that was performed to make them comparable and homogeneous. In addition, we will introduce a methodology that can be used to assess the compliance of a target webserver against a single or multiple guidelines, thereby enabling the implementation of the use cases described in Section 6.3

6.5.1 Recommendations Collection

To map all recommendations extracted from various guidelines into a common set of keywords, we had to analyze each document, manually parse their content (as all of them, with the exception of Mozilla's, are not available in a machine-readable format), and then map it using RFC 2119 keywords.

Table 6.4: Supported groups recommendations across guidelines (excerpt)

	NIST	BSI (federal applications)	ANSSI	Mozilla (modern)
secp256r1 / P-256	Recommended	Must	Recommended	Recommended
secp384r1 / P-384	Recommended	Recommended	Recommended	Recommended
ffdhe2048	Optional	Recommended	Optional	Not recommended
brainpoolP256r1	Not mentioned	Must	Optional	Not recommended
x25519	Not mentioned	Not mentioned	Optional	Recommended

The extraction of recommendations from NIST's and AgID's guidelines was straightforward because their documents already used standard keyword references (see Section 6.2). As the structure of BSI documents is comparable to that of the aforementioned ones, we decided to use the official English translation of TR-02102-2 [BSI22c] and map its RFC-like keywords to the standard ones. Due to its extensive use of adjectives that a non-native speaker cannot map directly, the ANSI guideline was the most challenging to map. To standardize the keyword set for this document type, we used the Discovery's report (see Section 6.4) in conjunction with our previously mapped guidelines to select the most precise keyword. The set of extracted recommendations can be divided into groups regulating the following elements:

- protocols (see Table 6.2);
- cipher suites;
- TLS extensions (see Table 6.3);
- supported groups (see Table 6.4);
- signature algorithms;
- hash algorithms;
- certificate signatures;
- key lengths;
- various certificate details.

The information extracted from the guidelines constitutes our recommendations dataset; for each configurable element, it specifies the level of requirements mandated by each agency-issued guideline. The full set of machine-readable recommendations, compiled by reviewing NIST, BSI, ANSSI, AgID and Mozilla guidelines, is made available in [Man23] for external verification and crowdsourcing. With this material, we overcome one of the major limitations of the guidelines, namely a lack of consistent notation and structuring, and are able to propose a methodology to perform compliance analysis against a single guideline or multiple guidelines.

6.5.2 Single Guideline

Table 6.5: Single Guideline - Compliance evaluation

Requirement level	Compare-to-one		Generate-after-one
	Element detected	Element missing	Include element?
Must / Required / Shall	Do nothing	ERROR: enable	Yes
Must not / Shall not	ERROR: disable	Do nothing	No
Recommended / Should	Do nothing	ALERT: enable	Yes
Not recommended / Should not	ALERT: disable	Do nothing	No
May / Optional	Do nothing	Do nothing	No
Not explicitly mentioned	Guideline-dependent	Do nothing	No

When a single guideline is considered, the act of determining if a deployment is compliant with a given set of requirements (compare-to-one scenario) and the process of generating a compliant configuration (generate-after-one scenario) can be accomplished by scrolling through each requirement and acting according to the requirement level.

In Table 6.5, we propose a methodology defining how to act in these two use cases. The first column indicates the requirement level an agency assigns to a given element, such as the presence of a specific extension or the availability of a given protocol or cipher; the next two columns describe how to behave if a given element is detected or absent, respectively, in the target configuration; and the fourth column describes how to generate a new configuration compliant with a target standard. Consider, as an example, a fragment of the compare-to-one scenario that only takes into account protocol compliance. If a system administrator wishes to make his or her deployment NIST-compliant (see the second column of Table 6.2), and assuming that the webserver already supports both TLS 1.1 and 1.3, our methodology would direct the administrator to:

- disable TLS 1.1 unless there exists a valid reason that requires it to be enabled (i.e. protocol `NOT RECOMMENDED`);
- enable TLS 1.2 as it `MUST` be available;
- keep TLS 1.3 enabled as it is `RECOMMENDED`.

Theoretically, each guideline can regulate every configurable aspect; however, this is not always the case, and it may occur that a particular element is enabled in a configuration but is not governed by the guideline. To address this edge case, we designed a specific behavior that operates as follows: if the element is not explicitly covered, but the document contains a policy for such circumstances, the policy is applied. Otherwise, its availability should be evaluated on an individual basis. The following are the observed policies for non-covered elements:

NIST all cipher suites not explicitly mentioned `MUST NOT` be used;

ANSSI all cipher suites not explicitly mentioned `SHOULD NOT` be used;

MOZILLA all protocols not explicitly mentioned `MUST NOT` be used.[4]

In the generate-after-one scenario, the process of building a configuration can be handled by listing all the `MUST` and `RECOMMENDED` guideline requirements. With NIST as the target guideline, a newly generated configuration file would only enable TLS1.2 and TLS 1.3.

6.5.3 Multiple Guidelines

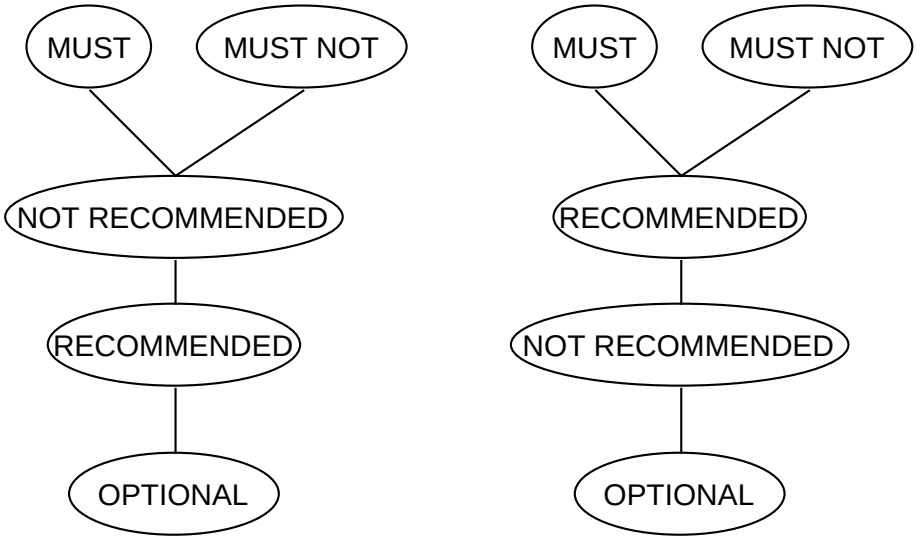
Comparing multiple guidelines is not straightforward because RFC 2119 provides no relationship between the requirements, making it impossible to compare two of them directly without defining a consistent methodology.

With the aim to close this gap and enable the possibility to check for compliance against multiple guidelines (compare-to-many and generate-after-many use cases), we define two partially ordered sets (posets). A poset $P = (S, \prec)$ consists of a set S and a binary relation $\prec$ that orders certain pairs of elements - i.e., a partial order. The partial order relation is a homogeneous relation that is transitive and antisymmetric. A pair of elements $a, b \in P$ is comparable if $a \prec b$ or $b \prec a$. The relation $a \prec b$ can be read as "a precedes b", and similarly $b \succ a$ can be read as "b succeeds a", where $\succ$ is the dual of $\prec$.

Our sets S are composed of the keywords defined by RFC 2119 while the orderings can be schematically summarized as follows:

- **Security wins**
  - `MUST` $\succ$ `RECOMMENDED` $\succ$ `OPTIONAL`
  - `MUST NOT` $\succ$ `NOT RECOMMENDED` $\succ$ `OPTIONAL`
  - **`NOT RECOMMENDED`** $\succ$ **`RECOMMENDED`**
- **Legacy wins**
  - `MUST` $\succ$ `RECOMMENDED` $\succ$ `OPTIONAL`
  - `MUST NOT` $\succ$ `NOT RECOMMENDED` $\succ$ `OPTIONAL`
  - **`RECOMMENDED`** $\succ$ **`NOT RECOMMENDED`**

[4] Policy inferred from [Moz22b].



(a) Security wins.

(b) Legacy wins.

Figure 6.3: Defined partial orders

We decided to define two distinct posets because indiscriminately considering any `RECOMMENDED` requirement to be stronger than a `NOT RECOMMENDED` one (i.e. `RECOMMENDED` $\succ$ `NOT RECOMMENDED`) could result in extremely strict configurations, while doing the opposite (i.e. `NOT RECOMMENDED` $\succ$ `RECOMMENDED`) would result in configurations containing insecure or deprecated features. By developing two options, we enable a configuration to reflect the system administrator's particular preferences. The proposed orders can also be represented by Hasse diagrams (see Figure 6.3), graphical representations in which every comparable element of the set is connected to another using a segment that indicates a relation. Hasse diagrams have an implied upward orientation; therefore, if $x \prec y$, x appears below y in the drawing; for instance, in the Hasse diagram illustrating the *Security wins* approach, `OPTIONAL` appears lower than `RECOMMENDED` because `OPTIONAL`$\prec$`RECOMMENDED`.

In both orders, `OPTIONAL` is the least element as it precedes all other elements. This means that, when compared to any element with a higher position, the `OPTIONAL` recommendation is ignored. Meanwhile, `MUST` and `MUST NOT` are the maximal elements, as they succeed all other elements.

