Lost in TLS? No More! Assisted Deployment of Secure TLS Configurations

Salvatore Manfredi[1,2](✉), Silvio Ranise[1], and Giada Sciarretta[1]

[1] Security & Trust, FBK, Trento, Italy
{smanfredi,ranise,giada.sciarretta}@fbk.eu
[2] University of Trento, Trento, Italy

Abstract. Over the last few years, there has been an almost exponential growth of TLS popularity and usage, especially among applications that deal with sensitive data. However, even with this widespread use, TLS remains for many system administrators a complex subject. The main reason is that they do not have the time to understand all the cryptographic algorithms and features used in a TLS suite and their relative weaknesses. For these reasons, many different tools have been developed to verify TLS implementations. However, they usually analyze the TLS configuration and provide a list of possible attacks, without specifying their mitigations. In this paper, we present TLSAssistant, a fully-featured tool that combines state-of-the-art TLS analyzers with a report system that suggests appropriate mitigations and shows the full set of viable attacks.

Keywords: TLS misconfiguration · Vulnerability detection · Assisted mitigations

1 Introduction

Transport Layer Security (TLS) consists of a set of cryptographic protocols designed to provide secure communications over a network. Developed as a successor of the Secure Socket Layer (SSL) protocol, TLS has gained popularity and widespread usage since the release of its first version in 1999 [21]. According to [37], more than 130,000 of the top Alexa websites [43] support one or multiple versions of the TLS protocol.

The popularity of TLS has encouraged attackers to find vulnerabilities and develop exploits as documented by a long line of reported attacks and corresponding fixes [1–5,16,17,28,31–33,42,46,48] together with the evolution of the standard TLS specification from 1.0 to 1.3 as a result of the strategies put in place by Internet service providers such as Apple, Google, Amazon, and Mozilla to deprecate the use of TLS versions 1.0 and 1.1 [7] and of the SHA-1 hash function [6]. The types of attacks vary widely and include the renegotiation of cipher

© IFIP International Federation for Information Processing 2019
Published by Springer Nature Switzerland AG 2019
S. N. Foley (Ed.): DBSec 2019, LNCS 11559, pp. 201–220, 2019.
https://doi.org/10.1007/978-3-030-22479-0_11

suites to exploit weak encryption algorithms [31], the knowledge of initialization vectors to retrieve symmetric keys [17], and the use of libraries to exploit poor certificate validation in deployments where clients are non-browsers [15]. The variety attacks is the result of *(i)* maintaining backward compatibility and *(ii)* evolving use case scenarios in which TLS is deployed. The main problem with *(i)* is illustrated by the following observation from [37]: more than 108,000 web sites still support TLS 1.0 that is vulnerable to a set of well-known attacks including Man In The Middle (MITM). The problem with *(ii)* has already been pointed out in [15] for SSL where it is shown that certificate validation—as supported by available libraries for developing clients not based on browsers (e.g., native mobile applications)—is flawed and permits to mount MITM attacks.

To help administrators in deploying secure TLS instances, a variety of tools [11,14,29,39,45,47,50,51] have been developed for identifying weaknesses that may lead to one or more known attacks. While such tools are quite effective in automatically finding vulnerabilities and issuing warning about possible attacks, the burden of finding adequate mitigation measures is completely left on the administrator who must first collect information about the identified problem and related fixes. Typically, such information is distributed in several sources ranging from scientific papers to blog posts. Even disregarding the effort to collect enough material to mitigate a security problem—notice that available tools have varying coverage of the known TLS attacks—administrators should have enough skills to understand the (often subtle) details and turn the information in a concrete strategy to fix the problem. In other words, there is a problem in making actionable the reports returned by available tools. To overcome this problem, we make the following four main contributions:

- we build an exhaustive catalogue of known attacks to TLS deployments;
- we perform a comparison of the state-of-the-art tools capable of identifying attacks of TLS deployments and characterize the coverage with respect to the catalogue compiled in the previous point;
- we design and build an open-source tool, called TLSAssistant[1], that reuses some of the tools for identifying attacks considered in the previous point to maximize coverage and enriches reports with possible mitigations and fixes, including code snippets when the TLS entities are among the most widely used (e.g., the TLS server is Apache);
- we experimentally evaluate the effectiveness of TLSAssistant by reporting our experience in using it in the context of the deployment of a large scale infrastructure for identity management and, most importantly, in a user study involving users with little or no security skills. The findings of the user study provide encouraging first evidence of the effectiveness of the tool as even un-experienced users were able to successfully mitigate complex attacks.

While many of the problems reported by TLSAssistant are server-side, particular attention has been devoted to the security issues that result from inadequate

[1] Available at sites.google.com/fbk.eu/tlsassistant.

certificate validation in mobile applications. The main motivation for this choice is the increasing role of mobile applications in accessing Internet resources combined with the serious security consequences of managing certificates without the help of browsers (as already observed in [15]).

Plan of the paper. Section 2 provides the necessary background notions on TLS and a brief overview of its vulnerabilities. Section 3 contains a comparison of the state-of-the-art automated tools for identifying TLS attacks both server and (mobile) client-side. Section 4 contains a comprehensive catalogue of attacks and related mitigations in the various version of TLS. Section 5 introduces the tool TLSAssistant with its architecture, usage, and some details about the implementation. Section 6 reports the use of TLSAssistant in the deployment of an eIDAS solution based on the new Italian identity cards and a user study involving bachelor and master degree students that were asked to fix two non-trivial vulnerabilities. Section 7 concludes the paper and highlights future work.

