Usability Insights from Establishing TLS Connections*

Lydia Kraus[1][0000−0002−1387−3578], Matěj Grabovský[2], Martin Ukrop[2][0000−0001−8110−8926], Katarína Galanská[2], and Vashek Matyáš[2]

[1] Institute of Computer Science
Masaryk University, Brno, Czechia
lydia.kraus@mail.muni.cz
[2] Centre for Research on Cryptography and Security
Masaryk University, Brno, Czechia
{mgrabovsky,mukrop,galanska}@mail.muni.cz, matyas@fi.muni.cz

Abstract. TLS is crucial to network security, but TLS-related APIs have been repeatedly shown to be misused. While existing usable security research focuses on cryptographic primitives, the specifics of TLS interfaces seem to be under-researched. We thus set out to investigate the usability of TLS-related APIs in multiple libraries with a focus on identifying the specifics of TLS. We conducted a three-fold exploratory study with altogether 60 graduate students comparing the APIs of three popular security libraries in establishing TLS connections: OpenSSL, GnuTLS, and mbed TLS. We qualitatively analyzed submitted reports commenting on API usability and tested created source code. User satisfaction emerged as an interesting, potentially under-researched theme as all APIs received both positive and negative reviews. Abstraction level, error handling, entity naming, and documentation emerged as the most salient usability themes. Regarding functionality, checking for revoked certificates was especially complicated and other basic security checks seemed not easy as well. In summary, although there were conflicting opinions on both the interface and documentation of the libraries, several usability issues were shared among participants, forming a target for closer inspection and subsequent improvement.

Keywords: API usability · TLS · User satisfaction · Usable security

1 Introduction

While the reliance on TLS for secure communications keeps growing [14], it has been repeatedly shown that real-world applications often contain vulnerabilities due to the misuse of TLS libraries [9,10,12,20]. Moreover, it has been demonstrated that current security and cryptographic interfaces are hard to

* This is the author's version. The final publication will be available in Springer's *International Federation for Information Processing book series.*

use [1,11,17,23] and that writing code that is both secure and functional is difficult even for professional developers [4,12,19]. It is thus clear that the usability of security APIs is paramount in helping programmers develop secure applications.

The primary means of studying usability is conducting controlled experiments with the intended end users [27], i.e., application developers in our case of security interfaces. However, since recruiting IT professionals is labourious [3,4,8], we have chosen to perform our study with graduate students, that is, future professional developers. Recent research in usable security shows this subpopulation to be adequate for exploratory studies [2,4,24,25,31].

Although experimental developer studies in usable security tend to aim broadly at cryptographic libraries, few have, to our knowledge, focused specifically on TLS programming interfaces. We have decided to explore this important area using qualitative methods, which seem attractive in this respect, given their potential to generate interesting hypotheses from comparatively small samples. Moreover, the benefits of qualitative research for usable security have been recognized in several recent studies [17,19,24,25,16,18].

Our paper describes the design and results of an exploratory study, conducted in three rounds and investigating the usability of TLS-related APIs, specifically of three popular libraries: OpenSSL, GnuTLS, and mbed TLS.

2 Methodology

To investigate the usability of TLS interfaces, we conducted a programming experiment with IT security students, asking participants to establish a TLS connection using the OpenSSL, GnuTLS, and mbed TLS libraries. We designed the study in an open manner with no preset hypotheses in an attempt to capture the potentially unique and disparate issues arising from using TLS-related APIs. Due to the qualitative nature of the study and the so far little researched phenomenon of TLS API usability, we iteratively analyzed the data and increased our sample size until the data reached saturation.

2.1 Setting and Participants

The study was conducted at the Faculty of Informatics at Masaryk University within a small master-level course called *Secure Coding Principles and Practices*. The study ran in three subsequent runs of the course (autumn 2018/autumn 2019/spring 2021[3]). The course aims to explain typical security issues related to secure coding and help students design applications in a more secure way. Students enrolling in the course are required to have at least a basic knowledge in applied cryptography, IT security, and programming in C or C++. The course is compulsory for the master specialization of *Information Security*.

The experimental task was set as a homework in the week discussing usability of security APIs. The students were told that the submitted data will be

[3] Note that the course was moved to spring in the academic year 2020/21 and did thus not run in autumn 2020.

anonymized and analyzed within local usable security research. Sharing their data (anonymously) for research purposes was advertised as optional (and not influencing the grading in any way). If they chose to opt out, we would exclude their data after grading the homework. The instructor made sure to explain in the class what data will be used, how it will be used, and that opting out of the research will not have any consequences for this or any other course taken at the faculty. The instructor also explained that no one else except him would know who opted out and that the data processing will be done solely on anonymized data. There was no compensation for joining the research. The study was part of a project on TLS usability that received approval from the institutional ethics board.