`MUST` and `MUST NOT` are not part of the defined ordering because they are both absolute requirements; as a result, they cannot be objectively compared, and there is no segment connecting them in Figure 6.3. Considering these guidelines, no evaluation of a requirement can result in a `MUST` vs. `MUST NOT` comparison. To make our methodology future-proof and aid in the man-

	Protocol	Requirement		Expected result		
				Security wins	Legacy wins	Order wins
AgID NIST (gov)	TLS 1.0	NOT RECOMMENDED MUST NOT		MUST NOT	MUST NOT	
AgID Mozilla Old	TLS 1.1	NOT RECOMMENDED RECOMMENDED		NOT RECOMMENDED	RECOMMENDED	
AgID Mozilla Mod.	TLS 1.2	MUST NOT RECOMMENDED		MUST	MUST	
ANSSI Custom	TLS 1.3	MUST MUST NOT		→	→	MUST
Custom ANSSI	TLS 1.3	MUST NOT MUST		→	→	MUST NOT
Custom NIST (gov)	TLS 1.0	MUST MUST NOT		→	→	MUST
Custom NIST (gov)	TLS 1.1	MUST NOT RECOMMENDED		MUST	MUST	
ANSSI Custom	TLS 1.2	OPTIONAL MUST NOT		MUST NOT	MUST NOT	
Custom NIST (gov)	TLS 1.3	MUST NOT RECOMMENDED		MUST NOT	MUST NOT	

Figure 6.4: Comparison approaches

agement of such scenarios, we introduce the concept of a *custom* guideline: this can be used to compare an organization's security policy, or to narrow an existing guideline by incorporating additional threat assumptions. To provide a consistent method for managing multiple guidelines while accounting for any possible comparison, we propose the following method:

Order wins the user explicitly chooses which authority should take precedence when comparing their set of recommendations. By doing so, the first authority's decisions on `MUST` and `MUST NOT` requirements directly reflects on the final output, ignoring the second authority choice.

Figure 6.4 illustrates its application for a subset of possible cases. Using TLS 1.3 as an example,

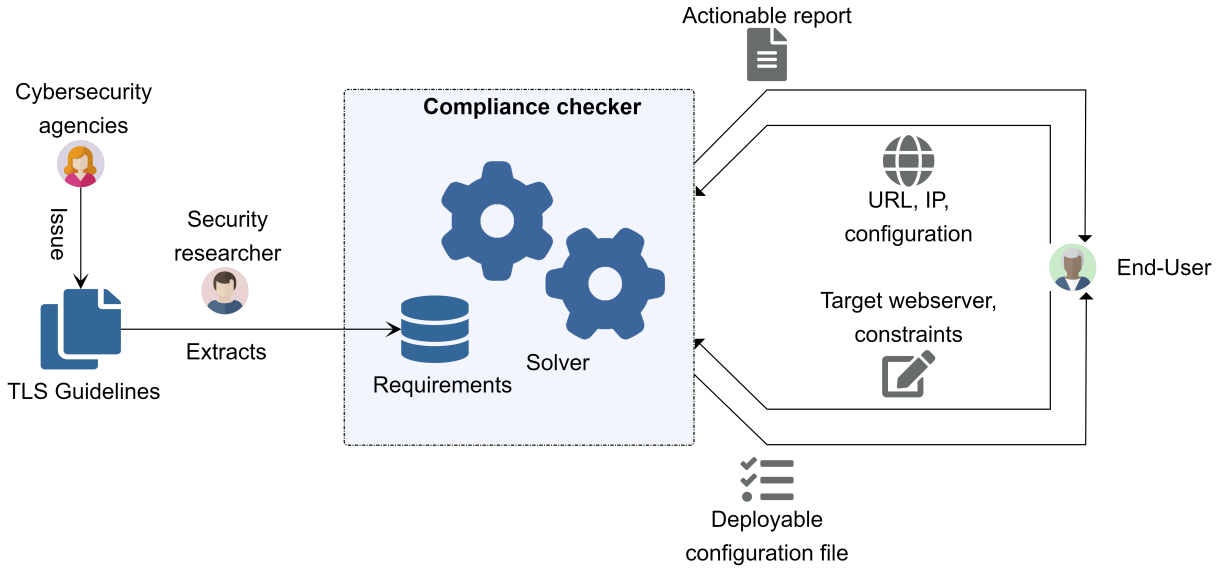


Figure 6.5: Compliance module architecture

its `MUST` requirement level would lose to a `MUST NOT` if it were placed second, but it would win if it were placed first.

In the event that an element is not mentioned in the guideline, we extend the methodology outlined in Section 6.5.2 and treat any missing requirement level as `OPTIONAL`. Thus, the element not covered by the guideline cannot override the defined one, but it is still considered in specific edge cases, such as when a non-covered element appears in a configuration (*compare-to-many* scenario) but neither guideline covers it.

6.6 Prototyping

To facilitate the adoption of the proposed methodology in the real world, we developed multiple prototypes incorporating various technologies and approaches. The goal is to release a compliance-checking module for TLSAssistant v2 that includes an internal database containing formalized set requirements extracted from the reviewed guidelines (see Figure 6.5). On each run, the user can either

- provide an URL or configuration file. The module compares the current state of each configurable element and returns a set of actionable hints to make the webserver compliant (compare-to-one and compare-to-many scenarios);
- specify a target webserver (i.e. Apache or NGINX) and a set of additional constraints.

Table 6.6: SAT solvers evaluation

Solver	Last release	`AllSAT`	`Unsat-core`
Boolector [Vv.22a]	May 2021	No	No
CVC4 [Vv.21]	May 2021	No	Yes
MathSAT [Fon22]	November 2022	Yes	Yes
PicoSAT [Joh16]	January 2016	No	Yes
Yices 2 [SRI22]	October 2021	No	Yes
Z3 [Mic22b]	September 2022	Yes	Yes

The module generates a configuration file that can be used to deploy an out-of-the-box compliant webserver (generate-after-one and generate-after-many scenarios).

The sections that follow will describe the technologies considered when designing the first two prototypes and two reference scenarios demonstrating their potential application.

6.6.1 SAT

SAT solvers were the first technology we attempted to implement when designing our first compliance analysis prototype.

6.6.1.1 Background

A SAT solver is an algorithm capable of determining whether there exists a solution that satisfies a given Boolean formula (i.e., a formula containing binary variables connected by logical relations). Specifically, the solver returns `SAT` (satisfiable) if it discovers a combination of variables that can satisfy the formula, and `UNSAT` (unsatisfiable) if it demonstrates that no such combination exists. If the formula can be satisfied, the solver returns a Model containing the boolean value of each variable that satisfies it. In addition, some SAT solvers offer the `AllSAT` and `Unsat-core` features. The former returns the entire set of Models that satisfy a given formula, whereas the latter, in addition to declaring the overall formula as Unsatisfiable, returns a minimal subset of constraints that, if removed, makes the formula satisfiable.

6.6.1.2 Prototyping Work

We compared the state-of-the-art in terms of SAT solvers in order to find the one that best met our necessities (i.e., availability of the `AllSAT` and `Unsat-core` features as we were not demanding on the performance side). We were especially interested in these two features because

their automation would have facilitated the integration process. In particular, the `AllSAT` output can be used to generate all possible configurations beginning from a given set of constraints (a combination of extracted recommendations and system administrator preferences), providing as output a suggested configuration and possible alternatives (generate-after-one and generate-after-many cases). Instead, the `Unsat-core` output would have automatically identified the configuration features that rendered a webserver non-compliant, enabling us to easily generate an actionable output describing configuration changes (compare-to-one and compare-to-many cases).

pySMT [Vv.22b], the Python library we decided to use as an abstraction layer, supported all of the evaluated solvers. Z3 and MathSAT were the tools who stood out from the comparison (see Table 6.6[5]). We preferred MathSAT because the research unit that created it is located in the same research foundation as us, making it easier to obtain direct support.

However, the use of a SAT solver for compliance presented an unexpected set of difficulties. Initially, the integration of SAT solver in a hybrid scenario (i.e., SAT constraints within a Python script) made debugging difficult, and the absence of adequate documentation for pySMT contributed to the overall slowness of implementation. In addition, the conversion of the method to compare against single guidelines (see Table 6.5) into dynamic boolean formulas proved to be more difficult, verbose and time-consuming than anticipated. Listing 6.1 shows a fragment of the code with lines 8-17 defining the compliance rule (expressed on line 16). In light of these considerations, we decided to move away from SAT solvers in favor of other methods that would provide an equally potent comparison with a shorter development time.

```
1  def tls_requirement(self, scan_results_tls):
2      """AgID requirement 2.1
3      [MUST] Use TLS 1.2 or higher versions
4      [SHOULD] Reject TLS versions prior to 1.2
5      """
6      solver = Solver(name="msat")
7
8      should_not_use_protocols = [
9          Not(self.tls_versions["SSLv2"]),
10         Not(self.tls_versions["SSLv3"]),
11         Not(self.tls_versions["TLSv1.0"]),
12         Not(self.tls_versions["TLSv1.1"]),
13     ]
14
15     must_use_protocols = [self.tls_versions["TLSv1.2"], self.tls_versions["
       TLSv1.3"]]
16     # compliance_rule: (!ssl2 && !ssl3 && !tls0 && !tls1) && (tls2 || tls3)
17     compliance_rule = And(And(*should_not_use_protocols), Or(*
       must_use_protocols))
18
19     solver.add_assertion(compliance_rule)
20     for protocol in scan_results_tls:  # add each detected constraint
```

[5]Since our first comparison on May 2021, CVC4 got succeeded by cvc5 but its evaluation is outside our scope.

```
            solver.add_assertion(Iff(self.tls_versions[protocol], Bool(True)))
        if solver.solve():
            print("Suggested configuration:")
            for protocol in self.tls_versions:
                value = (
                    "DISABLED"
                    if solver.get_value(self.tls_versions[protocol]).is_false()
                    else "ENABLED"
                )
                print(f"\t{protocol} = {value}")
        else:
            print("UNSAT")
```

Listing 6.1: Protocol compliance with a SAT solver

6.6.2 JSON Schema

```
{
    "protocols":[
        "SSLv2",
        "SSLv3",
        "TLSv1.2",
        "TLSv1.3"
    ],
    "ciphers3":[
        "TLS_AES_128_CCM_8_SHA256",
        "TLS_AES_128_GCM_SHA256",
        "TLS_AES_256_GCM_SHA384",
        "TLS_CHACHA20_POLY1305_SHA256"
    ],
    "ciphers2":[
        "ECDHE-ECDSA-AES256-GCM-SHA384",
        "ECDHE-RSA-AES256-GCM-SHA384",
        "ECDHE-ECDSA-CHACHA20-POLY1305"
    ],
    "certificate_signatures":[
        "ECDSA (P-256)"
    ],
    "tls_curves":[
        "X25519"
    ],
    "dh_parameter_size":2048,
    "hsts_max_age":63072000,
    "certificate_lifespan":90,
    "cipher_preference":"client",
    "secure_renegotiation_use":true,
    "legacy_renegotiation":false,
```

```
    "client_initiated_renegotiation": false,
    "post-handshake-authentication": true,
    "compression_use": false,
    "heartbeat-extension-use": false
}
```

Listing 6.2: JSON instance describing a server's configuration

JSON Schema is a declarative language for validating and annotating JSON documents [AWH]. It specifies a format for describing the structure of JSON data, including how a document should appear, how to extract data from it, and how to interact with it. Schemas may be expanded through the definition of additional vocabularies or, less formally, through the definition of additional keywords outside of any vocabulary.