2 Background

We provide some background notions about TLS needed to better understand the security implications of its use. We briefly describe the general structure of the TLS suite and some details in Sect. 2.1 and then give a concise guide to the main vulnerabilities in Sect. 2.2.

2.1 TLS

The Transport Layer Security (TLS) suite is composed of two main protocols:

Handshake: allows the parties to exchange all information required to establish a reliable session. Depending on the configuration, the handshake can provide either mutual or one-way authentication (usually is one-way, thus only server provides a certificate). The protocol supports two special messages: *(i)* `Change Cipher Spec` that signals the transition of the session to a different ciphering strategy and *(ii)* `Alert` which propagates potential alert messages.

Record: encapsulates the messages to be transmitted to ensure their security. The record protocol is composed of the following steps: *(i)* splitting of the data stream into chunks; *(ii)* compression of the chunks; *(iii)* generation of the Message Authentication Code (MAC) with the algorithm agreed during the handshake; *(iv)* encryption of the payload using the cipher chosen during the handshake; and *(v)* addition of the header to enable the packet to be transmitted.

Over the years, TLS has seen the release of four versions: v1.0 released in 1999 [21], v1.1 released in 2006 [22], v1.2 released in 2008 [23], v1.3 released in 2018 (August) [26]. In the following we will detail v1.2, as is the most widely supported, and v1.3, as it has introduced a set of changes to mitigate known vulnerabilities that affected the previous TLS versions.

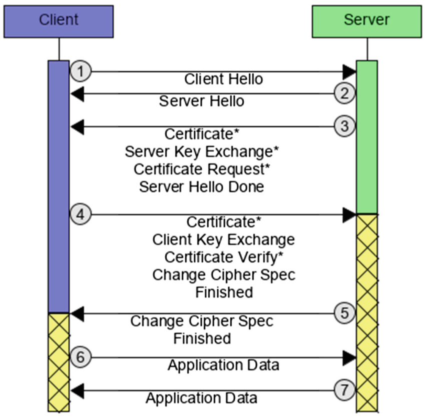


Fig. 1. TLS 1.2 full handshake.

TLS 1.2. According to Qualys' March monthly scan [37], TLS 1.2 is currently the most widely supported protocol with a coverage of 94.8%. Each message of the full handshake is shown in Fig. 1 (the striped sections show the encrypted transmission, asterisks indicate optional messages). For lack of space, we refer to the TLS specification [23] for the description of all the messages. Here, we specify the first two messages: *Client Hello* and *Server Hello*. As we will describe in Sect. 5, our tool analyzer will send different *Client Hello* messages and analyze the *Server Hello* answers to check the presence of vulnerabilities. The remaining messages are used to authenticate the parties, calculate the symmetric key and to apply the ciphering strategies.

Client Hello. The client can start the handshake at any time by sending a *Client Hello* message to the server, it contains: *(i)* the version of the protocol that the client wants to use (it should be the highest available); *(ii)* a random value obtained by chaining the timestamp (32 bit in UNIX time) and a randomly generated nonce (28 bytes); *(iii)* a session identifier: empty field indicating the will to create a new session (in case of session resumption the client behaves differently); *(iv)* list of supported cipher suites, each element has the following structure: `TLS_⟨KeyExchange⟩_WITH_⟨Cipher⟩_⟨Mac⟩`; *(v)* a list of supported compression methods; and *(vi)* a list of requested extensions (set of additional functionalities the server has to provide).

Server Hello. In response to the *Client Hello*, the server sends its hello message that contains: *(i)* a chosen protocol (the highest version supported by both parties); *(ii)* a random value obtained by chaining the timestamp (32 bit in UNIX time) and a randomly generated nonce (28 bytes); *(iii)* a freshly-generated value that will identify the new session (in case of session resumption, the server behaves differently); *(iv)* a chosen cipher suite; *(v)* a chosen compression method; and *(vi)* a list of required extensions (additional features).

Table 1. Main differences introduced with TLS 1.3

Status	What	Why
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

TLS 1.3. After years in the making, the final version of the standard has been published this August, 2018. Table 1 summarizes the key differences with TLS 1.2 according to the RFC [26]. Thanks to these changes, TLS 1.3 is not prone to any of the known attacks such as the ones related to legacy ciphers (e.g., Lucky 13) or broken hash algorithms (e.g., SLOTH).

2.2 Vulnerabilities

There exist many TLS-related vulnerabilities: some of them exploit the support of weak cryptographic aspects (e.g., weak ciphers and hash functions), others use an (un)voluntary weakening of security properties to bypass the authentication process (e.g., accepting self-signed certificates [34] and setting a permissive hostname verifier [34]) or the loss of trust in the PKI system due to a improper certificate generation (e.g., CA impairment [19] and Certificate Spoofing [25]). In Table 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; *(iii)* a brief description on how the attack can be mounted, and *(iv)* which version of TLS can be affected by such attack. To better understand the described vulnerability exploitation, we review some cryptographic aspects:

- **Export ciphers.** Weakened ciphers introduced by the U.S. government to limit the security of foreign countries' transmissions [20];
- **Stream ciphers.** Symmetric key ciphers in which each digit is encrypted combining it with a pseudorandom cipher [40];
- **Block ciphers.** Symmetric key ciphers in which a set of bits with a fixed length (called block) is encrypted all at once [44];
- **Compression mechanism.** TLS feature used to reduce the amount of data sent through the network [24];
- **Hash functions.** Function that takes as input data of arbitrary size and produces as output a string with fixed length [49];
- **Renegotiation.** TLS feature used to enhance the security of an already established session without dropping the current connection [27];

Table 2. Known TLS attacks.

