Empirical Validation on the Usability of Security Reports for Patching TLS Misconfigurations: User- and Case-Studies on Actionable Mitigations

Salvatore Manfredi[1,2*], Mariano Ceccato[3], Giada Sciarretta[1], and Silvio Ranise[1,4]

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

[2]DIBRIS, University of Genoa, Genoa, Italy

[3]University of Verona, Verona, Italy
mariano.ceccato@univr.it

[4]Department of Mathematics, University of Trento, Trento, Italy

Received: December 19, 2021; Accepted: February 10, 2022; Published: March 31, 2022

Abstract

Several automated tools have been proposed to detect vulnerabilities. These tools are mainly evaluated in terms of their accuracy in detecting vulnerabilities, but the evaluation of their usability is commonly neglected. Usability of automated security tools is particularly crucial when dealing with problems of cryptographic protocols for which even small—apparently insignificant—changes in configuration can result in vulnerabilities that, if exploited, pave the way to attacks with dramatic consequences for the confidentiality and integrity of the exchanged messages. This becomes even more acute when considering such ubiquitous protocols as the one for Transport Layer Security (TLS for short). In this paper, we present the design and the lessons learned of a user study, meant to compare two different approaches when reporting misconfigurations. Results reveal that including contextualized actionable mitigations in security reports significantly impact the accuracy and the time needed to patch TLS vulnerabilities. We used these results to build an open-source tool called TLSAssistant, able to combine state-of-the-art analyzers with a report systems that generates actionable mitigations to assist the user. Finally, we report our experience in using TLSAssistant in two case studies conducted in a corporate environment.

Keywords: vulnerability detection, usability study, actionable mitigations, security reports, TLS misconfiguration

1 Introduction

Transport Layer Security (TLS) consists of a set of cryptographic protocols designed to provide secure communications over a network. TLS is widely used in client-server applications to secure all the communications by preventing eavesdropping and tampering. It is mainly used to secure the traffic between a website and a web browser (HTTPS protocol). Another use of TLS is on top of Transport Layer protocols

*Corresponding author: Security & Trust, FBK, Via Sommarive, 18, 38123 Trento, Italy such as the File Transfer Protocol (FTP) for the transfer of computer files or the Simple Mail Transfer Protocol (SMTP) for electronic mail transmission.

The TLS deployment process requires a non-trivial amount of knowledge [1, 2] that includes to correctly set a server certificate, choose the available protocols, ciphers and enabling other security mechanisms (e.g., HSTS). Its popularity has encouraged attackers to find vulnerabilities. The types of attacks vary widely and include the renegotiation of cipher suites to exploit weak encryption algorithms [3], the knowledge of initialization vectors to retrieve symmetric keys [4], and the use of libraries to exploit poor certificate validation in deployments where clients are non-browsers [5]. The downside of allowing a high level of customization transfers on the system administrators the burden of securing the deployments.[1] However, according to [6], security does not get enough attention as more than 70,000 of the most popular websites [7] still support officially deprecated versions of the TLS protocol [8].

To help administrators in deploying secure TLS instances, during the years, a variety of tools have been developed to assist system administrators and ease their work. These tools are usually able to verify TLS implementations, analyzing the deployment and providing a list of the possible attacks but miss two important features: *(i)* an *explanation* of the identified vulnerabilities and the attacks that may exploit them, and *(ii) actionable hints* on how to mitigate the attacks. The *explanation* would allow administrators to learn about the attacks and put the potential security problems in the right context (such as impact and likelihood), help them to distinguish relevant from irrelevant information when searching for more details, and even ease the acceptance of *actionable hints* that can even take the form of code snippet to be copy-pasted in a configuration file. Without these two features, a system administrator is left with the burden of finding enough information to define an appropriate mitigation strategy. This task is far from trivial as this kind of information is spread across several sources ranging from scientific papers to blog posts; each one with its jargon and background assumptions. Even disregarding the effort to collect enough material to mitigate a security problem, administrators should have enough skills to understand the (often subtle) details and turn the information into a concrete strategy to fix the problem.

We conjecture that a tool should go beyond than just identifying vulnerabilities, and that contextualized actionable suggestions would be very effective in practically explaining system administrators how to fix a wrong configuration and remove a specific security problem. In this paper, we present one user study and two corporate case studies to investigate our conjecture. The user study consists in an experimental assessment of our hypothesis by quantifying the benefit of providing mitigation hints on the capability of system administrators to (correctly and quickly) patch a TLS misconfiguration. During the experiment, bachelor and master students were asked to play the role of unexperienced system administrators who should patch defective TLS configuration files. The case studies assess the adoption of our tool in corporate environments. The former is related to a mobile authentication scheme while the latter is a real-world deployment of a company that has deployed a misconfigured TLS instance in its infrastructure. To experiment in real-world scenarios, we have developed a tool, called TLSAssistant, capable of identifying vulnerabilities and providing actionable descriptions of mitigations based on the findings of the user study

Our findings prove the effectiveness of security reports containing high level descriptions of contextual information about identified security problems (e.g., of the identified vulnerabilities) together with actionable mitigations as even un-experienced users were able to successfully mitigate complex attacks.

In our previous publication [9] we have:

- formally validated the hypothesis that reports containing actionable hints are more efficient as we learned that they drastically decrease the time required to patch TLS vulnerabilities and improve both problem identification and its resolution;

[1] `https://acloudguru.com/blog/engineering/cloud-governance-and-managing-risk`

- shared the vulnerable VMs, slides, questionnaires as they can be used both to replicate our study and as valuable asset for educational scenarios.

This work expands the aforementioned publication as we:

- introduce TLSAssistant, an open-source tool develop to exploit our findings on actionable reports. We briefly describe its architecture along with the set of available mitigations and offered features. Furthermore, we present a new feature that aims to assist system administrators when evaluating the compliance of their configuration against technical standards. In particular, we focus on the Italian standard on TLS usage describing its structure and the requirement it presents [10];
- describe two case studies for the corporate adoption of our tool. The first one covers the analysis of a mechanism that uses the Italian electronic identity card (CIE 3.0) to perform online authentication. The second one is a real-world example of an outsourced Software as a Service (SaaS) and the impact its misconfiguration has on the host infrastructures that use it.

Plan of the paper. Section 2 provides the necessary background notions on TLS and its vulnerabilities. Section 3 contains an overview of the state-of-the-art in terms of usability studies and the impact of providing hints for mitigations. Section 4 describes the decision process that led to the choice of the state-of-the-art report used in an experimental user study involving Bachelor and Master degree students. Section 5 describes the design of the user study defined to collect empirical evidence on the reports' effectiveness and Section 6 reports the results of this study. The results are then discussed in Section 7 in terms of their impact and implications. Section 8 introduces the tool TLSAssistant with its architecture, usage, and features. The section also describes the catalogue of attacks and related mitigations (Section 8.2) together with a discussion related to the compliance checker (Section 8.3), a new feature that aims to assist users when evaluating the compliance of their webserver against technical standards. Section 9 reports the use of TLSAssistant in two corporate case studies. The analysis of a mobile authentication scheme and the quality review of a sensitive Software as a Service solution. Section 10 concludes the paper and highlights future work.

2 Background

We provide some background notions to better understand the experimentation. We briefly describe the general structure of the TLS protocol in Section 2.1 and give a concise description of two known vulnerabilities in Section 2.2.

2.1 Transport Layer Security

TLS has been designed to provide both confidentiality and integrity between communicating entities [11] and is composed of two layers: the Handshake and the Record protocol.

The Handshake protocol can provide either mutual or one-way authentication and allows the parties to exchange all the information required to establish a reliable session, this includes the choice of a set of algorithms that will be used. This set, called cipher suite, is proposed by the client within the first handshake message and then chosen by the server. The set of cipher suites supported by the server is chosen by the system administrator who deployed it.

The Record protocol is deployed on top of a transport protocol (such as TCP) and encapsulates the messages coming from higher levels, it ensures confidentiality by using symmetric encryption algorithms and integrity by calculating the hash of the messages being sent. The keys and the algorithms used are the ones agreed during the handshake.

2.2 Vulnerabilities on TLS

On protocol versions prior to 1.3, TLS suffers from a wide set of vulnerabilities [12]. While some of the attacks are due to flaws in the logic of the protocol, others exploit the support of now-deprecated cipher suites or (in)voluntary weakening of security properties to bypass the authentication process (such as accepting self-signed certificates [13]). In this paper, we focus on the two used in our experiments (see Section 5.2): CRIME and BREACH. Both are compression side-channel attacks that combine three elements: the presence of a recurring part within the transmitted messages, the fact that both TLS and HTTP do not hide the length of the sent data and the availability of DEFLATE [14], a compression algorithm that reduces the size of an input by replacing duplicate strings with a reference to their last occurrence.