Validation of JSON Schema imposes constraints on the structure of instance data [AWD]. If all of the instance's data satisfy all of the asserted constraints, the instance is valid according to the schema. This validation resembles the evaluation we attempted to replicate using SAT solvers for compliance analysis (see Section 6.6.1). Therefore, we map the required concept to the JSON schema and utilize its functionality to identify "invalid data" (i.e., the set of unmet requirements). Specifically, we created a JSON instance that mimics a webserver configuration (see Listing 6.2) and a schema that describes the structure expected by a specific guideline (see Listing 6.3). Each protocol found in the JSON instance that does not match the list provided on lines 7-14 of the schema will be reported as "invalid data" Consequently, the invalid data set will correspond to the `Unsat-core` exported by a solver. Only MUST NOT requirements are addressed by the schema's current structure. It will therefore require an additional evaluation step to refine its results and account for non-absolute keyword phrases (e.g., `RECOMMENDED`).

The JSON Schema prototyping is currently capable of comparing the provided instance to the AgID specification and generating a report indicating which parts of the webserver configuration must be modified to achieve compliance (compare-to-one scenario). The prototype can also generate a simple list containing the AgID-required elements (generate-after-one scenario). This needs to be converted to webserver-specific language before it can be used as a configuration file.

```
{
    "$id":"https://example.com/person.schema.json",
    "$schema":"https://json-schema.org/draft/2020-12/schema",
    "title":"AgID Intermediate schema",
    "type":"object",
    "properties":{
        "protocols":{
            "type":"array",
            "items":{
                "enum":[
                    "TLSv1.2",
                    "TLSv1.3"
                ]
```

```
            },
            "description":"The list of available protocols."
        },
        "secure_renegotiation_use":{
            "type":"boolean",
            "const":true,
            "description":"Secure renegotiation use."
        },
        "legacy_renegotiation":{
            "type":"boolean",
            "const":false,
            "description":"Legacy renegotiation."
        },
        "client_initiated_renegotiation":{
            "type":"boolean",
            "const":false,
            "description":"Accept client-initiated renegotiation."
        },
        "post-handshake-authentication":{
            "type":"boolean",
            "const":true,
            "description":"Allowing post-handshake authentication."
        },
        "compression_use":{
            "type":"boolean",
            "const":false,
            "description":"Allowing TLS compression."
        },
        "heartbeat-extension-use":{
            "type":"boolean",
            "const":false,
            "description":"Heartbeat extension use."
        }
    }
}
```

Listing 6.3: JSON Schema for AgID guideline (excerpt)

6.6.3 Reference Use Cases

In the following, we present two reference use cases that illustrate how the prototypes can be used to perform a compliance analysis.

6.6.3.1 Local Public Administration

Consider a non-specialist system administrator tasked with deploying a TLS server for a local Italian public administration service, thus advised to meet the AgID recommendations. Without the assistance provided by our methodology, the system administrator will need to manually review the AgID-issued documentation and spend time learning the technical requirements and how to apply them by configuring the server appropriately. This process is time-consuming even for skilled users [MCSR21a] and error-prone, which could slow down the overall creation of a secure and compliant webserver.

Using our module, the system administrator will be able to generate a working configuration from scratch by merely specifying the target guideline and the webserver for which the configuration file is needed. Figure 6.6 depicts a sample of the output of this generate-after-one scenario.

6.6.3.2 European Social Media Platform

To extend the preceding real-world scenario, we can consider a hypothetical social media platform built in Germany, with a pre-existing webserver that is already compliant with BSI guidelines, but with the intention of becoming available in France and compliant with its set of recommendations. The system administrator responsible for the upgrade must become familiar with both BSI and ANSSI guidelines and find a way to make the platform compliant in both states.

Without the methodology, the process will be time-consuming because a German speaker will first need to learn and comprehend a French-written technical document, the structure of which does not conform to RFC 2119 (see Section 6.2). In addition, the system administrator will need to combine the reviewed requirements with those of the previously deployed platform, attempting to strike a balance between both specifications. With our prototype, the system administrator will instead run the script on the target webserver, specifying the desire to make it compliant with both ANSSI and BSI (compare-to-many scenario; see Figure 6.7), and then follow the report to change the configuration with minimal effort and seamless integration.

6.7 Discussion

Due to the nuances within the various documents, the process of making a webserver compliant with standards and guidelines issued by international bodies and cybersecurity agencies can be complex and time-consuming. We assist system administrators in completing this delicate task by:

- manually reviewing five technical guidelines issued by four states (i.e., US, Germany,

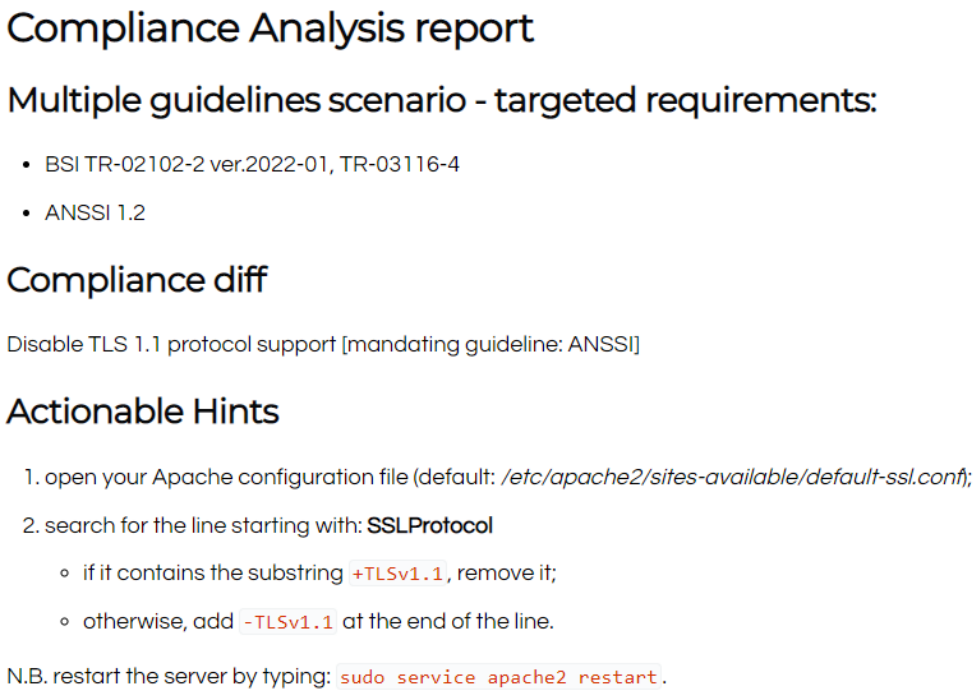
```
SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite          ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
SSLHonorCipherOrder     off

<VirtualHost *:443>
    SSLEngine on
    SSLCertificateFile      /path/to/cert_chain
    SSLCertificateKeyFile   /path/to/private_key
    Header always set Strict-Transport-Security "max-age=63072000"
</VirtualHost>
```

Figure 6.6: generate-after-one Apache output fragment

France and Italy), extracting their recommendations into a machine-readable format, and making them publicly available for external verification and crowdsourcing;

- defining a methodology for analyzing compliance with a single or multiple guidelines. It can be utilized to compare an existing deployment to agency-issued guidelines or to generate a configuration that is compliant-by-design;
- implementing the defined methodology through the creation of multiple prototypes that incorporate various technologies. The prototyping serves as a foundation for the development of a compliance-checking module for TLSAssistant v2 that can assist system administrators with the configuration process.



Compliance Analysis report

Multiple guidelines scenario - targeted requirements:

- BSI TR-02102-2 ver.2022-01, TR-03116-4
- ANSSI 1.2

Compliance diff

Disable TLS 1.1 protocol support [mandating guideline: ANSSI]

Actionable Hints

1. open your Apache configuration file (default: */etc/apache2/sites-available/default-ssl.conf*);
2. search for the line starting with: **SSLProtocol**
   - if it contains the substring `+TLSv1.1`, remove it;
   - otherwise, add `-TLSv1.1` at the end of the line.

N.B. restart the server by typing: `sudo service apache2 restart`.

Figure 6.7: compare-to-many Apache report fragment

Chapter 7

Impact on Collaborations

This chapter will focus on the scientific, technical, and social impact of a series of collaborations that have taken place over the past three years.

7.1 eIDAS Authentication Scheme

In a joint collaboration between FBK and IPZS (acronym for "Istituto Poligrafico e Zecca dello Stato") [Ist], which is the Italian state printing office and mint, in 2018 we have designed and deployed a mobile authentication mechanism that uses the Italian electronic identity card (CIE 3.0 - Carta d'Identitá Elettronica) [Minb]. The card, which is an NFC-readable plastic document, can perform both personal identification and online authentication (exploiting a stored key pair and related X.509 certificate). The designed infrastructure (see Figure 7.1) is composed of two elements managing the authentication process: an Android app, called CieID [Ist], able to interact with the CIE and an identity provider (IdP) that authenticates users combining a challenge-response protocol with SAML 2.0 [Mina]. In the scenario, the user connects to a service provider (SP) using a browser. In order to access the available services, the user needs to be authenticated thus the SP redirects it towards the IdP that will handle the procedure. More specifically, user authentication is performed by establishing a one-way TLS session between the IdP (server) and the mobile application (client). Within this secure channel, the app transmits the user's X.509 together with a message signed with the user's private key to authenticate the user towards the IdP.