Name	Vulnerability	Attack	Affects
3SHAKE [31]	Renegotiation feature	Completing three handshakes with incorrectly placed certificates	ANY
Bar Mitzvah [28]	RC4 steam cipher	Extracting weak keys by targeting the first 100 bytes of the ciphertext	ANY
BEAST [17]	Initialization vector in cipher block chain	Guessing the plaintext to retrieve the symmetric key	TLS 1.0
CRIME [33]	TLS header compression mechanism (DEFLATE)	Continuously requesting data from the server in order to decrypt the session cookies (inferring the encryption)	ANY
DROWN [3,36]	SSLv2 weakness due to the use of export ciphers	Decrypting intercepted TLS connections by connecting to an SSLv2 server that uses the same private key	SSLv2
Logjam [1,18]	Weakness of export cipher suites	Negotiating the use of weak cipher suites (`DHE_EXPORT`)	TLS 1.1
Lucky 13 [2,12]	CBC-mode weakness due to HMAC-SHA1 decryption failure information leakage	Replacing the last bytes with chosen bytes and monitoring the transmission time	ANY
POODLE [32]	SSLv3 weakness due to the missing validation of padding bytes	Downgrading to SSLv3 and guessing the padding in order to slowly recover plaintext	SSLv3
RC4 NOMORE [48]	Bias in the generation of the "random" keys of the RC4 stream cipher	Statistically analyzing the Fluhrer-McGrew biases	ANY
SLOTH [4,13]	Availability of weak hash functions	Requesting a RSA-MD5 certificate signature and looking for collisions	TLS 1.2
Reneg. [42]	Renegotiation feature	Blocking the handshake process of the victim and use it to complete the attacker's transaction	ANY
Sweet32 [5]	64-bit block ciphers	Mounting a birthday attack which creates collisions	TLS 1.2
Truncation [46]	Server incorrect handling of the TLS termination mode through multiple connections	Keeping the victim's session alive (by blocking the logout request sent to the server)	ANY
BREACH [16]	HTTP compression mechanism	Requesting data from the server in order to guess the response body (note: without downgrading the SSL/TLS connection)	ANY

Termination protocol. Exchange of alert messages which signals the end of the message sending [21, §7.2.1].

Among all the attacks, here we detail the two used in our experimentation (see Sect. 6.2): CRIME [33] and BREACH [16]. Both attacks are related to the availability of DEFLATE [24], a compression algorithm that reduces the size of an input by replacing duplicate strings with a reference to their last occurrence. Given that neither TLS nor HTTP hide the size of each message, an attacker can exploit this information leakage to steal sensitive data. Supposing the will to steal session cookies, the attack is performed by injecting (e.g., using a controlled JavaScript loaded by the victim) different characters into the client's messages trying to guess the cookie. Thanks to DEFLATE, if the guess is wrong and the characters are not part of the cookie, the size of the response will be bigger. On the other hand, if the attacker guessed correctly, the size will remain the same. This attack is referred as CRIME if it exploits the compression within TLS, BREACH otherwise.

3 Tools Comparison

There are many TLS analyzers on the market and we wanted to understand which one suited better our purposes. For this reason, we decided to compare them to find the one who had the highest amount of features.

Table 3. Tool comparison - server.

Checks	sslscan	sslenum	TLSSLed	TLS-atk	3Shake_chk	testssl
SSLv3, TLS 1.0, 1.1 and 1.2, RC4	●	●	●	●	❍	●
AES ciphers	◗	◗	●	●	❍	●
Weak ciphers	◗	◗	●	◗	❍	●
SSLv2, Secure renegotiation	●	❍	●	●	❍	●
POODLE, CBC-mode cipher, 3DES	●	●	❍	●	❍	●
MD5/SHA1 signature alert	●	●	●	❍	❍	●
Sweet32	●	◗	❍	◗	❍	●
Certificate expiration	●	❍	●	❍	❍	●
Weak DH parameters	●	●	❍	❍	❍	●
Heartbleed, TLS compression	●	❍	❍	●	❍	●
BEAST	◗	❍	●	❍	❍	●
TLS 1.3, DROWN	❍	❍	❍	●	❍	●
Qualys scoring	❍	●	❍	❍	❍	●
More analysis[a]	❍	❍	❍	❍	❍	●
3SHAKE	❍	❍	❍	❍	●	❍

[a] server's default picks, certificate info, HSTS, HPKP, security headers, cookie, reverse proxy, client simulations, SPDY and HTTP2 availability

Table 3 shows the comparison between six tools that perform server-related TLS vulnerabilities[2]. Each detection is identified depending on the type of information resulting. In particular, ●,◗ and ❍ mean an explicit, implicit (which can be inferred using other explicit detections) or missing detection, respectively. The evaluated tools are:

sslscan [39]: the analyzer is able to detect the full set of available ciphers on a webserver. The default output shows the full list of accepted/rejected connections, detailing each line with the cipher's name, its key length and the used protocol;

ssl-enum-ciphers [29]: script developed for the `nmap` security scanner [30] that lists all the available cipher suites, compression methods and a small set of possible misconfigurations. The generated report shows the set of ciphers (available per protocol) with the relative Qualys' rating [38], a grade which goes from A+ to F depending on the level of provided security;