CRIME [15] is a security exploit that allows an attacker to decrypt the transmission if the parties agreed to use TLS-level compression. Supposing the attacker's intention is to steal a session cookie (whose ownership authenticates the user), the attack is performed by creating different client's messages containing guesses. Due to the DEFLATE usage, if the guess is wrong, the size of the server's response will be bigger than a valid one.

BREACH [16] exploits the same mechanism but using HTTP-level compression. Once the attacker has correctly guessed the first character of the shared secret, it starts a phase of trial-and-error in which she/he increasingly guesses bigger parts of the secret until its completion.

Being optional, the presence of DEFLATE can be seen as the single point-of-failure and thus its deactivation – that in Apache consists in changing two different files – is the suggested mitigation for both attacks.

3 Related work

Usability Studies in Cyber Security. Several usability studies have been conducted in order to assess tools and methodologies in different cyber-security domains, such as password storage, penetration testing and code obfuscation.

Naiakshina et al. conducted qualitative usability studies either with students [17] and with freelance developers working remotely [18], asking them to build a password storage mechanism. These studies showed that participant security knowledge does not guarantee the delivery of secure software.

In the domain of risk assessment, Allodi et al. measured the accuracy [19] and the difficulty [20] for students (with different technical education) in using the Common Vulnerability Scoring System to assess the severity of software vulnerabilities. Labunets et al. conducted a series of empirical evaluations to compare the effectiveness of two classes of threats-analysis methods [21] and the comprehensibility of two risk model representations [22].

Scandariato et al. conducted a series of controlled experiments to compare static analysis and penetration testing tools, in terms of how well they support developers in accurately detecting vulnerabilities [23], and then in fixing the code [24].

Ceccato et al. measured how code obfuscation influences the correctness and effectiveness of understanding and change tasks [25]. In a successive work, the same authors presented an extension with a larger set of experiments conducted on more obfuscation techniques [26]. Then, their replication package has been used by Hänsch et al. [27] to conduct a similar experiment and assess a slightly different set of obfuscations. Viticchié et al. empirically evaluated the attack delay introduced by a data obfuscation [28] and by code splitting [29]. They confirmed that attacks are still possible on protected programs, but they are delayed by a factor of six for that technique.

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

Table 1: TLS Analyzers - Features comparison

Features	ssl-enum-ciphers	sslscan	SSL Server Test	sslyze	testssl.sh
Open source & downloadable	✓	✓	X	✓	✓
Actively maintained	●○○	●●●	●●●	●●●	●●●
TLS vulnerability checks	●○○	●○○	●●●	●●○	●●●
Standalone	●○○	●○○	●●●	●●○	●●●
Highly customizable scan	●○○	●●○	●○○	●○○	●●●

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. [30] performed a systematic investigation on how the documentation available to developers directly affects both security and privacy properties, finding that most developers do use search engines and StackOverflow to address issues, leading to poor implementation results. Gorski et al. [31] evaluated the impact of providing security advise in case of API misuse, proving that the offered advices impacted positively on code security and did not affect the overall usability of the interface.

Krombholz et al. investigated the mental models of both users and sysadmins, their results show a large amount of misconceptions about threat models, protocol components and also the very same benefits of using HTTPS [32]. In [33], they also performed a series of controlled experiments to highlight the difficulties of deploying HTTPS, proving that it is far too complex even for people with adequate expertise. The findings of the latter have been partially verified by Bernhard et al. as they performed two usability studies narrowing the procedure to the certificate acquisition [34].

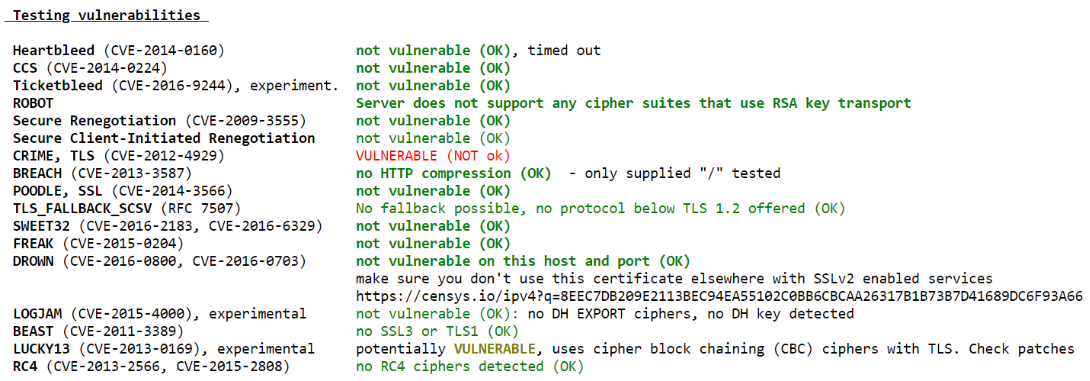
Finally, the study performed by Tiefenau et al. reveals that even experienced administrators struggle with keeping their systems up-to-date as the decision process that precedes the application of patches requires attention and is time-consuming [35]. To overcome this limitation, Li et al. suggests that helping system administrators during the information gathering would simplify the updating efforts and the likelihood of prioritizing the updates for the managed systems [36].

4 Security Reports

With the goal of evaluating the benefits of providing mitigation hints in a TLS security report we need to identify an offline TLS analyzer to use for our user study. The market offers several tools to support sysadmins with security TLS configurations, all of them adopt a similar approach, i.e., they repeatedly connect to the target server and send specifically crafted ClientHello messages. By checking the server's responses (i.e. Server-Hello messages), these tools infer the server configuration and assess if it is affected by known vulnerabilities. However, their reports usually contain only the list of detected vulnerabilities and they offer little or no explanation on how to actually mitigate the detected weaknesses. Among the publicly available scanners we can mention ssl-enum-ciphers [37], sslscan [38], testssl.sh [39], SSL Server Test [40] and sslyze [41]. We chose testssl.sh as it is the most complete and covers the largest amount of required features (see Table 1).

4.1 testssl.sh's Report

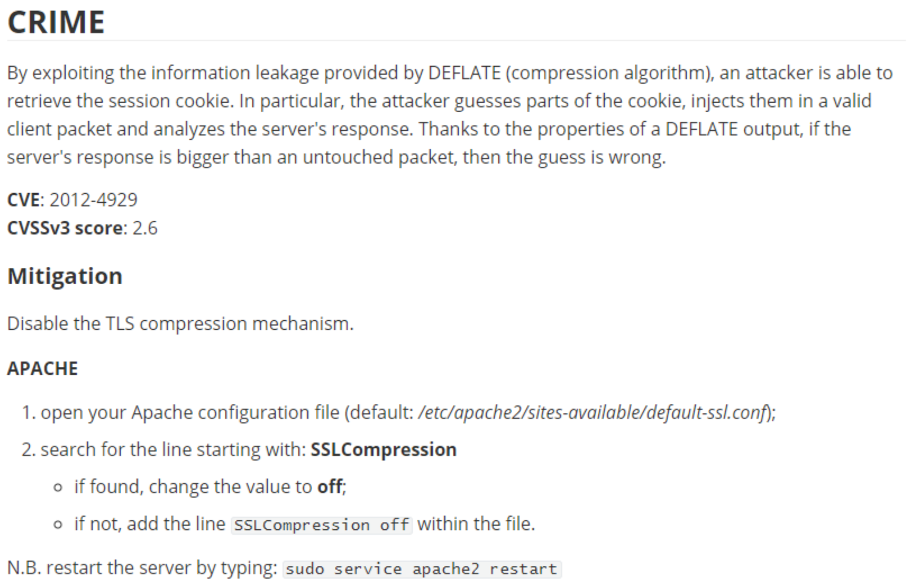
testssl.sh is a powerful open-source Bash script [39] that supports a wide range of state-of-the-art TLS-related checks. Checks include availability of ciphers and protocols, server preferences and an extensive set of information from the server certificate and its chain of trust. The report looks complete and quite verbose as it does not focus on the detected issues only, but it gives an overall view of the server status including also the passed checks (see a fragment in Figure 1).



```
 Testing vulnerabilities 

 Heartbleed (CVE-2014-0160)                not vulnerable (OK), timed out
 CCS (CVE-2014-0224)                       not vulnerable (OK)
 Ticketbleed (CVE-2016-9244), experiment.  not vulnerable (OK)
 ROBOT                                     Server does not support any cipher suites that use RSA key transport
 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)              No fallback possible, no protocol below TLS 1.2 offered (OK)
 SWEET32 (CVE-2016-2183, CVE-2016-6329)    not vulnerable (OK)
 FREAK (CVE-2015-0204)                     not vulnerable (OK)
 DROWN (CVE-2016-0800, CVE-2016-0703)      not vulnerable on this host and port (OK)
                                           make sure you don't use this certificate elsewhere with SSLv2 enabled services
                                           https://censys.io/ipv4?q=8EEC7DB209E2113BEC94EA55102C0BB6CBCAA26317B1B73B7D41689DC6F93A66
 LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK): no DH EXPORT ciphers, no DH key detected
 BEAST (CVE-2011-3389)                     no SSL3 or TLS1 (OK)
 LUCKY13 (CVE-2013-0169), experimental     potentially VULNERABLE, uses cipher block chaining (CBC) ciphers with TLS. Check patches
 RC4 (CVE-2013-2566, CVE-2015-2808)        no RC4 ciphers detected (OK)
```