Being the use of TLS the basic building block of the solution, any unpatched vulnerability may compromise the entire authentication process. We subsequently performed a security assessment of the implemented solution before its submission for the eIDAS notification [Eic]. The assessment included TLSAssistant's analysis. We discovered that the first release of the infrastructure

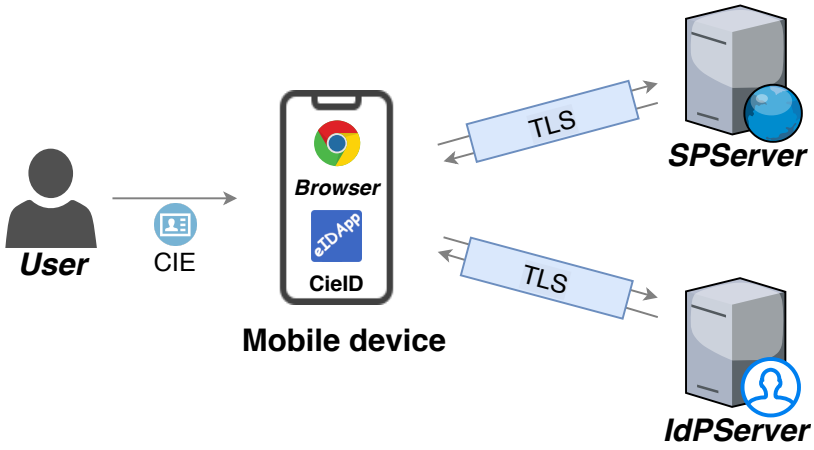


Figure 7.1: CIE ID infrastructure (simplified)

was prone to *Lucky 13*, *3SHAKE* and an incorrect certificate handling on the mobile side. After sharing the generated report with the developers, they promptly patched the two server-side issues, replaced the mobile TLS library with a stronger one and reported back that the TLSAssistant actionable report was both easy to understand and complete.

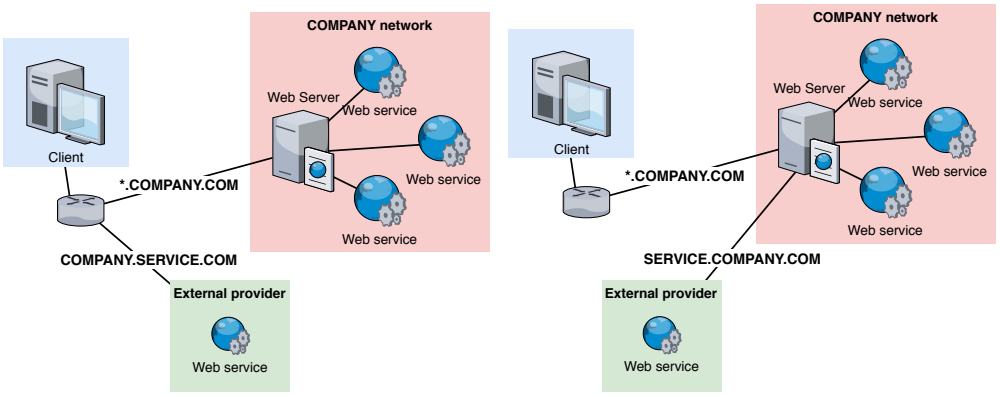
The secured authentication scheme was then submitted and notified by the European Union in September 2019 [Par].

7.2 Sensitive SaaS Configuration

The term *Software as a Service* (SaaS) refers to a cloud distribution model in which the provider offers its services via Internet. By removing the burden to install, maintain and protect the infrastructure from the users' shoulders, these services appeal to big organizations. Being reachable by web browsers, these services (e.g., used to keep track of medical data or whistleblowing) require both data confidentiality and integrity and thus a correct TLS configuration is mandatory.

There are different types of service integration but we focus on two: the first (see Figure 7.2(a)) requires employees to contact an external website to access the company-customized service (e.g., `fbk.SERVICE.com`) while the second (see Figure 7.2(b)) appears within the company domain (e.g., `SERVICE.fbk.eu`) but then redirects the users to a third-party managed server. These approaches come with the same set of benefits and threats but with different responsibilities, to assess this we analyzed two different deployments of the same service used within our organization.

In 2021, we ran TLSAssistant on two Fondazione Bruno Kessler's internal websites. The tool reported that both of them were prone to a series of attacks and, among these, the major threat



(a) Externally-hosted (b) Externally-redirected

Figure 7.2: Examples of SaaS integrations

came from the possibility to mount a Stripping attack. An SSL Stripping attack [Mar] can break the message confidentiality between parties. Even if the attack can only be mounted in the time frame between a browser's first boot and its reception of the HSTS [IET] directive, this is still an exploitable vulnerability especially in the analyzed SaaS (as in both cases, the provided service is only accessed once for specific operations). The suggested mitigation for this threat is the inclusion of the website in the Chrome HSTS preload list [Goo], a set of hostnames hardcoded within web browsers that will always be contacted only via HTTPS.

To perform a responsible disclosure, we shared each report with the respective provider and interacted with them to streamline the mitigation process. In the case of externally hosted services (i.e. `SERVICE.com`), we have seen a quick follow-up from the developers that promptly fixed the vulnerabilities using our mitigations and started the procedure to preload HSTS. After exchanging a few mails with the developers, we also discovered that they already used online scanners to evaluate their TLS soundness. By integrating state-of-the-art TLS analyzers, TLSAssistant has proven to be able to detect and provide appropriate mitigations to a wider range of vulnerabilities when compared to the online scanners used by the third-party company. In the other case (i.e. `SERVICE.fbk.eu`), despite the same mitigations, the situation was different. Specifically, eliminating the SSL Stripping attack was highly counter-intuitive because, being shown under our company's hostname, we needed to request the inclusion within Chrome's list, thus fixing a security flaw caused by an external SaaS.

With a growing interest toward SaaS usage, companies aim for simplicity but this may pave the way to expose their customers to unexpected security flaws and privacy violations because of the lack of a confidential channel. With TLSAssistant we showed that the choice of a cloud service provider and its consequent integration must be done in an accurate manner to preemptively protect the employees from any threat that can be involuntarily inherited.

Time	__name__	instance	vhost	vulnerability	Value
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	freak	0
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	heartbleed	0
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	hsts_preloading	1
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	hsts_set	0
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	https_enforced	1
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	logjam	1
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	lucky13	1
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	mitzvah	0
2022-12-19 17:08:08.251	tls_check	.fbk.eu	.fbk.eu	nomore	0

Figure 7.3: Grafana output for a single host (excerpt)

7.3 Continuous Monitoring of Enterprise Infrastructures

In April 2022, thanks to the results achieved by analyzing a subset of internally deployed SaaS (see Section 7.2), we launched an internal collaboration to incorporate TLSAssistant v2 into the IT monitoring platform of Fondazione Bruno Kessler. After a few technical meetings to comprehend what our tool could provide and the IT infrastructure, we agreed to extend TLSAssistant v2's output as a probe to Prometheus [Auta]. Prometheus is a monitoring system and time series database that can collect metrics from specific targets at specified intervals, display their results, and generate alerts when certain conditions are met. It is used in combination with Grafana, a visualization platform that acts as a dashboard when connected to a monitoring system like Prometheus [Lab].

The IT department suggested that we connect TLSAssistant with an output module capable of printing the following line for each analyzed website:

```
tls_check{vhost=hostname_analyzed,vulnerability=Module_name} VAL
```

followed by the value "1" if the target webserver is vulnerable and "0" otherwise. Due to TLSAssistant v2 modularity, the extension was written quickly and connected to Prometheus without difficulty. A portion of a standard analysis output can be seen in Listing 7.1.

```
tls_check{vhost=REDACTED.fbk.eu,vulnerability=freak} 0
tls_check{vhost=REDACTED.fbk.eu,vulnerability=heartbleed} 0
tls_check{vhost=REDACTED.fbk.eu,vulnerability=hsts_preloading} 1
tls_check{vhost=REDACTED.fbk.eu,vulnerability=hsts_set} 0
tls_check{vhost=REDACTED.fbk.eu,vulnerability=https_enforced} 1
tls_check{vhost=REDACTED.fbk.eu,vulnerability=logjam} 1
tls_check{vhost=REDACTED.fbk.eu,vulnerability=lucky13} 1
tls_check{vhost=REDACTED.fbk.eu,vulnerability=mitzvah} 0
tls_check{vhost=REDACTED.fbk.eu,vulnerability=nomore} 0
```

Listing 7.1: Prometheus output module fragment

The equivalent output can be displayed in Grafana by using a view that lists all the issues detected for a single host (see Figure 7.3) or by summing the results per analyzed host (see Figure 7.4).

7.4 Feedback on Agency-issued TLS Guidelines

AgID ver.2020-01 [Age20] is the first document covering TLS guidelines issued by the Italian public administration, published in November 2020. We reviewed its content immediately after its publication and discovered that it was a hybrid set of recommendations, as some of them refer to Mozilla's recommendations for server-side TLS. One of the suggestions drew our attention because it could have resulted in configuration inconsistencies. Section 3.1 of [Age20] references RFC5746 [RRDO] and states that "a server SHOULD reject a client-initiated session renegotiation". The recommendation did not include Secure Client-Initiated Renegotiation, which was developed in response to CVE-2011-1473 but is not directly mentioned in RFC5746 [MIT19].

After expressing our findings, the authors of the guideline confirmed the existence of ambiguity and stated that its resolution would be published in a revised version of the document.