TLSSLed.sh [45]: built on top of an older version of `sslscan` [39], this script check if the server supports old protocols, weak ciphers and for the certificate signature. The verbose output highlights the results using different colours;

TLS-Attacker [47]: open source framework for analyzing TLS libraries. It can be fully-customized to perform any kind of connection and contains a set of pre-configured attacks for testing purposes. By running each attack, the user can understand whether or not the server is vulnerable;

3SHAKE checker [51]: is a simple script that checks if the target server supports `extended_master_secret`, an extension specifically designed to mitigate the 3SHAKE [31] attack. The output shows, for each available version of TLS, if the extension request has been accepted;

testssl.sh [50]: is a fully-featured open source command-line 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` is able to list the set of ciphers available per protocol, analyze the chain of trust of a provided certificate, simulate handshakes and much more. These features make `testssl.sh` the most powerful tool among the evaluated. The generated report contains the results for all the performed analysis, associated with a colour that signals the severity of the detected result.

All the listed tools work by repeatedly connecting to the target server using specifically crafted *ClientHello* messages. By checking the server's responses (i.e. *ServerHello*), the tools are able to understand the server's configuration. Besides the amount of provided features, the compared tools have a major limitation: all of them offer little or no explanation on how to actually mitigate the detected weaknesses. This somehow defies their purpose given that a system administrator will still have to spend a lot of time and effort researching the most appropriate set of mitigations to apply.

[2] Given the need for modularity, we focused on local analyzers rather than their online counterparts.

Table 4. Tool comparison - mobile clients.

Checks	Mallodroid	Tapioca
Detect non-default trust managers	●	❍
Check client's certificate validation	◗	●
Enumerate contacted hosts	❍	●
Validate HTTPS negotiations	❍	●
Read encrypted traffic	❍	●

3.1 Mobile Clients

As mentioned in the introduction, while in a browser the handle 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 or to implement his own methods. In both cases, an incorrect certificate handling may lead to several authentication-related issues. For this reason, there is the need for specific tools.

Table 4 shows the differences between two Android-related analyzers:

Mallodroid [14]: Python script (built on top of Androguard [10]) that performs static analysis on the code of an Android application. Taking as input the app installer (`.apk`), Mallodroid 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 an non-standard trust manager and checks the related methods;

Tapioca [11]: testing framework that performs a series of unique checks by simulating a MITM. Using different types of packet capture, the tools 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.

4 Mitigations Identification

Given the known vulnerabilities described in Sect. 2.2, system administrators should identify and follow a set of mitigations. To assist them we have collected in Table 5 the current best practice to mitigate the known vulnerabilities of TLS 1.2. The vast majority of the mitigations is applied by changing some lines in the server's configuration file while the remaining are related to vulnerable/outdated support libraries. The identification of such mitigations is not trivial because the currently available reports (see Appendix B) lack of clear indications on which is the source of misconfiguration.

Table 5. List of Mitigations for TLS 1.2.

Mitigation	Attack
Disable renegotiation	3SHAKE [31]
	Renegotiation attack [42]
Enable the use of extended_master_secret TLS extension	3SHAKE [31]
Disable RC4	Bar Mitzvah [28]
	RC4 NOMORE [48]
Disable the compression mechanism	CRIME [33]
Disable SSLv2	DROWN [3]
Use AEAD ciphers	Lucky 13 [2]
Disable SSLv3	POODLE [32]
Disable RSA-MD5 certificate signature	SLOTH [4]
Enforce AES usage (and disable 3DES when possible)	Sweet32 [5]
Enforce the termination mode	TLS Truncation [46]
Disable HTTP compression (may slow down the transmission)	BREACH [16]
Ignore self-signed certificates and perform a complete validation (up to the trusted root)	Accept self-signed certs [34]
Check if the hostname (from the certificate) matches the one related to the transmission	Setting a permissive hostname verifier [34]

5 TLSAssistant

During our study of TLS-related vulnerabilities, we noticed that all the currently available TLS analyzers have two major limitations. Putting aside the amount of provided features, all the examined tools gave little or no explanation on how to actually mitigate the detected weaknesses. On the other hand, every tool focuses on a specific party of the communication (either server or client) thus making its usage only part of a complete analysis.

To assist average system administrators and app developers to deploy resilient instances of the TLS protocol we propose TLSAssistant. By bringing together different powerful analyzers, our tool is able to cover a full-range of analysis on all the parties involved in a secure communication and to provide a set of mitigation measures that aim to thwart the impact of the identified vulnerabilities.

5.1 Architecture

TLSAssistant is written in Bash and can thus 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. The content of the report depends on the detected weaknesses and on the level of verbosity the user chose. Being built on top of other works, our TLSAssistant has been designed to be modular and easily upgradable. Figure 2 shows the architecture with its two main components: Analyzer and Evaluator.