Figure 1: testssl.sh report fragment



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 2: Enhanced report fragment on CRIME

4.2 Actionable Reports

To understand if system administrators would benefit from a concise yet informative report (see Section 5) we took a subset of testssl.sh reports, removed all the passed and informative checks and added a number of descriptive elements collected by fetching information from both scientific literature and each vendor's technical documentation. The resulting report provides a clear explanation of how the detected vulnerabilities can be exploited and a set of actionable mitigation measures that aim to thwart their impact, and operatively guide a system administrator in removing the found security defects. The provided mitigations are described at various levels of abstraction (see a fragment in Figure 2):

Attack Description: a high-level explanation—along with its CVE ID [42] and CVSS score [43]—on how the flaw can be exploited and which security property will be affected. It can be used to better understand the impact of the issue and prioritize its mitigation;

Textual description: natural language description of the TLS vulnerability and related mitigations (brief explanation of the actions to perform);

Code snippet: a fragment of code that can be copy-pasted into the webserver's configuration to seamlessly fix the weakness. Together with the snippet, the report will provide a set of steps on how to find the correct file/line to edit.

5 User Study Design

The *goal* of this study is to analyze the effect of providing a set of mitigations to system administrators with the *purpose* of evaluating the support offered by the actionable reports in patching a defective 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 two research questions:

RQ1. Do 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. Do 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 enhance the identification and patching of insecure TLS configurations in terms of speed and precision. There are other interesting point of views to consider such as those of *(i)* a researcher interested to empirically assess the benefit of hints in 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 guarantee effective deployment of a correctly configured infrastructure.

The experimental settings have been designed following the template and guidelines by Wohlin et al. [44] to select participants and present their demographics (Section 5.1), to define the experimental design and select appropriate metrics and dependent/independent variables (Section 5.2), to identify the most appropriate statistical tests (Section 5.3) and to identify the threats to the validity of our findings (Section 5.4).

5.1 Demographic Statistical Sample

We involved 62 participants in this study. They are Bachelor and Master students from the departments of Computer Science and Mathemathics of the University of Trento playing the role of unexperienced 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.

Participants were aware that they could drop at any time with no consequences, as they would not have been evaluated for their performance in the experiment. There was no compensation (neither money nor bonus in the exam mark) for their participation in the study.

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 2. 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).

Table 2: 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

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). 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).

Table 3: 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

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 [45]). 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 [46]; 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 [47, 48, 49].

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 were receiving 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 [50] with particular attention to trustworthiness, fairness, privacy, and confidentiality.

5.2 Experimental Setup and Execution

Systems. The systems used to conduct the experiment are two web servers with defective 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 Virtual-box machines. Instead of using a random pair of detectable misconfigurations (e.g., POODLE [51], Sweet32 [52] or others) we selected two vulnerabilities that are comparable in terms of complexity of the operations required to patch. Moreover, we ensured that they can realistically 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 2.2). We overcome the simplistic nature of the chosen vulnerabilities in two corporate case studies (see Section 9) that show harder-to-fix misconfigurations discovered in the wild. 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 [53] and TLS library [54]. 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, 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 [55], 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 [55] 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 [55]. 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 [55] to evaluate the overall usability of the TLS analyzer report is measured by a survey questionnaire (available in [56] and analyzed in Section 6.3).

Experimental Design. We adopt a counter-balanced experimental design intended to fit two lab sessions of 40 minutes each. Participants are randomly assigned to four groups (despite they work alone) balanced based on their seniority, each one working in two labs on different systems with different treatments. The design allows for considering different combinations of *Systems* and *Treatments* in different order across *Labs* (see Table 4).

Table 4: 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$_{hint}$

Experimental Procedure. Before the experiment, participants were properly trained with lectures and exercises on TLS, to recall the required background [56]. 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.

The experimental context has been set as similar as possible to a realistic scenario. As such, participants could run scanning tools as often as wanted, they could inspect the scan reports and browse the Internet to look for additional information. Moreover, by performing a dry-run test we ensured that the overall experiment can realistically be finished in two hours. The dry-run test helped us also to refine 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 [56] and we have described the collected data in Section 5.1.

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 [56] and discuss the answers in Section 6.3. The survey questionnaires deal with cognitive effects of the treatments on the behavior of the participants and perceived usefulness of the provided report.

5.3 Statistical Tests

We are interested to assess if the presence of mitigation hints has an impact on the *Correctness* and in the *Time* taken to fix defective TLS configurations. However, observed differences in the correctness of corrective tasks and time spent on them could be due to random variations 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 significance might mean also that there was an effect, but the effect was not observable (possibly because of to a small set of observations), rather than that the effect does not exist, i.e., we risk to commit a type-II error (nullification fallacy [57]). 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.

For each participant there are two distinct data points, one for the first lab and another one for the second lab, so we never perform multiple pairwise comparisons with overlapping data. Thus, there is no risk to inflate the family-wise error rate, and no correction factor (e.g., 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 [58] 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 [59].

Time. To test the differences in *Time*, we perform the two-tailed Mann-Whitney U test on all samples [60]. 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 [61] 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 $d \geq 0.474$ [62].

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* [63] (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 [64]. 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. The Fisher's exact test is used on categorical data, to compare the frequencies of yes/no answers for participants who worked with different tools. The Mann-Whitney U test is used to analyze answers to overall questions (not specific to any lab) when the answers were formulated using a Likert scale, checking for the null-hypothesis that the average answer was negative or neutral.

5.4 Threats to Validity

The main threats to the validity of this experiment belong to the internal, construct, conclusion and external validity threat categories [44].

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 5.2, 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 vulnerabilities related to a TLS configuration, namely BREACH and CRIME. Although different vulnerabilities might occur, the results obtained with these vulnerabilities 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.

6 User Study Results

This section presents the data collected during the experimental validation. After analyzing data with sound statistical tests, we formulate answers to the two research questions stated at the beginning of Section 5.

6.1 Analysis of Correctness

Table 5 shows the distributions of correct/wrong answers when using testssl.sh's reports (TR_{list}) and the actionable reports (TR_{hint}).

Almost all the participants were able to correctly patch the vulnerability when they were provided mitigations and code snippets, only one participant was not. Conversely, when participants were provided only with the list of vulnerabilities,[2] 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 5: Correctness in fixing an incorrect TLS configuration

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

Table 6 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 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.

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

System does not significantly influence the *Correctness* of tasks, so we can conclude that the two applications and the two corrective tasks were well-balanced and none was harder or easier to fix. Moreover, the *Lab* is not a significant factor, thus there is no evident learning effect between the two experimental sessions. This means that having performed a first lab does not improve the accuracy in the second lab and we only measure the difference actually due to 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 5) 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.

Table 6: 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

6.2 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 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 7. 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).

Table 8 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.

Table 7: 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

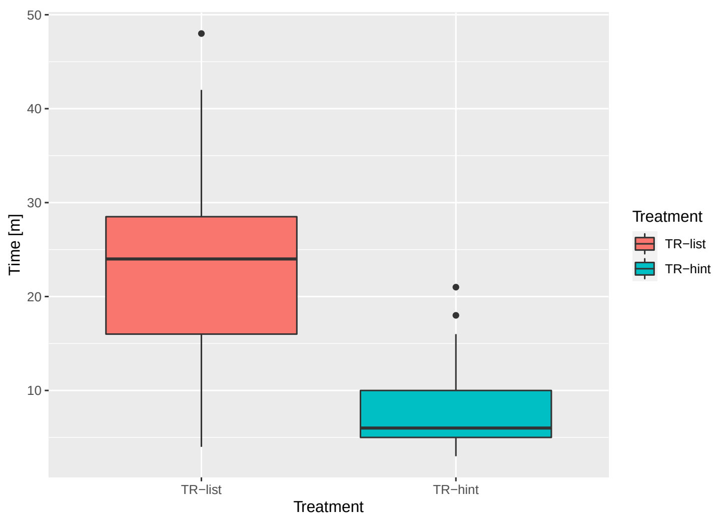


Figure 3: Time (in minutes) to fix a security issue

Table 8: 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

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 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.