Time	vhost	Value ↓
2022-12-19 16:53:40.198	.fbk.eu	7
2022-12-19 16:53:40.198	.fbk.eu	7
2022-12-19 16:53:40.198	.fbk.eu	7
2022-12-19 16:53:40.198	.fbk.eu	6
2022-12-19 16:53:40.198	.fbk.eu	6
2022-12-19 16:53:40.198	.fbk.eu	6
2022-12-19 16:53:40.198	.it	5
2022-12-19 16:53:40.198	.it	5
2022-12-19 16:53:40.198	.fbk.eu	5
2022-12-19 16:53:40.198	.fbk.eu	5
2022-12-19 16:53:40.198	.fbk.eu	5
2022-12-19 16:53:40.198	.fbk.eu	5
2022-12-19 16:53:40.198	.fbk.eu	5
2022-12-19 16:53:40.198	.fbk.eu	4
2022-12-19 16:53:40.198	.fbk.eu	4
2022-12-19 16:53:40.198	.fbk.eu	4
2022-12-19 16:53:40.198	.fbk.eu	4

Figure 7.4: Grafana output summed per-host (excerpt)

Chapter 8

Conclusions and Future Work

Managing TLS configurations is a difficult task because the numerous configurable elements have varying and significant effects on system security. System administrators - which are typically not required to have in-depth knowledge of TLS or its vulnerabilities - may not be familiar with all of these components. This issue has been observed in the context of multiple collaborations, demonstrating that it is not merely theoretical. Although numerous tools can detect the same set of vulnerabilities, the manner in which they inform the user does not allow the user to comprehend the repercussions of a potential attack or, more importantly, how to fix the issue in a timely manner.

To assist system administrators and Android developers, we designed TLSAssistant. TLSAssistant is an open source modular framework capable of detecting a large number of vulnerabilities. It is based on the analysis capabilities of several cutting-edge tools. This information can be utilized to generate an actionable report for the user. A report containing a list of vulnerabilities and a guide that instructs the user on how to correct them as quickly and efficiently as possible. TLSAssistant is capable of dynamically loading new modules, performing custom analyses, and integrating with third-party systems due to its modularity. To evaluate its usability and real-world impact, we designed a study in which participants received generic and actionable TLSAssistant output. We observed that mitigation hints help to patch defects in TLS configurations, by reducing the probability of error by 30 times and the time to complete the fix by 3 times.

In addition, we have taken an important first step toward simplifying the process of making webservers compliant with a set of national and supranational guidelines by developing a methodology that can direct users towards a common objective. The methodology can be used to manage an existing deployment or to generate a new configuration that is compliant out of the box.

8.1 Future Work

As we move forward with TLSAssistant v2 as our new baseline, we have several future direction that we want to pursue. On one hand, we plan to further enhance our tool's set of features and on the other we want to extend the research topic covering tangent topic. On the TLSAssistant v2 enhancement side, our main goal is to integrate the proposed compliance-analysis methodology. To aid in identifying potential vulnerabilities and implementing necessary security measures, we also intend to create a "dashboard" feature that can evaluate whether the target webserver is following the best practices on network security, such as those listed in the OWASP cheatsheet [Tea]. This dashboard will provide an easy-to-read overview of the server's security status.

In our efforts to continually enhance the comprehensiveness and effectiveness of the tool, we plan to incorporate mitigations that require a higher level of expertise to implement, such as those that require recompilation or patching of downloadable sources. This will enable end users to patch vulnerabilities that may require more effort. To further improve the correctness of the output, we will integrate additional TLS analyzers with intersecting analyses, reducing the amount of false positives and improving accuracy. Moreover, to ensure that the tool remains up-to-date and effective, we will automatically collect the most recent vulnerabilities and implementation issues by monitoring CVEs, mailing lists, and other similar sources.

Regarding new research directions, we intend to concentrate our efforts on PKI-related issues. With the availability of cross-signed certificates, we plan to investigate how to correctly validate them by examining all possible paths leading up to all available signing roots. Furthermore, we aim to create a user-friendly yet highly customizable TLS flow library that will aid us in testing webservers and TLS libraries for implementation vulnerabilities that could compromise the security of a transmission between two parties. The library extension will also be capable of generating a human-readable message sequence chart flow that can be used to debug implementation issues and expand the educational perspective of our work.

By incorporating these features into our tool, we aim to provide a powerful and comprehensive solution that can assist in the evaluation and enhancement of webserver and Android app security.

Bibliography

[ABC+17] Luca Allodi, Silvio Biagioni, Bruno Crispo, Katsiaryna Labunets, Fabio Massacci, and Wagner Santos. Estimating the assessment difficulty of cvss environmental metrics: An experiment. In Tran Khanh Dang, Roland Wagner, Josef Küng, Nam Thoai, Makoto Takizawa, and Erich J. Neuhold, editors, *Future Data and Security Engineering*, pages 23–39, Cham, 2017. Springer International Publishing.

[ABF+16] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, 2016.

[ACMS20] Luca Allodi, Marco Cremonini, Fabio Massacci, and Woohyun Shim. Measuring the accuracy of software vulnerability assessments: experiments with students and professionals. *Empirical Software Engineering*, 25, 01 2020.

[Age20] Agenzia per l'Italia Digitale. Raccomandazioni agid in merito allo standard transport layer security (tls). `https://cert-agid.gov.it/wp-content/uploads/2020/11/AgID-RACCSECTLS-01.pdf`, November 2020.

[ANS16] ANSSI. Recommandations de sécurité relatives à tls. `https://www.ssi.gouv.fr/uploads/2016/09/guide_tls_v1.1.pdf`, August 2016.

[ANS20] ANSSI. Recommandations de sécurité relatives à tls. `https://www.ssi.gouv.fr/uploads/2020/03/anssi-guide-recommandations_de_securite_relatives_a_tls-v1.2.pdf`, March 2020.

[AP13] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *IEEE Symposium on Security and Privacy, SP*, pages 526–540, 2013.

[Ass18] Association for Computing Machinery. Acm code of ethics and professional conduct. `https://www.acm.org/binaries/content/assets/about/acm-code-of-ethics-booklet.pdf`, 2018.

[Auta] Prometheus Authors. Prometheus - monitoring system and time series database. https://prometheus.io.

[Autb] The Graphviz Authors. Graphviz. https://graphviz.org.

[AWD] Ben Hutton Austin Wright, Henry Andrews and Greg Dennis. Json schema validation: A vocabulary for structural validation of json. https://json-schema.org/draft/2019-09/json-schema-validation.html.

[AWH] Henry Andrews Austin Wright and Ben Hutton. Json schema: A media type for describing json documents. https://json-schema.org/draft/2020-12/json-schema-core.html.

[Bar20] Elaine Barker. Recommendation for key management, part 1: General (revision 5). https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final, 05 2020.

[BBB+12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management, part 1: General (revision 3). https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-3/archive/2012-07-10, July 2012.

[BCC+20] Andrea Bisegna, Roberto Carbone, Mariano Ceccato, Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, Alessandro Tomasi, and Emanuele Viglianisi. 6. automated assistance to the security assessment of api for financial services. Now Publishers, 2020.

[BDLP+15] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Alfredo Pironti, Adam Langley, and Marsh Ray. Transport layer security (tls) session hash and extended master secret extension. https://www.rfc-editor.org/rfc/rfc7627, September 2015.

[BDM+21] Marcus Brinkmann, Christian Dresen, Robert Merget, Damian Poddebniak, Jens Müller, Juraj Somorovsky, Jörg Schwenk, and Sebastian Schinzel. ALPACA: Application layer protocol confusion - analyzing and mitigating cracks in TLS authentication. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 4293–4310. USENIX Association, August 2021.

[Ber] Berlin Group. NextGenPSD2 Access to Account Interoperability Framework - Implementation Guidelines V1.3.4. https://www.berlin-group.org/nextgenpsd2-downloads.

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. 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, CCS '16, page 456–467, New York, NY, USA, 2016. Association for Computing Machinery.

[Ble98] Daniel Bleichenbacher. Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs #1. In Hugo Krawczyk, editor, *Advances in Cryptology — CRYPTO '98*, pages 1–12, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[Boh22] Marko Bohanec. Dexi: A program for multi-attribute decision making. `https://kt.ijs.si/MarkoBohanec/dexi.html`, 2022.

[Bra97] Scott Bradner. Key words for use in rfcs to indicate requirement levels. `https://www.rfc-editor.org/rfc/rfc2119`, March 1997.

[BSA+19] Matthew Bernhard, Jonathan Sharman, Claudia Ziegler Acemyan, Philip Kortum, Dan S. Wallach, and J. Alex Halderman. On the usability of https deployment. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–10, New York, NY, USA, 2019. Association for Computing Machinery.

[BSI20] BSI. Bsi tr-02102 cryptographic mechanisms. `https://web.archive.org/web/20200511084108/https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/tr02102/tr02102_node.html`, May 2020.

[BSI22a] BSI. Bsi-tr-03116-4. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-4.pdf?__blob=publicationFile&v=3`, January 2022.

[BSI22b] BSI. Technical guideline tr-02102-2 cryptographic mechanisms: Recommendations and key lengths. `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=3`, January 2022.

[BSI22c] BSI. Technical guideline tr-02102-2 cryptographic mechanisms: Recommendations and key lengths. `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf?__blob=publicationFile&v=4`, January 2022.

[BSY18] Hanno Böck, Juraj Somorovsky, and Craig Young. Return of Bleichenbacher's oracle threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 817–849, 2018.

[BTPL15] Richard Barnes, Martin Thomson, Alfredo Pironti, and Adam Langley. Deprecating secure sockets layer version 3.0. `https://www.rfc-editor.org/rfc/rfc7568.txt`, June 2015.

[Car01] Michelle Cartwright. Experimentation in software engineering: An introduction. by claes wohlin, per runeson, martin höst, magnus c. ohlsson, björn regnell and anders wesslén. published by kluwer academic publishers, norwell, massachusetts, u.s.a., 1999. isbn: 0-7923-8682-5, 204 pages. price: U.k. ?83.00, u.s.a. $120.00, hard cover. *Software Testing, Verification and Reliability*, 11(3):198–199, 2001.

[CDPF+14] Mariano Ceccato, Massimiliano Di Penta, Paolo Falcarin, Filippo Ricca, Marco Torchiano, and Paolo Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19(4):1040–1074, 2014.

[CEFR05] C M Chernick, III Edington, C, M J Fanto, and R Rosenthal. Guidelines for the selection, configuration, and use of transport layer security (tls) implementations. `https://dx.doi.org/10.6028/NIST.SP.800-52`, 2005.