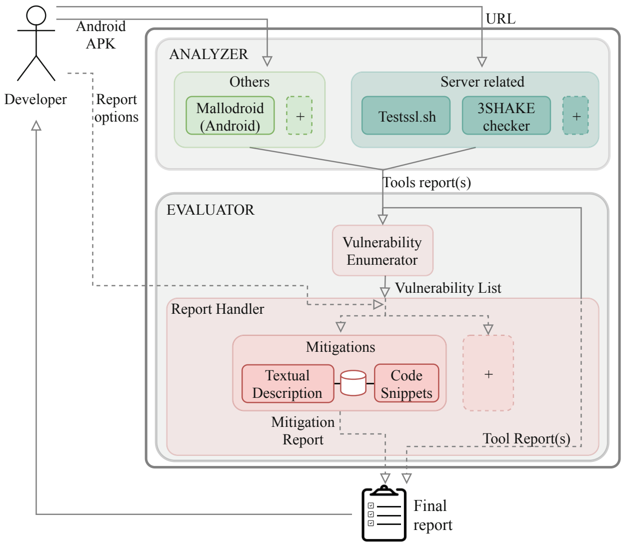


Fig. 2. TLSAssistant architecture

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. Currently, the set of integrated tools consists of command-line scripts written either in Bash or Python. At the time of writing, the Analyzer integrates the following tools:

- **Testssl.sh** [50] chosen among many others (as shown in Sect. 3) due to the enormous amount of features and for its ongoing development;
- **3SHAKE checker** [51] added to make the Analyzer able to test whether a server is vulnerable to the Triple Handshake Attack or not. This is an example of the continuous integration that has driven the design of TLSAssistant: being able to integrate different analyzers to become a useful toolbox for a complete TLS-vulnerability detection;

Mallodroid [14] even if less powerful than Tapioca (see Sect. 3), it was more suitable for our modularity requirement. Indeed, the current version of the installer of Tapioca turns the client machine into a dedicated appliance; a design choice incompatible with our tool.

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 and our main contribution, it is responsible for the enumeration of the detected vulnerabilities and the generation of the report that will guide the system administrator towards all the mitigations to be applied. It can be seen as 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 system administrators' choice, renders the final output. While TLSAssistant has been developed to be modular, the only available source of information currently available 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. The Report handler currently offers three kinds of report, 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. Is the most basic form of report, it only contains a description of how the related mitigation works;
- **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 currently only able to provide snippets for Apache HTTP server. We plan to extend the code coverage to all the most common webservers available on the market;
- **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.

6 Experimental Evaluation

To evaluate TLSAssistant's efficacy, we have analyzed a real use-case scenario involving the Italian eID card (CIE 3.0) [9] (Sect. 6.1) and conducted a user-study experimentation involving university students (Sect. 6.2). These two instances helped us prove that the result of our work is effective both for security experts, who may benefit from an additional support, and for unexperienced users who seamlessly became able to perform complex mitigations without the need for an in-depth knowledge.

6.1 Use-Case: CIE 3.0

In a joint collaboration between FBK and IPZS (acronym for "Istituto Poligrafico e Zecca dello Stato") [35], which is the Italian state printing office and mint, we implemented a mobile authentication mechanisms that uses the Italian electronic identity card (CIE 3.0 - Carta d'Identitá Elettronica) [9] to access public administration online services. Being the use of TLS the basic building block of the solution, any unpatched vulnerability (see Sect. 2.2) may compromise the entire authentication process.

For this reason, we run TLSAssistant targeting a prototype of the infrastructure and found that the deployment (which was entering in the final development stages) was prone to *Lucky 13*, *3SHAKE* and an incorrect certificate handling. These three issues, that can be easily go unnoticed for a variety of reasons, have now been fixed. This example clearly shows how running a tool like TLSAssistant can help even expert system administrators to determine if a new deployment contains some severe misconfigurations.

6.2 User Study

The following paragraphs will detail the settings of the experimentation (designed following the template and guidelines by Wohlin et al. [8]) and a summary of the main results.

Experiment Scoping and Planning. As described in Sect. 5, TLSAssistant is based on the most powerful TLS analyzers available on the market. The main additional feature is the generation of a report that assist the user during the mitigation process: together with the list of vulnerabilities, a textual description of the mitigations and (when is possible) a corresponding code snippet is provided.

The *goal* of this study is to analyze the effect of providing a set of mitigations with the *purpose* of evaluating the support offered by TLSAssistant in patching a TLS configuration.

The *context* of this study consists of:

Subjects: 16 Bachelor and Master students from University of Trento (with background on information security) playing the role of an unexperienced system administrator;

Objects: two VMs with custom-compiled misconfigured versions of Apache HTTP Server v2.4.37 and OpenSSL v1.0.2:
- O_1 a TLS configuration vulnerable to BREACH;
- O_2 a TLS configuration vulnerable to CRIME;

It is important to note that the proposed objects are representative of realistic TLS misconfigurations. To fit the time constraint of our experiment, only one vulnerability is present in each object. The selected objects are comparable in terms of complexity of the operation required to patch the problem.

Research Questions and Hypothesis Formulation. In this study, we want to evaluate whether the report provided by TLSAssistant (with textual descriptions of the mitigations and code snippets) facilitates the patching task in terms of time and correctness. Thus, our research questions are:

RQ_1 (on time): does the time spent by a system administrator in patching an error decrease when the tool provides a text description of the mitigation and the corresponding code snippet?

RQ_2 (on correctness): does the capabilities of a system administrator in patching an error increase when the tool provides a text description of the mitigation and the corresponding code snippet?

Thus, the null hypothesis can be formulated as follows:

H_{01} (on time): providing a text description of the mitigation and the corresponding code snippet does not significantly decrease the time spent by a system administrator to patch the error;

H_{02} (on correctness): providing a text description together with a code snippet of the mitigation does not significantly increase the capability of a system administrator to patch the error.