6.3 Analysis of Survey Questionnaire

The survey questionnaire (available in [56]) is composed of three parts and is meant to collect the participants opinion on the experiment.

First Part. The objective of the first part is to collect the participants opinion on the security reports, to indirectly compare them. Answers to this first part are reported in Table 9. 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%", "≥ 20% and < 40%", "≥ 40% and < 60%", "≥ 60% and < 80%" and "≥ 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 9 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 6.2, where participants who worked with no mitigation hints took longer to complete their tasks.

Consistently, looking at responses to the second question, we notice that only participants working with the actionable reports experienced no difficulty in completing the tasks. Tasks were harder to complete when participants were only supported by the list of security defects.

Considering the answers to the third question, we see that online searches were used by the majority of the participants who worked just with the list of vulnerabilities (53 positive answers versus 7 negative answers). When the actionable reports were used, instead, the majority of participants did not resort to online search (3 positive answers versus 59).

Moving to the next two questions about time, we see a different approach on solving the assigned task. 53 participants assigned to TR_{list} spent less than 60% of the lab time looking at TLS configuration code, meaning that they did not try to understand how TLS was configured but focused on searching online or in the TLS report for the solution. Conversely, when using the actionable reports almost none of the participants searched online for TLS documentation.

Second Part. The second part of the survey is a more direct comparison between the two alternative approaches. In fact, we asked participants to make an explicit decision about the two reports. For each question, Table 11 reports the number [3] of decisions that participants formulated about tasks supported by TR_{hint} and TR_{list}.

The first question asked which was the most useful report when fixing the misconfiguration. The majority of the participants (41) considered the actionable reports the most useful.

Table 9: 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**

[3]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.

Table 10: 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**

Table 11: 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

The second question investigated if security reports were easy to read. 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 10 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 ≤ *"Neutral"*) confirms that this trend is statistical significant.

Third Part. This last part, 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* [65, 66] and is thus left for future work.

7 User Study Discussion

Often, new approaches may show trade-offs between contrasting goals and an optimal cost-benefit equilibrium has to be found. According to our experimental results, this is not the case when integrating actionable security hints into security reports. Indeed, we were able to observe a positive effect on both *Correctness* and *Time* of completion of the tasks.

In fact, 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.

Limited information: *Automated security tools convey insufficient information.* A TLS configuration is quite complex as it contains a lot of properties that might not be of immediate understanding, thus the bare list of security problems is not informative enough to let a system administrator fix it. Indeed, additional information is needed to fill the gap of a security report and guide the system administrator towards figuring what changes are needed. 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 6.3 and Table 9).

Correctness and Time: *Actionable maintenance hints improve correctness and time to fix.* Despite different tools find the same vulnerabilities, it is crucial how these are reported 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 6) in a timely manner (see Table 8 and Figure 3). 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 conducting corrective maintenance, a flat list of the detected vulnerabilities is not perceived as very useful, probably because they are neither informative nor easy to read. This leaves system administrators with no clue where to look for getting the necessary information or to distinguish relevant from irrelevant data gathered from, e.g., online searches. Thus, additional information has to be collected either spending time on the code or reading additional documentation. Conversely, sysadmins 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 [56] and Table 11). We use these considerations to build an open-source tool named TLSAssistant [67], discussed in the next section.

8 TLSAssistant

Our user study (see Sections 5-7) reveals that the usability of reports is largely impacted by the availability of contextualized actionable hints, as they have a positive effect on both the correctness and the time needed to fix a TLS vulnerability. To assist average system administrators to deploy resilient instances of the TLS protocol and to test existing ones we built TLSAssistant[4]. 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. Combining the effectiveness of the tools with our set of proposed mitigations, TLSAssistant is able to provide a wide set of actionable security measures that aim to thwart the impact of the identified vulnerabilities.

8.1 Architecture

TLSAssistant is a fully-featured tool that combines state-of-the-art TLS analyzers with a report system that suggests appropriate mitigations and identifies a wide range of viable attacks. Among the available options, 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, the type of report and the detected webserver. Figure 4 shows a high-level architecture with its two characteristic elements: the set of ANALYZERS and the MITIGATIONS.

ANALYZERS. It is the part of our tool that handles the analysis of both webservers and Android applications. It is composed of a series of state-of-the-art TLS analyzers (e.g., testssl.sh [39], tlsfuzzer [68], MalloDroid [69] and SUPER [70]) that are able to detect the presence of a wide range of vulnerabilities. By design, our tool has a flexible architecture that allows a continuous integration of newer and more sophisticated tools. The integrated tools allow the ANALYZERS 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 tool related to the required scan, collect their reports and combine them with our MITIGATIONS.

[4]`https://st.fbk.eu/tools/TLSAssistant`

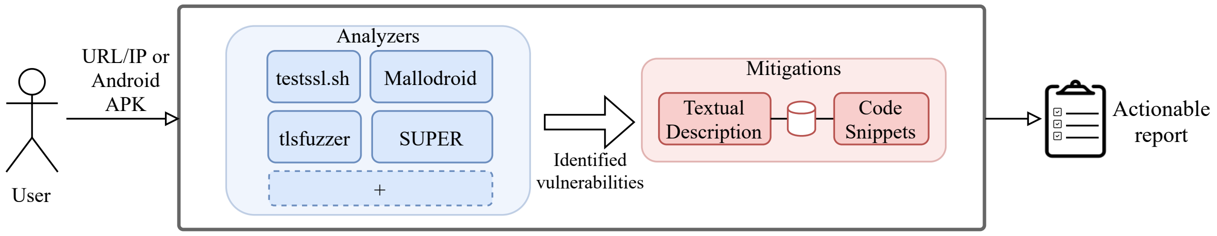


Figure 4: TLSAssistant architecture

MITIGATIONS. It is the core of TLSAssistant and is responsible for the generation of the actionable report as it combines the information retrieved from each TLS analyzer's output with an attack description, a textual description of the mitigation and a code snippet (see Section 4.2).

Since its first iteration, created after the user study, TLSAssistant kept undergoing major changes to enhance both its analysis capabilities and the set of offered features. Among these we can mention:

- **HSTS-related checks.** The introduction of various tests that can detect the lack of proper HSTS [71] configuration and other related checks. These checks led to the detection of a vulnerability in a SaaS used within our organization (see Section 9.2);
- **NGINX mitigations.** The inclusion of new NGINX-specific code snippets that, combined with the Apache ones that were already available in the previous release of the tool, make TLSAssistant able to provide mitigations for 79.47% of the webservers' market share [53];
- **compliance analysis.** The ongoing work on a feature that aims to assist system administrators with the configuration of webservers following technical standards mandated by large organizations (e.g., national public administration services) (see Section 8.3.

8.2 Mitigations Database

To assist system administrators, we have collected in Table 12 the current best practices to mitigate the known vulnerabilities of TLS up to v1.2[5] derived from those listed in [83]. The vast majority of the mitigations is applied by changing some lines in the server's configuration file, others are related to vulnerable/outdated support libraries. The identification of such mitigations is non trivial because the currently available reports (see Figure 1) lack of clear indications on which is the source of misconfiguration.

These collected mitigations compose TLSAssistant's knowledge base and allow the generation of the actionable reports already described in Section 4.2.

8.3 Compliance analysis

During the years, many national cyber security authorities (e.g., USA NIST [84], Italian AGID [85], German BSI [86] and ANSSI [87]) have released technical guidelines [88, 10, 89, 90] to define the use and configuration of TLS for both government and user-facing services. While the existence of a technical documentation can help raising awareness and setting a common security level, each national authority has independently chosen a different requirement level for each configurable element, thus

[5]TLS 1.3 removed a wide set of deprecated ciphers and features thus making some of the listed vulnerabilities no more relevant. However, notice that TLS 1.2 is still widely adopted by a large number of servers.

making a configuration compliant in one state and non-compliant in another. An example of this can be seen when taking into account the acceptance level of TLS 1.1 across different standards. Its usage may be accepted in US and Italy (under specific circumstances [91]) while it is completely forbidden for French and German government services. Moreover, the bureaucratic nature of these documents is two-faced as they can set a security level while being in contrast with latest standards and state-of-the-art assessments. For example, both TLS 1.0 and 1.1 have been officially deprecated in March 2021 [8] but none of the presented standards has been updated to reflect that change. This means that a service compliant under a specific standard can still be affected by multiple vulnerabilities while being considered compliant. In the rest of the section, we discuss how we have extended TLSAssistant to help identify and resolve this kind of compliance problem while guaranteeing an adequate level of security.