[CFN+19] Stefano Calzavara, Riccardo Focardi, Matus Nemec, Alvise Rabitti, and Marco Squarcina. Postcards from the post-http world: Amplification of https vulnerabilities in the web ecosystem. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 281–298, 2019.

[CMM+15] Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D Nguyen, and Paolo Tonella. Do automatically generated test cases make debugging easier? an experimental assessment of debugging effectiveness and efficiency. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25(1):1–38, 2015.

[Coh88] J. Cohen. *Statistical power analysis for the behavioral sciences (2nd ed.)*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

[Com15] European Commission. Payment services (psd 2) - directive (eu) 2015/2366. `https://eur-lex.europa.eu/eli/dir/2015/2366/oj`, 2015.

[Com17] European Commission. Commission delegated regulation (eu) 2018/389. `https://eur-lex.europa.eu/eli/reg_del/2018/389/oj`, 2017.

[Com22] European Commission. European commission. `https://ec.europa.eu/info/index_en`, 2022.

[Coo21] David Cooper. Test for conformance to nist sp 800-52. https://github.com/drwetter/testssl.sh/issues/333, 2021.

[Cry] Cryptosense. Discovery. https://discovery.cryptosense.com.

[Cry22a] Cryptosense. Cryptosense discovery. discovery.cryptosense.com/, 2022.

[Cry22b] Cryptosense SA. Cryptosense. https://cryptosense.com/, 2022.

[CS16] Mariano Ceccato and Riccardo Scandariato. Static analysis and penetration testing from the perspective of maintenance teams. In *Proceedings of 10th International Symposium on Empirical Software Engineering and Measurement*, ESEM 2016, pages 25:1–25:6, Ciudad Real, Spain, 2016. ACM.

[DA99] Tim Dierks and Christopher Allen. The tls protocol version 1.0. https://www.rfc-editor.org/rfc/rfc2246.html, January 1999.

[Dat18] IETF Datatracker. Rfc 8446 - the transport layer security (tls) protocol version 1.3. https://datatracker.ietf.org/doc/rfc8446/history/, August 2018.

[Dat22] Datanyze. Web and application servers market share report. https://www.datanyze.com/market-share/web-and-application-servers, 2022.

[Dat23] Datanyze. Openssl market share and competitor report. https://www.datanyze.com/market-share/other-it-infrastructure-software, 2023.

[Des] A. Desnos. Github: Androguard. https://github.com/androguard/androguard.

[Deva] Android Developers. Android keystore system. https://developer.android.com/training/articles/keystore.

[Devb] Android Developers. Cryptography. https://developer.android.com/guide/topics/security/cryptography.

[Devc] Android Developers. Security with network protocols. https://developer.android.com/training/articles/security-ssl.

[Devd] Android Developers. Sslcertificatesocketfactory. https://developer.android.com/reference/android/net/SSLCertificateSocketFactory.

[Dev07] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press; 7 edition, 2007.

[Die14] Tim Dierks. Security standards and name changes in the browser wars. https://tim.dierks.org/2014/05/security-standards-and-name-changes-in.html, 2014.

[Diq22] Alban Diquet. Sslyze. https://github.com/nabla-c0d3/sslyze, 2022.

[Dor] W. Dormann. Announcing CERT Tapioca 2.0 for Network Traffic Analysis. https://insights.sei.cmu.edu/cert/2018/05/announcing-cert-tapioca-20-for-network-traffic-analysis.html.

[DR06] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.1. https://www.rfc-editor.org/rfc/rfc4346.html, April 2006.

[DR08] Tim Dierks and Eric Rescorla. The transport layer security (tls) protocol version 1.2. https://www.rfc-editor.org/rfc/rfc5246.html, August 2008.

[Duc] P. Ducklin. The SLOTH attacks: why laziness about cryptography puts security at risk. https://nakedsecurity.sophos.com/2016/01/08/the-sloth-attacks-why-laziness-about-cryptography-puts-security-at-risk/.

[ECR16] ECRYPT. Algorithms, key size and protocols report. https://www.ecrypt.eu.org/csa/publications.html, 2016.

[Eic] Marie Eichholtzer. Italy - eID. https://ec.europa.eu/cefdigital/wiki/display/EIDCOMMUNITY/Italy+-+eID.

[Eur] European Parliament. Directive (EU) 2015/2366 of the European Parliament and of the Council on payment services in the internal market, amending Directives 2002/65/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L2366&from=EN.

[Fet21] Daniel Fett. Tls profiler. https://github.com/danielfett/tlsprofiler, 2021.

[FHM+12] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgärtner, Bernd Freisleben, and Matthew Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 50–61, 2012.

[FIN] FINSEC. Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures. `https://www.finsec-project.eu/`.

[FIN19a] FINSEC D2.5. *FINSEC Reference Architecture II*, 10 2019.

[FIN19b] FINSEC D3.9. *Finance Sector Security Knowledge Base I*, 10 2019. Due to be updated in Deliverable D3.10 in 2021.

[FIN20] FINSEC D4.5. *Risk Assessment Engine for Critical Infrastructures in the Financial Sector II*, 03 2020. Due to be updated in Deliverable D4.6 in 2021.

[Fon22] Fondazione Bruno Kessler and DISI-University of Trento. Mathsat 5 - an smt solver for formal verification & more. `https://mathsat.fbk.eu`, 2022.

[GHP12] Y. Gluck, N. Harris, and A. Prado. Breach: reviving the crime attack. `https://breachattack.com/`, 2012.

[GIW+18] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 265–281, Baltimore, MD, August 2018. USENIX Association.

[GK05] Robert J. Grissom and John J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2nd edition edition, 2005.

[Goo] Google Open Source. HSTS Preload List. `https://hstspreload.org`.

[Goo22] Google. Hsts list. `https://www.chromium.org/hsts`, 2022.

[Gre] M. Green. Attack of the week: Logjam. `https://blog.cryptographyengineering.com/2015/05/22/attack-of-week-logjam/`.

[Gre11] Matthew Green. A diversion: Beast attack on tls/ssl encryption. `https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/`, 2011.

[GS67] B. G. Glaser and A. L. Strauss. *The Discovery of Grounded Theory*. Aldine, Chicago, 1967.

[HJB12] Jeff Hodges, Collin Jackson, and Adam Barth. Http strict transport security (hsts). https://www.rfc-editor.org/rfc/rfc6797.txt, November 2012.

[HRW00] Martin Höst, Björn Regnell, and Claes Wohlin. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201–214, 2000.

[IET] IETF. HTTP Strict Transport Security (HSTS). https://tools.ietf.org/pdf/rfc6797.pdf.

[IET22] IETF. Ietf — internet engineering task force. https://www.ietf.org, 2022.

[Imm22] Immuniweb. Ssl security test. https://www.immuniweb.com/ssl/, 2022.

[Inc22] Yahoo Inc. Netscape - internet service. https://isp.netscape.com/, 2022.

[ISO18] Iso 9241. ergonomics of human-system interaction — part 11: Usability: Definitions and concepts, 2018.

[Ist] Istituto Poligrafico e Zecca dello Stato S.p.A. CieID. play.google.com/store/apps/details?id=it.ipzs.cieid.

[Jia07] Jiming Jiang. *Linear and generalized linear mixed models and their applications*. Springer Science & Business Media, 2007.

[Joh16] Johannes Kepler Universität Linz. Picosat. http://fmv.jku.at/picosat/, 2016.

[Kar22] Hubert Kariom. Ssl and tls protocol test suite and fuzzer: tlsfuzzer. https://github.com/tlsfuzzer/tlsfuzzer, 2022.

[KBP+19] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. "if https were secure, i wouldn't need 2fa" - end user and administrator mental models of https. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 246–263, 2019.

[KFLS15] Anton Kühberger, Astrid Fritz, Eva Lermer, and Thomas Scherndl. The significance fallacy in inferential statistics. *BMC research notes*, 8(1):84, 2015.

[KMSW17] Katharina Krombholz, Wilfried Mayer, Martin Schmiedecker, and Edgar Weippl. "i have no idea what i'm doing" - on the usability of deploying HTTPS. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1339–1356, Vancouver, BC, August 2017. USENIX Association.

[KRA+18] Platon Kotzias, Abbas Razaghpanah, Johanna Amann, Kenneth G. Paterson, Narseo Vallina-Rodriguez, and Juan Caballero. Coming of age: A longitudinal study of tls deployment. In *Proceedings of the Internet Measurement Conference 2018*, IMC '18, page 415–428, New York, NY, USA, 2018. Association for Computing Machinery.

[Lab] Grafana Labs. Grafana: The open observability platform. `https://grafana.com`.

[Lei17] Barry Leiba. Ambiguity of uppercase vs lowercase in rfc 2119 key words. `https://www.rfc-editor.org/rfc/rfc8174`, May 2017.

[Lin19] LinkedIn Corp. Github: Quick android review kit. `https://github.com/linkedin/qark`, 2019.

[LMP+17] Katsiaryna Labunets, Fabio Massacci, Federica Paci, Sabrina Marczak, and Flávio Moreira de Oliveira. Model comprehension for security risk assessment: an empirical comparison of tabular vs. graphical representations. *Empirical Software Engineering*, 22(6):3017–3056, Feb 2017.

[LMPT13] Katsiaryna Labunets, Fabio Massacci, Federica Paci, and Le Minh Sang Tran. An experimental comparison of two risk-based security methods. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 163–172. IEEE, Oct 2013.

[LRM+19] Frank Li, Lisa Rogers, Arunesh Mathur, Nathan Malkin, and Marshini Chetty. Keepers of the machines: Examining how system administrators manage software updates for multiple machines. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, Santa Clara, CA, August 2019. USENIX Association.

[LSS11] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. Model-driven risk analysis. `https://dx.doi.org/10.1007/978-3-642-12323-8`, 2011.

[Man] Itsik Mantin. Bar mitzvah attack. `https://www.blackhat.com/docs/asia-15/materials/asia-15-Mantin-Bar-Mitzvah-Attack-Breaking-SSL-With-13-Year-Old-RC4-Weakness-wp.pdf`.