Variables Selection. To measure the subject's capability to perform a patching task (*vulnerability detected and solved*) and the time spent, we asked subjects to run the provided tool, look at the resulting report, and perform the patching task (perform the required operations to patch the misconfiguration).

The main factor of the experiment — that acts as an independent variable — is the presence of the treatment during the execution of the task. In our experiment, we have considered the following alternative treatments:

Treatment 1 (Tr_1): TLSAssistant provides as report a list of vulnerabilities plus a textual description of the mitigations and a suggested code snippet to perform the mitigation.

Treatment 2 (Tr_2): TLSAssistant provides as report the original reports of the tools that are composing the server-related module of the ANALYZER (Testssl.sh and 3SHAKE checker).

Experiment Design and Procedure. We adopt a counter-balanced experiment design intended to fit two lab sessions. Subjects are classified into four groups (despite they work alone), each one working in two labs on different objects with different treatments. The design allows for considering different combinations of objects and treatments in different order across labs (see Table 6).

Before our experiment, subjects were properly trained with lectures and exercises on TLS. The purpose of training is to make subjects confident about the kind of tasks they are going to perform and the environment they will have available.

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

Table 6. Labs.

	Group A	Group B	Group C	Group D
Lab 1	O_1 with Tr_1	O_2 with Tr_2	O_2 with Tr_1	O_1 with Tr_2
Lab 2	O_2 with Tr_2	O_1 with Tr_1	O_1 with Tr_2	O_2 with Tr_1

1. complete a pre-experiment survey questionnaire;
2. for each of the two labs to be performed: (i) mark the start time; (ii) perform the patching task; and (iii) mark the stop time;
3. complete a post-experiment survey questionnaire.

Post-experiment survey questionnaire (reported in Appendix A) deals with object clarity of the tasks, cognitive effects of the treatments on the behaviour of the subjects and perceived usefulness of TLSAssistant.

Results. The amount of time required to correctly patch a vulnerability is significantly longer when working with the report provided in Tr_2 than when working with the report with the mitigations (Tr_1): 25 min on average to fix a vulnerability with Tr_2, 7 min on average to fix a vulnerability with Tr_1. Thus, hypothesis H_{01} on time can be rejected. Therefore, we can formulate the following alternative hypothesis:

H_{A1}**:** providing a text description of the mitigation and the corresponding code snippet decreases the time spent by a system administrator to patch the error.

Regarding the task correctness all students were able to correctly patch the vulnerability with Tr_1; however, just the 68.75% of students was able to perform a proper vulnerability patch with Tr_2, which corresponds to a 31.25% difference on the overall sampled population. For this reason, we can accept the following alternative hypothesis:

H_{A2}**:** providing a text description together with a code snippet of the mitigation increases the capability of a system administrator to patch the error.

Moreover, from the post-experiment survey we can learned that the 81.25% of the students considers Tr_1 more useful and the 93.25% assessed that Tr_2 is more complex to understand. In addition, all the students positively recommend our tool. Here we report some comments:

"Fast, correct and easy to use. It found the vulnerability and helped me solving it"
"It would be very easy to fix such vulnerabilities following the given instructions. Also, you can search for more info about the vulnerability itself, which can help you to learn more about TLS."
"I won't waste a lot of time looking for all vulnerabilities"

7 Conclusions and Future Work

To assist system administrators with limited security skills to deploy resilient instances of the TLS protocol suite we propose TLSAssistant, a fully-featured tool that combines state-of-the-art tools with a report system that provides appropriate mitigations.

To design this tool, we have: *(i)* compared the state-of-the-art tools for TLS analysis, *(ii)* classified known TLS vulnerabilities and *(iii)* identified their mitigations. Finally, to validate the efficacy of our tool, we performed a user-study experimentation involving university students and analyzed a real use-case scenario involving the Italian eID card (CIE 3.0) [9].

As future work, we plan to extend TLSAssistant 's capabilities by *(i)* improving the webserver coverage; *(ii)* supporting more inputs (e.g., configuration files); *(iii)* automatize the mitigation process and further analyze experimentation's results by using statistical test, including co-factor analysis such as subject's experience, learning across tasks and more. As a second objective, we plan to use TLSAssistant to increase awareness and education in cybersecurity. A step in this direction is being made by integrating CVE identifiers, CVSS scores and modelling a series of attack trees [41], a hierarchical representation on how each attack can be mounted and which security properties it violates. Finally, we also plan to make TLSAssistant's source code freely available for anyone who wants to contribute to this project.

Acknowledgments. The authors would like to thank IPZS for the collaboration on the development of the authentication solution based on the CIE 3.0 carried out in the context of the joint laboratory DigimatLab between FBK and IPZS.