8.3.1 Use Cases and Related Work

Agenzia per l'Italia Digital (AgID) [85] is the technical agency of the Italian's Council of Ministers created in 2014. It aims to guarantee the diffusion of information and communication technologies in PAs by fostering innovation and economic growth by implementing the Three-Year Plan for Public Administration [92]. In November 2020, AgID released a technical document outlining a set of recommendations for the implementation and usage of the TLS cryptographic protocol [10]. The document directly covers various parts of the configuration and includes Mozilla's profiles definition [93] to suggest the configuration of a wide set of specific settings (i.e. available ciphers, certificate configuration, TLS extensions and more) a PA should use in citizens-facing websites. The following month, we had the opportunity to evaluate the compliance of an authentication mechanism related to the Italian state (see Section 9.1 for details) against the AgID standard and realized the amount of time and effort required to perform a similar task.

To overcome this problem and assist system administrators with the configuration of webserver com-

Table 12: List of Mitigations for TLS 1.2

Mitigation	**Attack**
Disable RC4	Bar Mitzvah [72] RC4 NOMORE [73]
Disable renegotiation (or use custom library's mitigations)	Renegotiation attack [74]
Disable RSA	ROBOT [75]
Disable TLS compression mechanism	CRIME [15]
Disable HTTP compression mechanism	BREACH [16]
Disable SSLv2	DROWN [76]
Disable SSLv3	POODLE [51]
Disable RSA-MD5 certificate signature	SLOTH [77]
Disable 3DES	Sweet32 [52]
Enable the use of `extended master secret` extension [78]	3SHAKE [3]
Enforce the use of AEAD ciphers (or use libraries' custom mitigations)	Lucky 13 [79]
Enforce `close_notify` alert message usage	TLS Truncation [80]
Use and preload HSTS [81]	SSL Stripping [82]
Ignore self-signed certificates	Accept self-signed certs [13]
Check if the certificate's CN field corresponds to the server's hostname and fully verify the chain of trust	Connecting using a valid but incorrect certificate [13]

plying with technical standards, we started to develop a compliance checking module to TLSAssistant. The new feature currently focuses on two use cases:

- **Comparison** - a user with a deployed webserver wants to check its compliance against a standard. TLSAssistant will list all the differences and provide contextualized actionable hints to eliminate (if possible) or substantially reduce them;
- **Generation** - a user who wants to deploy a new webserver, compliant with a standard, will get a complete configuration that can directly be used to deploy a service that is compliant by-design.

This newer TLSAssistant feature may look similar but significantly differs from currently available tools. We compare with some of the available tools below.

TLS Profiler [94] compares a webserver against Mozilla's profiles [93] and provides a set of bullet points highlighting the differences. Its report does not offer actionable hints and may introduces some confusion as each differing element is shown separately (while all ciphers and protocols are usually handled with a single line in the configuration). Our aim is both to provide actionable snippets and to extend the comparison feature by allowing to check against technical standards;

Discovery [95] is a free server analyzer provided by Cryptosense[6] that compares the webserver against ANSSI, NIST, and other standards. Its report is very bare as it just lists, for each standard, the unmet requirements and redirects the user to each standard's technical documentation. With our checker, we help system administrators in gaining knowledge about what needs to be changed in order to become compliant without having the need to search through scientific or technical literature;

testssl.sh [39] is planning to add a rating template allowing users to define multiple standards and check against them [96]. We will closely follow the progress because, being already included within our set of analyzers, we may benefit from such a feature;

Mozilla's configuration generator [97] is similar to our second use-case but only covers Mozilla profiles [93]. We aim to generate a similar configuration following a model compliant with a full set of requirements (i.e. not only restricted to the Mozilla subset of requirements).

The problem of checking if a website is compliant with one or more standards, and contextually suggest how to reduce the difference between configurations, can be tackled by using different algorithmic solutions. Among these, we exploit a class of constraint solvers—called Boolean solvers—that allow for encoding the compliance problem into a logical problem (by using an appropriate translator) and to map the solution of the latter back to the former. We use this technology to build the first version of the checker, which applies the described use case to the AgID standard for TLS usage. Since Boolean solvers are known to be able to discharge large Boolean problems, this seems to suggest that we can extend such an approach to checking compliance with respect to large and complex standards. In the future, we plan to extend the compliance module by including the capability of checking the compliance with respect to all the major standards such as NIST and BSI. We believe that the approach based on constraint solver lends itself well to such extensions because it consists of encoding the compliance problem into a logical problem that can be discharged by the available Boolean solver.

[6]`https://cryptosense.com`

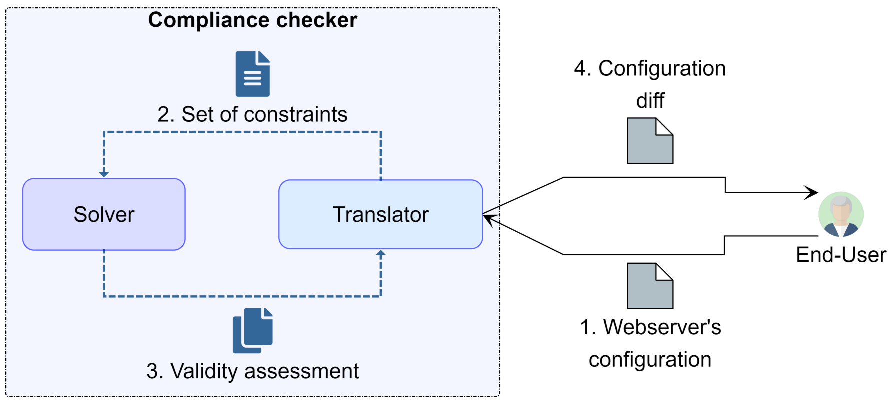


Figure 5: Compliance checker architecture

9 Corporate Case Studies

To evaluate TLSAssistant's efficacy in a corporate context, we have analyzed two real use-case scenarios: the former involving the Italian electronic identity card [98] (Section 9.1) and the latter related to a SaaS used within our organization (Section 9.2). In both scenarios, the outcome shows how running a tool like TLSAssistant can help even expert system administrators to determine if a new deployment contains security misconfigurations and guide them in the patching process.

9.1 CIE 3.0

In a joint collaboration between FBK and IPZS (acronym for "Istituto Poligrafico e Zecca dello Stato")[7], which is the Italian state printing office and mint, we have designed and deployed a mobile authentication mechanism that uses the Italian electronic identity card (CIE 3.0 - Carta d'Identitá Elettronica) [98]. 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 6) is composed of two elements managing the authentication process: an Android app, called CieID [99], able to interact with the CIE and an identity provider (IdP) that authenticates users combining a challenge-response protocol with SAML 2.0 [100]. 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 [101]. The assessment included TLSAssistant's analysis. We discovered that the first release of the infrastructure 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 report was both easy to understand and complete. These features greatly helped the system administrators to quickly fix the vulnerabilities.

[7]https://www.ipzs.it

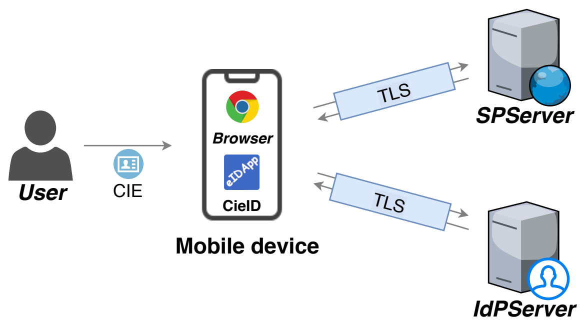


Figure 6: CIE ID infrastructure (simplified)

9.2 Sensitive SaaS

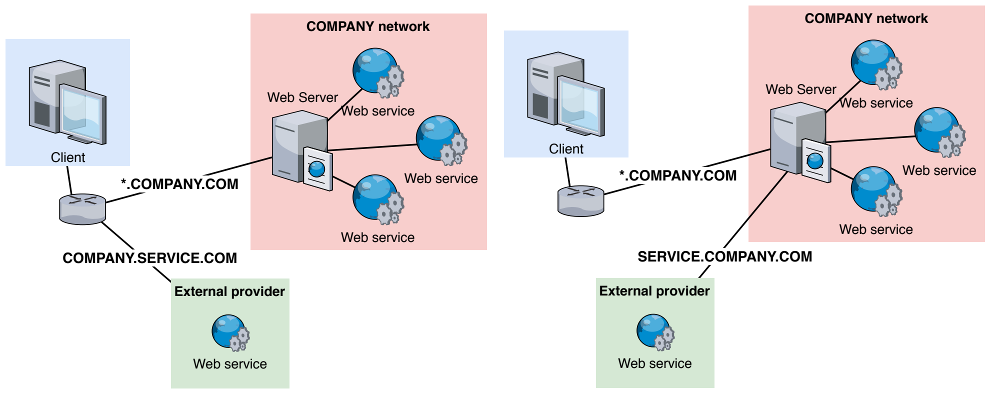
The term *Software as a Service* (SaaS) refers to a cloud distribution model in which the provider offers its services via Internet. By removing the burden to install, maintain and protect the infrastructure from the users' shoulders, these services appeal to big organizations. Being reachable by web 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(a)) requires employees to contact an external website to access the company-customized service (e.g., `COMPANY.SERVICE.com`) while the second (see Figure 7(b)) appears within the company domain (e.g., `SERVICE.COMPANY.com`) 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.