[Man23] Salvatore Manfredi. Companion Page: automated assistance for actionable security: Security and compliance of tls configurations. `https://st.fbk.eu/complementary/UNIGE2023_MANFREDI`, 2023.

[Mar] M. Marlinspike. New Tricks For Defeating SSL In Practice. `https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf`.

[MBA+21] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting most-significant-bit-oracles in tls-dh(e). In *Proceedings of the 30th USENIX Security Symposium*, Proceedings of the 30th USENIX Security Symposium, pages 213–230. USENIX Association, 2021.

[MCSR21a] Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. Do security reports meet usability? lessons learned from using actionable mitigations for patching tls misconfigurations. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ARES 21, New York, NY, USA, 2021. Association for Computing Machinery.

[MCSR21b] Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. Replication Package: do security reports meet usability? lessons learned from using actionable mitigations for patching tls misconfigurations. `https://st.fbk.eu/complementary/ETACS2021`, 2021.

[MCSR22] Salvatore Manfredi, Mariano Ceccato, Giada Sciarretta, and Silvio Ranise. Empirical validation on the usability of security reports for patching tls misconfigurations: User- and case-studies on actionable mitigations. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 13(1):56–86, Mar 2022.

[MDK] B. Möller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting The SSL 3.0 Fallback . `https://www.openssl.org/~bodo/ssl-poodle.pdf`.

[MI14] Microsoft-Inria. Triple handshakes considered harmful: Breaking and fixing authentication over tls. `https://www.mitls.org/pages/attacks/3SHAKE`, 2014.

[Mic22a] Microsoft. Microsoft. `https://www.microsoft.com`, 2022.

[Mic22b] Microsoft Research. Z3 theorem prover. `https://github.com/Z3Prover/z3`, 2022.

[Mig] Matt Might. The illustrated guide to a ph.d. `https://matt.might.net/articles/phd-school-in-pictures/`.

[Mina] Ministero Dell'Interno. Accesso ai servizi in rete mediante la CIE 3.0. `https://www.cartaidentita.interno.gov.it/CIE3.0-ManualeSP.pdf`.

[Minb] Ministero dell'Interno. La carta di identità elettronica (cie). `https://www.cartaidentita.interno.gov.it`.

[MIT] MITRE. Common Vulnerabilities and Exposures. https://cve.mitre.org/.

[MIT19] MITRE. Cve-2011-1473. https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2011-1473, 20119.

[MIT09] MITRE. Cve-2009-3555. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555, 2009.

[MIT14a] MITRE. Cve-2014-0224. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0224, 2014.

[MIT14b] MITRE. Cve-2014-3566. https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-3566, 2014.

[MIT15] MITRE. Cve-2015-0204. https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-0204, 2015.

[Moz19] Mozilla. Hsts list. https://wiki.mozilla.org/SecurityEngineering/HTTP_Strict_Transport_Security_(HSTS)_Preload_List, 2019.

[Moz20] Mozilla Foundation. Security/server side tls. https://wiki.mozilla.org/Security/Server_Side_TLS, July 2020.

[Moz22a] Mozilla Foundation. Mozilla observatory. https://observatory.mozilla.org, 2022.

[Moz22b] Mozilla Foundation. Ssl configuration generator. https://ssl-config.mozilla.org, 2022.

[MRS19] Salvatore Manfredi, Silvio Ranise, and Giada Sciarretta. Lost in tls? no more! assisted deployment of secure tls configurations. page 201–220. Springer International Publishing, 2019.

[MRST20] Salvatore Manfredi, Silvio Ranise, Giada Sciarretta, and Alessandro Tomasi. Tl-sassistant goes finsec a security platform integration extending threat intelligence language. In *Cyber-Physical Security for Critical Infrastructures Protection: First International Workshop, CPS4CIP 2020, Guildford, UK, September 18, 2020, Revised Selected Papers*, page 16–30, Berlin, Heidelberg, 2020. Springer-Verlag.

[NDG+19] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. "if you want, i can store the encrypted password": A password-storage field study with freelance developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery.

[NDT+17] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. Why do developers get password storage wrong? a qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 311–328, New York, NY, USA, 2017. Association for Computing Machinery.

[Net] Netscape Communications. The Secure Sockets Layer (SSL) Protocol Version 3.0. https://tools.ietf.org/pdf/rfc6101.pdf.

[NIS12] NIST. Cve-2012-4929. https://nvd.nist.gov/vuln/detail/CVE-2012-4929, 2012.

[NIS19a] NIST. Compliance faqs: Federal information processing standards (fips). https://www.nist.gov/standardsgov/compliance-faqs-federal-information-processing-standards-fips, November 2019.

[NIS19b] NIST. Guidelines for the selection, configuration, and use of transport layer security (tls) implementations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf, August 2019.

[NIS22] NIST. Cybersecurity framework. https://www.nist.gov/cyberframework, 2022.

[Now17] NowSecure. Fully validate ssl/tls. https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/fully-validate-ssl-tls.html, 2017.

[OAS23] OASIS Open. STIX - A structured language for cyber threat intelligence. https://oasis-open.github.io/cti-documentation/stix/intro, 2023.

[Ope] OpenID Foundation. *Financial-grade API (FAPI)*.

[Ope22] OpenSSL. Changelog. https://www.openssl.org/news/changelog.html, 2022.

[Pal] Pallets. Welcome to flask. https://flask.palletsprojects.com/en/2.2.x/.

[Par] European Parliament. Document 52019xc0913(02). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.C_.2019.309.01.0009.01.ENG&toc=OJ:C:2019:309:TOC.

[PCI18] PCI. Recommendation for key management, part 1: General (revision 3). https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI_DSS_v3-2-1.pdf, may 2018.

[PCI22] PCI Security Standards Council. Document library. https://www.pcisecuritystandards.org/document_library, 2022.

[Por] Thomas Pornin. What is DROWN and how does it work? https://security.stackexchange.com/a/116140/186367.

[PRE] PRETA Open Banking Europe. Security and Identification Standards for APIs & Communications. https://www.openbankingeurope.eu/media/1398/oasis-obe-api-identification-and-security-standards-for-apis-and-communications.pdf.

[Pre22] Presidenza del Consiglio dei Ministri. Agid - agenzia per l'italia digitale. https://www.agid.gov.it, 2022.

[PSSC22] LLC. PCI Security Standards Council. Official pci security standards council site. https://www.pcisecuritystandards.org, 2022.

[Qua22a] Qualys. Ssl pulse. https://www.ssllabs.com/ssl-pulse/, 2022.

[Qua22b] Qualys. Ssl server test. https://www.ssllabs.com/ssltest/, 2022.

[rbs17] rbsec. sslscan. https://github.com/rbsec/sslscan/releases/tag/1.11.11-rbsec, 2017.

[Res18] Eric Rescorla. The transport layer security (tls) protocol version 1.3. https://www.rfc-editor.org/rfc/rfc8446, August 2018.

[Ris22] Ivan Ristić. Ssl/tls and pki history. https://www.feistyduck.com/ssl-tls-and-pki-history/, 2022.

[RMSR22a] Matteo Rizzi, Salvatore Manfredi, Giada Sciarretta, and Silvio Ranise. Demo: Tlsassistant v2: A modular and extensible framework for securing tls. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies*, SACMAT '22, page 271–272, New York, NY, USA, 2022. Association for Computing Machinery.

[RMSR22b] Matteo Rizzi, Salvatore Manfredi, Giada Sciarretta, and Silvio Ranise. A modular and extensible framework for securing TLS. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy*. ACM, apr 2022.

[RRDO] Eric Rescorla, Marsh Ray, Steve Dispensa, and Nasko Oskov. Transport Layer Security (TLS) Renegotiation Indication Extension. https://www.rfc-editor.org/rfc/rfc5746.

[Ruh22] Hackmanit GmbH Ruhr University Bochum, Paderborn University. Tls-scanner. https://github.com/tls-attacker/TLS-Scanner, 2022.

[SAW08] Mikael Svahnberg, Aybüke Aurum, and Claes Wohlin. Using students as subjects - an empirical evaluation. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, page 288–290, New York, NY, USA, 2008. Association for Computing Machinery.

[SC90] A. Strauss and J. Corbin. *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage, London, 1990.

[Sch] B. Schneier. Attack Trees. https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

[Sch99] Bruce Schneier. Attack trees. https://www.schneier.com/academic/archives/1999/12/attack_trees.html, 1999.

[Sec] Security & Trust Research Unit. Tlsassistant. https://github.com/stfbk/tlsassistant.

[Ser22] Amazon Web Services. Alexa top sites. https://aws.amazon.com/alexa-top-sites/, 2022.

[She07] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures (4th Ed.)*. Chapman & All, 2007.

[Sho22a] Shodan Search Engine. ssl.version:sslv3. https://www.shodan.io/search?query=ssl.version%3Asslv3, 2022. Accessed on 15.12.2022.

[Sho22b] Shodan Search Engine. vuln:cve-2014-0160. https://www.shodan.io/search?query=vuln%3ACVE-2014-0160, 2022. Accessed on 15.12.2022.

[Sho22c] Shodan Search Engine. vuln:cve-2016-2183. https://www.shodan.io/search?query=vuln%3ACVE-2016-2183, 2022. Accessed on 15.12.2022.

[SKvW+14] Benjamin Saefken, Thomas Kneib, Clara-Sophie van Waveren, Sonja Greven, et al. A unifying approach to the estimation of the conditional akaike information in generalized linear mixed models. *Electronic Journal of Statistics*, 8(1):201–225, 2014.

[SM16] Janet M. Six and Ritch Macefield. How to determine the right number of participants for usability studies. https://www.uxmatters.com/mt/archives/2016/01/how-to-determine-the-right-number-of-participants-for-usability-studies.php, 2016.

[SM17] N. Samarasinghe and M. Mannan. Short paper: TLS ecosystems in networked devices vs. web servers. In *Financial Cryptography and Data Security - 21st International Conference, FC 2017*, pages 533–541, 2017.

[SMJ15] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. Are students representatives of professionals in software engineering experiments? ICSE '15, page 666–676, Florence, Italy, 2015. IEEE Press.