A Post-questionnaire

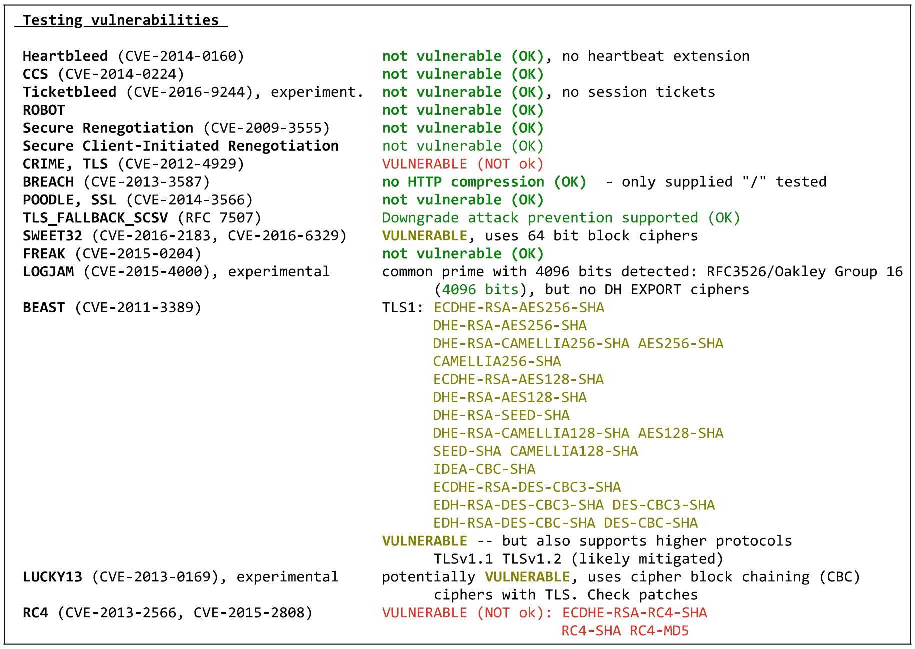
Table 7 shows the content of the post-experiment survey questionnaire mentioned in Sect. 6.2. It deals with object clarity of the tasks, cognitive effects of the treatments on the behaviour of the subjects and perceived usefulness of TLSAssistant. The first set of questions (Q1–Q6) needs to be answered twice (one answer for each performed lab) while the remaining set only needs to be answered once as it refers to the overall session.

Table 7. Post-experiment survey questionnaire.

ID	Applies to	Question
Q1	Each lab	I had enough time to perform the tasks (1–5)
Q2	Each lab	I experienced no difficulty in patching the vulnerability given the report (1–5)
Q3	Each lab	How much time (in terms of percentage) did you spend looking at the TLS configuration code? (0, $<$20%, $\geq$20% and $<$40%, $\geq$40% and $<$60%, $\geq$60% and $<$80%, $\geq$80%)
Q4	Each lab	How much time (in terms of percentage) did you spend looking at online documentation on TLS vulnerabilities? (0, $<$20%, $\geq$20% and $<$40%, $\geq$40% and $<$60%, $\geq$60% and $<$80%, $\geq$80%)
Q5	Each lab	Provide some examples of online queries you used to search the vulnerabilities online (e.g., keywords used)
Q6	Each lab	Which steps did you take to perform the tasks? (e.g., run command Y, opened file X, ..)
Q7	Overall	Which report did you find more useful. (Report of Lab 1–2)
Q8	Overall	Which report did you find more easy to read. (Report of Lab 1–2)
Q9	Overall	Which report did you find more complex to understand. (Report of Lab 1–2)
Q10	Overall	The textual description of the mitigation is useful to complete the tasks (1–5)
Q11	Overall	The code snippet is useful to complete the tasks (1–5)
Q12	Overall	How did you use the code snippet? (Copy-pasted where needed, Typed manually where needed, Used to perform a web search)
Q13	Overall	Would you use TLSAssistant for your work? (Yes, No, Maybe)
Q14	Overall	Motivate your answer (to the previous question). (open question)
Q15	Overall	Do you know any tool that performs similar tasks? (open question)
Q16	Overall	Do you have any suggestion related to the tool usage? (open question)
Q17	Overall	Do you have any suggestion related to the amount of information provided by TLSAssistant's report? (open question)

B Report snippet

To show the effort required by a system administrator in identifying the required mitigation, we show a snippet of the `testssl`'s report (see Fig. 3). It contains the



```
 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
```

Fig. 3. testssl report snippet (Color figure online)

list of checked vulnerabilities matched with their presence in the analyzed TLS deployment. The status of each vulnerability is shown with a combination of a string (e.g.; "potentially vulnerable") and a color that represents the severity of the finding.

Not the shown snippet nor any other part of the report give any useful insight on how to actually mitigate the detected vulnerabilities.

References

1. Adrian, D., et al.: Imperfect forward secrecy: how Diffie-Hellman fails in practice. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015). https://doi.org/10.1145/2810103.2813707
2. AlFardan, N.J., Paterson, K.G.: Lucky thirteen: breaking the TLS and DTLS record protocols. In: IEEE Symposium on Security and Privacy, SP, pp. 526–540 (2013). https://doi.org/10.1109/SP.2013.42
3. Aviram, N., et al.: DROWN: breaking TLS with SSLv2. In: 25th USENIX Security Symposium (2016)
4. Bhargavan, K., Leurent, G.: Transcript collision attacks: breaking authentication in TLS, IKE and SSH. In: 23rd Annual Network and Distributed System Security Symposium, NDSS (2016)