After running TLSAssistant we discovered that both of them were prone to a series of attacks and, among these, the major threat came from the possibility to mount a Stripping attack. An SSL Stripping attack [82] 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 [71] 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 [81], 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.COMPANY.com`), despite the same mitigations, the situation was different. Specifically, eliminating the SSL Stripping vulnerability 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



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

Figure 7: SaaS Integrations

any threat that can be involuntarily inherited.

9.3 Corporate Case Studies Discussion

The findings from these case studies clearly supplement those from the user study (see Section 7). In both cases, we discovered vulnerabilities that require a mitigation less trivial than the ones in the user study (see Section 5). In particular, the first scenario (see Section 9.1) shows that even professional developers working on a newly-designed security sensitive project benefit from the presence of clear and concise mitigations. So, we can speculate that the availability of code snippets allowed a faster patching, in fact two out of three vulnerabilities were not trivial and they would have required a deep study of the available literature to elaborate the correct patch.

The second scenario (see Section 9.2) highlighted that even an already deployed, commercial-grade service lacked an appropriate comprehension on the impact of specific configurations. Our analysis showed how the users' privacy was at stake and how the presence of actionable hints have enabled the system administrators to promptly fix the detected issues. This case study also highlights the need for a higher level of awareness when developing and configuring SaaS, as the attack surface they contribute to create opens to a wide amount of possible misconfigurations and attacks.

10 Conclusions

The usability of automated tools for vulnerability detection is quite a neglected topic. We have designed and conducted a user study aiming at filling this gap and understanding if security reports do meet usability. Our experiments reveal that the usability of reports is largely impacted by the availability of contextualized actionable hints, as they have a positive effect on both the correctness and the time needed to fix a TLS vulnerability. In addition, empirical evidence allows us to formulate a set of lessons learned that pave the way towards improving in general the usability of automated security tools. We can summarize the lesson learnt as follows. The main drawback is that the information provided (such as a flat list of detected problems) is often insufficient to enable users with little experience in security to mitigate a vulnerability. This can be alleviated by adding succinct textual explanations describing both the identified vulnerabilities and how they can be exploited in attacks. Besides reducing the time to fix a vulnerability and increasing the correctness of the applied patches, this approach has the potential to improve both knowledge and capabilities of administrators with less experience and to simplify the maintenance of complex systems. Thus, both the productivity and the security posture of an organization are improved. We used these results to build an open-source tool called TLSAssistant, able to combine state-of-the-art analyzers with a report system that generates actionable mitigations to assist system administrators. Finally, we present two case studies for assessing the adoption of our tool in corporate environment. As future work, we plan to further improve our tool by increasing the amount of supported webservers, building new analysis modules able to check against new vulnerabilities and a dynamic risk assessment system that considers different dimension (e.g., vulnerability type, number of affected virtual hosts, the impact of the vulnerability itself) to guide the user in prioritizing the fixes. We also plan to perform more experiments and understand how the application of suggested mitigations can jeopardize the availability of legacy systems and how participants may behave when presented with patches that require more complex tasks.

Reproducibility

All experimental material, including the vulnerable webservers, slides and questionnaires, together with the experimental data and the assets used for the training are available in the replication package [56].

Acknowledgments

This work has been partially supported by ”Futuro & Conoscenza Srl”, jointly created by FBK and the Italian National Mint and Printing House (IPZS), and by the MIUR “Dipartimenti di Eccellenza: Informatica per Industria 4.0” 2018-2022 grant. We would also like to thank Matteo Rizzi for his work on NGINX-related mitigations.

References

[1] Mozilla Security. Web security cheat sheet, 2018. `https://infosec.mozilla.org/guidelines/web_security` [Online; accessed on March 15, 2022].

[2] J.C. Perez. Ssl: Deceptively simple, yet hard to implement, 2016. `https://blog.qualys.com/product-tech/2016/12/12/ssl-deceptively-simple-yet-hard-to-implement` [Online; accessed on March 15, 2022].

[3] Microsoft-Inria. Triple handshakes considered harmful: Breaking and fixing authentication over tls, 2014. `https://www.mitls.org/pages/attacks/3SHAKE` [Online; accessed on March 15, 2022].

[4] M. Green. A diversion: Beast attack on tls/ssl encryption, 2011. `https://blog.cryptographyengineering.com/2011/09/21/brief-diversion-beast-attack-on-tlsssl/` [Online; accessed on March 15, 2022].

[5] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world. In *Proc. of the 19th ACM conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA*, pages 38–49. ACM, October 2012.

[6] Qualys. Ssl pulse, 2021. `https://www.ssllabs.com/ssl-pulse/` [Online; accessed on March 15, 2022].

[7] Amazon Web Services. Alexa top sites, 2021. `https://aws.amazon.com/alexa-top-sites/` [Online; accessed on March 15, 2022].

[8] K. Moriarty and S. Farrell. Deprecating tlsv1.0 and tlsv1.1, 2021. `https://tools.ietf.org/html/rfc8996` [Online; accessed on March 15, 2022].

[9] S. Manfredi, M. Ceccato, G. Sciarretta, and S. Ranise. Do security reports meet usability? lessons learned from using actionable mitigations for patching tls misconfigurations. In *Proc. of the 16th International*

Conference on Availability, Reliability and Security (ARES'21), Vienna, Austria, pages 1–13. ACM, August 2021.

[10] AgID. Raccomandazioni AGID - TLS e Cipher Suite , 2020. `https://www.agid.gov.it/it/sicurezza/tls-e-cipher-suite` [Online; accessed on March 15, 2022].

[11] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2, August 2008. `http://www.rfc-editor.org/rfc/rfc5246.txt` [Online; accessed on March 15, 2022].

[12] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing known attacks on transport layer security (tls) and datagram tls (dtls), February 2015. `http://www.rfc-editor.org/rfc/rfc7457.txt` [Online; accessed on March 15, 2022].

[13] NowSecure. Fully validate ssl/tls, 2017. `https://books.nowsecure.com/secure-mobile-development/en/sensitive-data/fully-validate-ssl-tls.html` [Online; accessed on March 15, 2022].

[14] Scott Hollenbeck. Transport layer security protocol compression methods, May 2004. `http://www.rfc-editor.org/rfc/rfc3749.txt` [Online; accessed on March 15, 2022].

[15] NIST. Cve-2012-4929, 2012. `https://nvd.nist.gov/vuln/detail/CVE-2012-4929` [Online; accessed on March 15, 2022].

[16] Y. Gluck, N. Harris, and A. Prado. Breach: reviving the crime attack, 2012. `http://breachattack.com/` [Online; accessed on March 15, 2022].

[17] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith. Why do developers get password storage wrong? a qualitative usability study. In *Proc. of the 24th ACM SIGSAC Conference on Computer and Communications Security (CCS'17), Dallas, Texas, USA*, pages 311–328. ACM, October 2017.

[18] A. Naiakshina, A. Danilova, E. Gerlitz, E.V. Zezschwitz, and M. Smith. If you want, i can store the encrypted password": A password-storage field study with freelance developers. In *Proc. of the 37th Conference on Human Factors in Computing Systems (CHI'19), Glasgow, Scotland, United Kingdom*, pages 1–12. ACM, May 2019.

[19] L. Allodi, M. Cremonini, F. Massacci, and W. Shim. Measuring the accuracy of software vulnerability assessments: experiments with students and professionals. *Empirical Software Engineering*, 25:1063–1094, January 2020.

[20] L. Allodi, S. Biagioni, B. Crispo, K. Labunets, F. Massacci, and W. Santos. Estimating the assessment difficulty of cvss environmental metrics: An experiment. In *Proc. of the 4th International Conference on Future Data and Security Engineering (FDSE'17), Ho Chi Minh City, Vietnam*, volume 10646 of *Lecture Notes in Computer Science*, pages 23–39. Springer, Cham, November-December 2017.

[21] K. Labunets, F. Massacci, F. Paci, and L.M.S. Tran. An Experimental Comparison of Two Risk-Based Security Methods. In *Proc. of the 6th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'13), Baltimore, Maryland, USA*, pages 163–172. IEEE, October 2013.