[Sol] Ask Solem. Celery - distributed task queue. https://docs.celeryq.dev/en/stable/.

[Squ22] Square. Okhttp. https://square.github.io/okhttp/, 2022.

[SRI22] SRI International's Computer Science Laboratory. The yices smt solver. https://github.com/SRI-CSL/yices2, 2022.

[SUP18] SUPERAndroidAnalyzer. Github: Secure, unified, powerful and extensible rust android analyzer. https://github.com/SUPERAndroidAnalyzer/super, 2018.

[SWJ13] Riccardo Scandariato, James Walden, and Wouter Joosen. Static analysis versus penetration testing: A controlled experiment. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, Nov 2013.

[Syn14] Synopsys, Inc. The heartbleed bug. https://heartbleed.com, 2014.

[Tea] OWASP CheatSheets Series Team. Transport layer protection cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html.

[Tea19] Worldpay Editorial Team. What's the history of pci dss? https://www.fisglobal.com/en-gb/insights/merchant-solutions-worldpay/article/pci-dss-history-everything-you-need-to-know, 2019.

[THKvZ20a] Christian Tiefenau, Maximilian Häring, Katharina Krombholz, and Emanuel von Zezschwitz. Security, availability, and multiple information sources: Exploring update behavior of system administrators. In *SOUPS @ USENIX Security Symposium*, 2020.

[THKvZ20b] Christian Tiefenau, Maximilian Häring, Katharina Krombholz, and Emanuel von Zezschwitz. Security, availability, and multiple information sources: Exploring update behavior of system administrators. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 239–258. USENIX Association, August 2020.

[uEG+18] Aleš Černivec, Gencer Erdogan, Alejandra Gonzalez, Atle Refsdal, and Antonio Alvarez Romero. Employing graphical risk models to facilitate cyber-risk monitoring - the WISER approach. In *Graphical Models for Security (GraMSec) 2017*, volume 10744 of *LNCS*, pages 127–146, 2018.

[U.S22] U.S. Department of Commerce. National institute of standards and technology. `https://www.nist.gov/`, 2022.

[utd15] utds3lab. Github: Smv-hunter. `https://github.com/utds3lab/SMVHunter`, 2015.

[VP] M. Vanhoef and F. Piessens. RC4 NOMORE (Numerous Occurrence MOnitoring & Recovery Exploit). `https://www.rc4nomore.com/`.

[Vv.21] Vv.Aa. Github: Cvc4. `https://cvc4.github.io`, 2021.

[Vv.22a] Vv.Aa. Github: boolector. `https://github.com/Boolector/boolector`, 2022.

[Vv.22b] Vv.Aa. Github: pysmt. `https://github.com/pysmt/pysmt`, 2022.

[Wet22] Dirk Wetter. /bin/bash based ssl/tls tester: testssl.sh. `https://testssl.sh`, 2022.

[WMY18] L. Waked, M. Mannan, and A. M. Youssef. The sorry state of TLS security in enterprise interception appliances. *CoRR*, abs/1809.08729, 2018.

[WSF] Wsf15 - the 2015 workshop on security frameworks. `https://www.dmi.unict.it/giamp/wsf/15edition.php`.

[You] C. Young. TLS Extended Master Secret Extension: Fixing a Hole in TLS. `https://www.tripwire.com/state-of-security/security-data-protection/security-hardening/tls-extended-master-secret-extension-fixing-a-hole-in-tls/`.

Appendix A

Survey Questionnaires' Content

Table A.1 shows the three parts of the survey questionnaire.

Table A.1: Questions in the survey questionnaire

#	Question	Answer
First part		
Lab 1		
1	I had enough time to perform the task	[1-5] Likert scale
2	I experienced no difficulty in patching the vulnerability given the report	[1-5] Likert scale
3	How much time (in terms of percentage) did you spend looking at the TLS configuration code	[≤≥a] Likert scale
4	How much time (in terms of percentage) did you spend looking at online documentation on TLS vulnerabilities	[≤≥a] Likert scale
5	Provide some examples of online queries you used to search the vulnerabilities online (e.g. keywords used)	[None]/free text
6	Which steps did you take to perform the tasks? (e.g. run command Y, opened file X, ..)	[None]/free text
Lab 2		
1	I had enough time to perform the task	[1-5] Likert scale
2	I experienced no difficulty in patching the vulnerability given the report	[1-5] Likert scale
3	How much time (in terms of percentage) did you spend looking at the TLS configuration code	[≤≥a] Likert scale
4	How much time (in terms of percentage) did you spend looking at online documentation on TLS vulnerabilities	[≤≥a] Likert scale

Continued on next page

Table A.1 – *Continued from previous page*

#	Question	Answer
5	Provide some examples of online queries you used to search the vulnerabilities online (e.g. keywords used)	[None]/free text
6	Which steps did you take to perform the tasks? (e.g. run command Y, opened file X, ..)	[None]/free text
Second part		
1	Which report did you find more useful?	Lab1/Lab2
2	Which report did you find more easy to read?	Lab1/Lab2
3	Which report did you find more complex to understand?	Lab1/Lab2
4	The textual description of the mitigation is useful to complete the tasks	[1-5] Likert scale
5	The code snippet is useful to complete the tasks	[1-5] Likert scale
6	How did you use the code snippet?	[None]/free text
Third part		
1	Would you use it for your work?	[None]/free text
2	Motivate your answer (to the previous question)	[None]/free text
3	Do you know any tool that performs similar tasks?	[None]/free text
4	Do you have any suggestion related to the tool usage?	[None]/free text
5	Do you have any suggestion related to the amount of information provided by the tool's report (Report.md)?	[None]/free text

Appendix B

Leaves Content of TLS Attack Trees

B.1 Break Confidentiality

The following table depicts a detailed view of the leaves shown in Figure 3.5.

Table B.1: Break Confidentiality - Full Leaves

Leaf	Prerequisites	Attack Steps	Final iteration
DROWN	- RSA-generated session key AND - SSLv2 available on the target server OR - Same private key used by a parallel server that supports SSLv2	1. Sniff packets autonomously generated by the victim OR 1. Sniff packets generated through a controlled JavaScript (within the victim's browser) 2. Repeatedly initiate SSLv2 connections using (export-grade) RSA and a peculiar ClientMasterKey 3. Bruteforce server's responses to leak information 4. Exploit the server's response to recover the MasterSecret 5. Use the MasterSecret to decrypt the transmission	

Continued on next page

Table B.1 – *Continued from previous page*

Leaf	**Prerequisites**	**Attack Steps**	**Final iteration**
ROBOT	RSA key exchange negotiated	1. Sniff the victim's transmissions (ClientKeyExchange included) 2. Repeatedly generate and send ClientKeyExchange messages with different padding 3. Check if the server accepted the request (padding correctly guessed)	Use the guessed pre-master secret to generate the shared key
3SHAKE	- Renegotiation available - Resumption available - RSA key exchange negotiated	1. Wait for a client-initiated handshake 2. Connect to the server (sharing the parameters, different tls-unique) 3. Wait for a session resumption by the client 4. Resume the connection to the server (same tls-unique) 5. Optionally inject a malicious request 6. Trigger the server to request a client certificate 7. Forward client's handshake messages to the server 8. Snoop the connection exploiting the same origin policy	

B.2 Break Authentication

The following table depicts a detailed view of the leaves shown in Figure 3.7.

Table B.2: Break Authentication - Full Leaves

Leaf	Prerequisites	Attack Steps	Final iteration
BREACH	HTTP compression enabled	1. Gain JavaScript control on the victim's browser 2. Inject different characters into the client's messages	Retrieve parts of the cookie by analyzing the response size
CRIME	TLS compression enabled	1. Gain JavaScript control on the victim's browser 2. Inject different characters into the client's messages	Retrieve parts of the cookie by analyzing the response size
Lucky 13	- CBC mode cipher negotiated - HMAC-SHA1 negotiated	1. Gain JavaScript control on the victim's browser 2. Tweak and truncate the encrypted packets 3. Analyze the time the server needs to detect the error	Retrieve parts of the cookie by measuring the response delay over every iteration
Sweet32	3DES cipher negotiated (64-bit blocks)	1. Gain JavaScript control on the victim's browser 2. Repeatedly query the server (2^{32} requests)	Check for collisions with a known block
Bar Mitzvah	RC4 cipher negotiated	1. Sniff the victim's transmissions 2. Detect the use of a weak key (invariance weakness) 3. Predict the LSBs of the keystream 4. Try to decrypt the related plaintext 5. Use the cookie to impersonate the client	
RC4 NOMORE	RC4 cipher negotiated	1. Gain JavaScript control on the victim's browser 2. Force the client to contact the server 3. Capture the encrypted requests 4. Calculate the candidate tokens (using the Fluhrer-McGrew biases)	Check which token is the correct one
CA impairment		1. Acquire the CA's signing key 2. Sign a fake certificate 3. Impersonate the server	

Continued on next page

Table B.2 – *Continued from previous page*

Leaf	Prerequisites	Attack Steps	Final iteration
Certificate spoofing		1. Generate a fake self-signed certificate 2. Impersonate the server	
Truncation attack	- Ongoing parallel connections - Incorrect handling of the termination protocol (server side) - Victim using a shared terminal	1. Identify and drop the client's logout request 2. Exploit the connection kept open using the shared terminal	
POODLE	- SSLv3 available - CBC mode ciphers available	1. Force SSLv3 with a CBC cipher (via MITM) 2. Repeatedly replace the ciphertext's padding 3. Send the crafted requests 4. Check if the server accepted the request (padding correctly guessed) 5. Rebuild the cookie two bytes at time	
SLOTH	- RSA-MD5 certificate signature supported - Client uses the same certificate with multiple servers (one malicious)	1. Receive the victim's ClientHello 2. Perform the handshake until the key exchange 3. Compute a chosen-prefix collision of two strings 4. Send a crafted hello to the server 5. Perform the handshake until the key exchange 6. Send a crafted certificate request to the client (containing the other strings) [the hashes will now coincide] 7. Forward the server's HelloDone to the client 8. Forward the client's messages to the server 9. Impersonate the client	