5. Bhargavan, K., Leurent, G.: On the practical (in-)security of 64-bit block ciphers: collision attacks on HTTP over TLS and OpenVPN. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016 (2016). https://doi.org/10.1145/2976749.2978423
6. Blog, G.S.: SHA-1 Certificates in Chrome. https://security.googleblog.com/2016/11/sha-1-certificates-in-chrome.html
7. Bright, P.: Apple, Google, Microsoft, and Mozilla come together to end TLS 1.0. https://arstechnica.com/gadgets/2018/10/browser-vendors-unite-to-end-support-for-20-year-old-tls-1-0/
8. Cartwright, M.: Book Review: Experimentation in Software Engineering: An Introduction. By Wohlin, C, Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. Kluwer Academic Publishers (1999). ISBN 0-7923-8682-5. Softw. Test. Verif. Reliab. (2001). https://doi.org/10.1002/stvr.230
9. Dell'Interno, M.: Carta di identitá elettronica. https://www.cartaidentita.interno.gov.it
10. Desnos, A.: Github: Androguard. https://github.com/androguard/androguard
11. Dormann, W.: 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
12. Ducklin, P.: Boffins 'crack' HTTPS encryption in Lucky Thirteen attack. https://nakedsecurity.sophos.com/2013/02/07/boffins-crack-https-encryptionin-lucky-thirteen-attack/
13. Ducklin, P.: 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/
14. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: 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, pp. 50–61 (2012). https://doi.org/10.1145/2382196.2382205
15. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: validating SSL certificates in non-browser software. In: ACM Conference on Computer and Communications Security, pp. 38–49 (2012). https://doi.org/10.1145/2382196.2382204
16. Gluck, Y., Harris, N., Prado, A.: BREACH: reviving the CRIME attack. http://breachattack.com/
17. Green, M.: A Diversion: BEAST Attack on TLS/SSL Encryption. https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/
18. Green, M.: Attack of the week: Logjam. https://blog.cryptographyengineering.com/2015/05/22/attack-of-week-logjam/
19. Green, M.: The Internet is broken: could we please fix it? https://blog.cryptographyengineering.com/2012/02/28/how-to-fix-internet/
20. Grimmett, J.: Encryption export controls (2001). http://www.au.af.mil/au/awc/awcgate/crs/rl30273.pdf
21. Group, N.W.: The TLS Protocol: Version 1.0. https://tools.ietf.org/pdf/rfc2246.pdf
22. Group, N.W.: The Transport Layer Security (TLS) Protocol: Version 1.1. https://tools.ietf.org/pdf/rfc4346.pdf
23. Group, N.W.: The Transport Layer Security (TLS) Protocol: Version 1.2. https://tools.ietf.org/pdf/rfc5246.pdf
24. Group, N.W.: Transport Layer Security Protocol Compression Methods. https://tools.ietf.org/pdf/rfc3749.pdf

25. Group, O.W.: OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens. https://tools.ietf.org/pdf/draft-ietf-oauth-mtls-10.pdf
26. IETF: The Transport Layer Security (TLS) Protocol: Version 1.3. https://tools.ietf.org/pdf/rfc8446.pdf
27. IETF: Transport Layer Security (TLS) Renegotiation Indication Extension. https://tools.ietf.org/pdf/rfc5746.pdf
28. IMPERVA: Attacking SSL when using RC4. https://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf
29. Kolybabi, M., Lawrence, G.: ssl-enum-ciphers. https://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html
30. Lyon, G.: Nmap: the Network Mapper. https://nmap.org
31. Microsoft-Inria: Triple Handshakes Considered Harmful: Breaking and Fixing Authentication over TLS. https://www.mitls.org/pages/attacks/3SHAKE
32. Möller, B., Duong, T., Kotowicz, K.: This POODLE Bites: Exploiting the SSL 3.0 Fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf
33. NIST: CVE-2012-4929. https://nvd.nist.gov/vuln/detail/CVE-2012-4929
34. NowSecure: Fully Validate SSL/TLS. https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/fully-validate-ssl-tls.html
35. Poligrafico e Zecca dello Stato Italiano. https://www.ipzs.it
36. Pornin, T.: What is DROWN and how does it work? https://security.stackexchange.com/a/116140/186367
37. Qualys: SSL Pulse. https://www.ssllabs.com/ssl-pulse/
38. Qualys, I.: SSL Server Rating Guide. https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide
39. rbsec. https://github.com/rbsec/sslscan/releases/tag/1.11.11-rbsec
40. Robshaw, M.: Stream ciphers (1995). ftp://ftp.rsasecurity.com/pub/pdfs/tr701.pdf
41. Schneier, B.: Attack Trees. https://www.schneier.com/academic/archives/1999/12/attack_trees.html
42. SecurityLearn: SSL Attacks. http://www.securitylearn.net/tag/ssl-renegotiation-attack/
43. Services, A.W.: Alexa Top Sites. https://aws.amazon.com/alexa-top-sites/
44. Shannon, C.E.: Communication theory of secrecy systems*. Bell Syst. Tech. J. **28** (1949). https://doi.org/10.1002/j.1538-7305.1949.tb00928.x
45. Siles, R.: TLSSLed v1.3. http://blog.taddong.com/2013/02/tlssled-v13.html
46. Smyth, B., Pironti, A.: Truncating TLS connections to violate beliefs in web applications. In: 7th USENIX Workshop on Offensive Technologies, WOOT (2013)
47. Somorovsky, J.: Systematic fuzzing and testing of TLS libraries. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, pp. 1492–1504 (2016). https://doi.org/10.1145/2976749.2978411
48. Vanhoef, M., Piessens, F.: RC4 NOMORE (Numerous Occurrence MOnitoring & Recovery Exploit). https://www.rc4nomore.com/
49. Weisstein, E.: Hash Function. http://mathworld.wolfram.com/HashFunction.html
50. Wetter, D.: /bin/bash based SSL/TLS tester: testssl.sh. https://testssl.sh
51. Young, C.: 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/