[22] K. Labunets, F. Massacci, F. Paci, S. Marczak, and F.M.D. Oliveira. Model comprehension for security risk assessment: an empirical comparison of tabular vs. graphical representations. *Empirical Software Engineering*, 22(6):3017–3056, February 2017.

[23] R. Scandariato, J. Walden, and W. Joosen. Static analysis versus penetration testing: A controlled experiment. In *Proc. of the 24th International Symposium on Software Reliability Engineering (ISSRE'13), Pasadena, California, USA*, pages 451–460. IEEE, November 2013.

[24] M. Ceccato and R. Scandariato. Static analysis and penetration testing from the perspective of maintenance teams. In *Proc. of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'16), Ciudad Real, Spain*, pages 1–6. ACM, September 2016.

[25] M. Ceccato, M.D. Penta, Jasvir Nagra, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. The effectiveness of source code obfuscation: An experimental assessment. In *Proc. of the 17th International Conference on Program Comprehension (ICPC'09), Vancouver, British Columbia, Canada*, pages 178–187. IEEE, May 2009.

[26] M. Ceccato, M.D. Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engi-*

neering, 19(4):1040–1074, August 2014.

[27] N. Hänsch, A. Schankin, M. Protsenko, F. Freiling, and Z. Benenson. Programming experience might not help in comprehending obfuscated source code efficiently. In *Proc. of the 14th Symposium on Usable Privacy and Security (SOUPS'18), Baltimore, Maryland, USA*, pages 341–356. USENIX Association, August 2018.

[28] A. Viticchie, L. Regano, M. Torchiano, C. Basile, M. Ceccato, P. Tonella, and R. Tiella. Assessment of Source Code Obfuscation Techniques. In *Proc. of the 16th International Working Conference on Source Code Analysis and Manipulation (SCAM'16), Raleigh, North Carolina, USA*, pages 11–20. IEEE, October 2016.

[29] A. Viticchié, L. Regano, C. Basile, M. Torchiano, M. Ceccato, and P. Tonella. Empirical assessment of the effort needed to attack programs protected with client/server code splitting. *Empirical Software Engineering*, 25(1):1–48, July 2020.

[30] Y. Acar, M. Backes, S. Fahl, D. Kim, M.L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *Proc. of the 33rd IEEE Symposium on Security and Privacy (SP'16), San Jose, California, USA*, pages 289–305. IEEE, May 2016.

[31] P.L. Gorski, L.L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse. In *Proc. of the 14th Symposium on Usable Privacy and Security (SOUPS'18), Baltimore, Maryland, USA*, pages 265–281. USENIX Association, August 2018.

[32] K. Krombholz, K. Busse, K. Pfeffer, M. Smith, and E.V. Zezschwitz. "if https were secure, i wouldn't need 2fa" - end user and administrator mental models of https. In *Proc. of the 36th IEEE Symposium on Security and Privacy (SP'19), San Francisco, California, USA*, pages 246–263. IEEE, May 2019.

[33] K. Krombholz, W. Mayer, M. Schmiedecker, and E. Weippl. I have no idea what i'm doing" - on the usability of deploying HTTPS. In *Proc. of the 26th USENIX Security Symposium (SEC'17), Vancouver, British Columbia, Canada*, pages 1339–1356. USENIX Association, August 2017.

[34] M. Bernhard, J. Sharman, C.Z. Acemyan, P. Kortum, D.S. Wallach, and J.A. Halderman. On the usability of https deployment. In *Proc. of the 37th 2019 Conference on Human Factors in Computing Systems (CHI'19),*, pages 1–10. ACM, May 2019.

[35] C. Tiefenau, M. Häring, K. Krombholz, and E.V. Zezschwitz. Security, availability, and multiple information sources: Exploring update behavior of system administrators. In *Proc. of the 16th Symposium on Usable Privacy and Security (SOUPS'20), Virtual Conference*, pages 239–258. USENIX Association, August 2020.

[36] F. Li, L. Rogers, A. Mathur, N. Malkin, and M. Chetty. Keepers of the machines: Examining how system administrators manage software updates for multiple machines. In *Proc. of the 15th Symposium on Usable Privacy and Security (SOUPS'19), Santa Clara, CA, USA*, pages 272–288. USENIX Association, August 2019.

[37] M. Kolybabi and G. Lawrence. ssl-enum-ciphers, 2020. `https://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html` [Online; accessed on March 15, 2022].

[38] rbsec. sslscan, 2017. `https://github.com/rbsec/sslscan/releases/tag/1.11.11-rbsec` [Online; accessed on March 15, 2022].

[39] D. Wetter. /bin/bash based ssl/tls tester: testssl.sh, 2021. `https://testssl.sh` [Online; accessed on March 15, 2022].

[40] Qualys. Ssl server test, 2021. `https://www.ssllabs.com/ssltest/` [Online; accessed on March 15, 2022].

[41] A. Diquet. Github: sslyze, 2021. `https://github.com/nabla-c0d3/sslyze` [Online; accessed on March 15, 2022].

[42] MITRE. Cve, 2021. `https://cve.mitre.org` [Online; accessed on March 15, 2022].

[43] Forum of Incident Response and Security Teams, Inc. Common vulnerability scoring system, 2021. `https://www.first.org/cvss/` [Online; accessed on March 15, 2022].

[44] M. Cartwright. Book review: Experimentation in software engineering: An introduction. by claes wohlin, per runeson, martin höst, magnus c. ohlsson, björn regnell and anders wesslén. kluwer academic publishers, 1999. *Software Testing, Verification and Reliability*, 11(3):198–199, September 2001.

[45] M. Ceccato, A. Marchetto, L. Mariani, C.D. Nguyen, and P. Tonella. Do automatically generated test cases make debugging easier? an experimental assessment of debugging effectiveness and efficiency. *ACM Transactions on Software Engineering and Methodology*, 25(1):1–38, December 2015.

[46] J.M. Six and R. Macefield. How to determine the right number of participants for usability studies, 2016. `https://www.uxmatters.com/mt/archives/2016/01/how-to-determine-the-right-number-of-participants-for-usability-studies.php` [Online; accessed on March 15, 2022].

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

[48] M. Svahnberg, A. Aurum, and C. Wohlin. Using students as subjects - an empirical evaluation. In *Proc. of the 2nd ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'08), Kaiserslautern, Germany*, pages 288–290. ACM, October 2008.

[49] I. Salman, A.T. Misirli, and N. Juristo. Are students representatives of professionals in software engineering experiments? In *Proc. of the 37th IEEE International Conference on Software Engineering (ICSE'15), Florence, Italy*, pages 666–676. IEEE, May 2015.

[50] Association for Computing Machinery. Acm code of ethics and professional conduct, 2018. `https://www.acm.org/binaries/content/assets/about/acm-code-of-ethics-booklet.pdf` [Online; accessed on March 15, 2022].

[51] B. Möller, T. Duong, and K. Kotowicz. This poodle bites: Exploiting the ssl 3.0 fallback, 2014. `https://www.openssl.org/~bodo/ssl-poodle.pdf` [Online; accessed on March 15, 2022].

[52] K. Bhargavan and G. Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and openvpn. In *Proc. of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16), Vienna, Austria*, pages 456–467. ACM, October 2016.

[53] Datanyze. Web and application servers market share report, 2021. `https://www.datanyze.com/market-share/web-and-application-servers` [Online; accessed on March 15, 2022].

[54] Datanyze. Openssl market share and competitor report, 2021. `https://www.datanyze.com/market-share/other-it-infrastructure-software` [Online; accessed on March 15, 2022].

[55] ISO. ISO 9241. Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts. Technical Report ISO 9241-11, International Organization for Standardization, 2018.

[56] S. Manfredi, M. Ceccato, G. Sciarretta, and S. Ranise. Replication Package: do security reports meet usability? lessons learned from using actionable mitigations for patching tls misconfigurations, 2021. `https://st.fbk.eu/complementary/ETACS2021` [Online; accessed on March 15, 2022].

[57] A. Kühberger, A.Fritz, E. Lermer, and T. Scherndl. The significance fallacy in inferential statistics. *BMC research notes*, 8(84):1–9, March 2015.

[58] J.L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury Press, 2007.

[59] M. Ceccato, M.D. Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19(4):1040–1074, February 2014.

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

[61] R.J. Grissom and J.J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2005.

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

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

[64] B. Saefken, T. Kneib, C.V. Waveren, and S. Greven. A unifying approach to the estimation of the conditional akaike information in generalized linear mixed models. *Electronic Journal of Statistics*, 8(1):201–225, February 2014.

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

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

[67] Security & Trust Research Unit. Tlsassistant, July 2021. `https://github.com/stfbk/tlsassistant` [Online; accessed on March 15, 2022].

[68] H. Kario. SSL and TLS protocol test suite and fuzzer: tlsfuzzer, 2021. `https://github.com/tlsfuzzer/tlsfuzzer` [Online; accessed on March 15, 2022].

[69] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith. Why eve and mallory love android: An analysis of android ssl (in)security. In *Proc. of the 2012 ACM Conference on Computer and Communications Security (CCS'12), Raleigh, North Carolina, USA*, pages 50–61. ACM, October 2012.

[70] SUPERAndroidAnalyzer. Github: Secure, unified, powerful and extensible rust android analyzer, 2018. `https://github.com/SUPERAndroidAnalyzer/super` [Online; accessed on March 15, 2022].

[71] J. Hodges, C. Jackson, and A. Barth. Http strict transport security (hsts), November 2012. `http://www.rfc-editor.org/rfc/rfc6797.txt` [Online; accessed on March 15, 2022].

[72] I. Mantin. Bar Mitzvah Attack: Breaking SSL with 13-Year Old RC4 Weakness, 2015. `https://www.blackhat.com/docs/asia-15/materials/asia-15-Mantin-Bar-Mitzvah-Attack-Breaking-SSL-With-13-Year-Old-RC4-Weakness-wp.pdf` [Online; accessed on March 15, 2022].

[73] M. Vanhoef and F. Piessens. RC4 NOMORE (Numerous Occurrence MOnitoring & Recovery Exploit), 2015. `https://www.rc4nomore.com/` [Online; accessed on March 15, 2022].

[74] SecurityLearn. SSL Attacks, 2013. `http://www.securitylearn.net/tag/ssl-renegotiation-attack/` [Online; accessed on March 15, 2022].

[75] H. Böck, J. Somorovsky, and C. Young. Return of bleichenbacher's oracle threat (robot). In *Proc. of the 27th USENIX Security Symposium (SEC'18), Baltimore, Maryland, USA*, pages 817–849. USENIX Association, August 2018.

[76] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J.A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS using sslv2. In *Proc. of the 25th USENIX Security Symposium (SEC'16), Austin, Texas, USA*, pages 689–706. USENIX Association, August 2016.

[77] K. Bhargavan and G. Leurent. Transcript collision attacks: Breaking authentication in tls, ike, and ssh. In *Proc. of the 23rd Network and Distributed System Security Symposium (NDSS'16), San Diego, California, USA*, pages 1–17. Internet Society, February 2016.

[78] IETF. Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension, 2015. `https://tools.ietf.org/html/rfc7627` [Online; accessed on March 15, 2022].

[79] N.J.A. Fardan and K.G. Paterson. Lucky thirteen: Breaking the tls and dtls record protocols. In *Proc. of the 30th IEEE Symposium on Security and Privacy (SP'13), Berkeley, California, USA*, pages 526–540. IEEE, May 2013.

[80] B. Smyth and A. Pironti. Truncating tls connections to violate beliefs in web applications. In *Proc. of the 7th USENIX Workshop on Offensive Technologies (WOOT'13), Washington, USA*, pages 1–9. USENIX Association, August 2013.

[81] Google Open Source. HSTS Preload List, 2020. `https://opensource.google.com/projects/hstspreload` [Online; accessed on March 15, 2022].

[82] M. Marlinspike. New Tricks For Defeating SSL In Practice, 2009. `https://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf` [Online; accessed on March 15, 2022].

[83] S. Manfredi, S. Ranise, and G. Sciarretta. Lost in tls? no more! assisted deployment of secure tls configurations. In *Proc. of the 33rd IFIP Annual Conference on Data and Applications Security and Privacy (DBSec'19), Charleston, South Carolina, USA*, volume 11559 of *Lecture Notes in Computer Science*, pages 201–220. Springer, Cham, July 2019.

[84] U.S. Department of Commerce. National Institute of Standards and Technology, 2021. `https://www.nist.gov` [Online; accessed on March 15, 2022].

[85] AgID. Agenzia per l'Italia Digitale, 2021. https://www.agid.gov.it [Online; accessed on March 15, 2022].

[86] BSI. Federal Office for Information Security, 2021. https://www.bsi.bund.de/EN/Home/home_node.html [Online; accessed on March 15, 2022].

[87] Secretariat-General for National Defence and Security. Agence nationale de la sécurité des systèmes d'information, 2021. https://www.ssi.gouv.fr/en/ [Online; accessed on March 15, 2022].

[88] NIST. NIST SP 800-52 Rev. 2, August 2019. https://csrc.nist.gov/News/2019/nist-publishes-sp-800-52-revision-2 [Online; accessed on March 15, 2022].

[89] BSI. BSI TR-02102-2, 2021. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.pdf [Online; accessed on March 15, 2022].

[90] ANSSI. Recommandations de sécurité relatives à TLS, 2020. https://www.ssi.gouv.fr/particulier/guide/recommandations-de-securite-relatives-a-tls [Online; accessed on March 15, 2022].

[91] S. Bradner. Key words for use in rfcs to indicate requirement levels, 1997. https://tools.ietf.org/html/rfc2119 [Online; accessed on March 15, 2022].

[92] AgID. The Three-Year Plan for ICT in the Public Administration, 2020. https://pianotriennale-ict.italia.it/en/ [Online; accessed on March 15, 2022].

[93] Mozilla Security. Server side tls, 2020. https://wiki.mozilla.org/index.php?title=Security/Server_Side_TLS [Online; accessed on March 15, 2022].

[94] A. Diquet. GitHub: tlsprofiler, 2020. https://github.com/danielfett/tlsprofiler [Online; accessed on March 15, 2022].

[95] Cryptosense. Discovery, 2020. https://discovery.cryptosense.com [Online; accessed on March 15, 2022].

[96] D. Wetter. Rating template, 2018. https://github.com/drwetter/testssl.sh/issues/1108 [Online; accessed on March 15, 2022].

[97] Mozilla Security. Mozilla ssl configuration generator, 2021. https://ssl-config.mozilla.org/ [Online; accessed on March 15, 2022].

[98] M. Dell'Interno. Carta di identitá elettronica, 2020. https://www.cartaidentita.interno.gov.it [Online; accessed on March 15, 2022].

[99] Istituto Poligrafico e Zecca dello Stato S.p.A. CieID, 2020. https://play.google.com/store/apps/details?id=it.ipzs.cieid [Online; accessed on March 15, 2022].

[100] M. Dell'Interno. Accesso ai servizi in rete mediante la CIE 3.0, 2020. https://www.confindustria.ge.it/images/CIE3.0-ManualeSP.pdf [Online; accessed on March 15, 2022].

[101] M. Eichholtzer. Italy - eID, 2019. https://ec.europa.eu/cefdigital/wiki/display/EIDCOMMUNITY/Italy+-+eID [Online; accessed on March 15, 2022].

Author Biography

Salvatore Manfredi is a Ph.D. Candidate at the Security & Trust research unit in Fondazione Bruno Kessler in a joint scholarship with University of Genoa; working on Identity Management for Digital Financial Infrastructures. He graduated from University of Trento with a thesis focused on assisting users in securing TLS deployment. In the last years he collaborated on the security analysis of the authentication scheme involving the Italian electronic identity card (CIE 3.0) and integrated TLSAssistant within the Horizon 2020-funded FINSEC project.

Mariano Ceccato is (tenure track) Assistant Professor in the Computer Science department in University of Verona. Until 2019 he was tenured researcher in the Security & Trust research unit in Fondazione Bruno Kessler, Trento, where he was principal investigator of several publicly funded research projects. Mariano received the PhD in Computer Science from the University of Trento in 2006, and the Master Degree in Software Engineering from University of Padova in 2003. He was recently visiting research scientist in University of Luxembourg. He is author or co-author of more than 70 research papers published in international journals and conferences/workshops. His research interests include security testing, penetration testing, code hardening and empirical studies.

Giada Sciarretta is a tenure track researcher of the Security & Trust research unit of Fondazione Bruno Kessler. She obtained her MSc in mathematics and received her PhD in computer science at the University of Trento in 2012 and 2018, respectively. Her research focuses on digital identity with a specialization in the design, security (with informal and formal specification) and risk assessment of access delegation and single sign-on protocols (e.g., OAuth 2.0 and OpenID Connect), multi-factor authentication and fully-remote enrollment procedures.

Silvio Ranise is Full Professor of Computer Science at the University of Trento and Director of the Center for Cybersecurity of the Fondazione Bruno Kessler In Trento. Before, he held a research position at INRIA in France and was visiting professor at the University of Milan, Italy. His research interests are digital identity, the security of cloud-edge solutions, and applied cryptography.