The particular seminar on the usability of security APIs contained a short lecture-style introduction to usable security for both end-users and IT professionals (cca 60 minutes). It included multiple examples of (un)usability issues including TLS in non-browser software [12], deploying HTTPS on Apache servers [19] and ideas on developer-resistant cryptography [7]. This was followed by an in-lecture activity: In teams of 2–3, students were asked to design an intentionally *unusable* C-like API for encryption/decryption routines (to actively engage in the usability issues of APIs).

Altogether, there were 60 students participating in the study. In 2018, there were 13 students enrolled in the course. Nine completed the homework within the assigned week. Two of them were exchange students. In 2019, there were 32 students enrolled in the course. 26 completed the homework within the assigned week. Nine of them were exchange students. In 2021, there were 33 students enrolled in the course. 25 of them completed the homework within the assigned week. None of them was an exchange student. The 2021 sample was security-wise the most experienced one, with 18 out of 25 (72%) participants who had taken a related course which also handles OpenSSL – the *Laboratory of security and applied cryptography* – previously or in parallel, while this applied to 15 of 26 participants (57.7%) in 2019 and five out of nine participants (55.6%) in 2018. In all three years, none of the students who submitted the homework opted out of the research.

2.2 Experimental Task

In all three rounds of the experiment, the core of the task was to implement a simple TLS client in C using three different TLS libraries, in no predefined order: OpenSSL (v1.1.1 or above), GnuTLS (v3.5.6 or similar), and mbed TLS (v2.16.0 or similar). Apart from the implementation, we asked for a short written report comparing the differences of the APIs from the point of usability and usable security. Code skeletons with library initialization calls and working Makefiles were provided for each library to speed up the development.

We chose OpenSSL [28] as it is one of the most popular cryptographic libraries for generating keys and certificates [26]. GnuTLS [13], also quite widely used, was preferred to other alternatives since we cooperate with the maintainer and thus have a higher chance of incorporating improvements based on this research

upstream. Lastly, mbed TLS [22] (formerly PolarSSL) was chosen as its API seems rather different from both OpenSSL and GnuTLS. All three libraries have a rather low-level TLS interface, especially in comparison with TLS libraries in higher-level languages such as Python[4], Java[5], Go[6], Javascript[7], or Rust[8].

The full task specification was to establish a TLS connection with the server *www.example.com*, check its certificate, enforce the minimal version of TLS 1.2 and gracefully close the connection. The certificate check was required to include at least the expiration date, server hostname match, the validity of the chain, and the revocation status (either by CRL or OCSP) using the default OS certificate store as the trust anchor. The participants were advised to use BadSSL [5] for debugging the implementation.

The report, conceived as a free-form written reflection to be submitted as a separate PDF file, was required to be at least one page long and contain five to ten specific points comparing the used libraries. We refrained from using existing scales for usability evaluation due to the exploratory nature of the study. Instead, participants were given a set of inspirational questions to illustrate the kind of reflection we sought. These were selected from an existing usability questionnaire for security APIs [32] based on their relevance to the task. The final set of questions can be seen at https://crocs.fi.muni.cz/public/papers/ifipsec2022.

2.3 Data Collection and Processing

For each participant, we collected the source code and the written report. Each implementation was then analyzed from the point of task compliance (functionality). We tested if the code compiled, if it succeeded in connecting to *example.com* and other valid domains, and how it handled selected invalid certificates on *badssl.com* subdomains [5]. The handling of revoked certificates was further checked on *revoked.grc.com*. If the program failed to compile, we first manually inspected the code and tried to identify the cause of the failure. In case of minor errors (such as a wrong set filepath), we fixed the error and continued. If the program still failed to compile or connect to one of the valid domains, we considered it failing and did not proceed with further tests. If it established a connection with any of the "defective" hosts without any errors, we considered it failing as well.

The submitted reports were analyzed using inductive coding. Firstly, two researchers processed the data of the first round (from the year 2018) using open coding [29]. Multiple codes naturally arose from the structure of the task (the questions given as inspiration). After the open coding, the researchers discussed the created codes, looked for reoccurring patterns in the data (axial coding), and consolidated a common codebook. To ensure analysis reliability and consistency, a third independent coder then coded all the reports using the codebook

[4] docs.python.org/3/library/ssl.html
[5] https://cr.openjdk.java.net/ iris/se/11/latestSpec/api/java.base/javax/net/ssl/SSLSocket.html
[6] https://pkg.go.dev/crypto/tls
[7] https://nodejs.org/api/tls.html
[8] https://docs.rs/rustls/latest/rustls/

created by the first two researchers. We calculated interrater agreement (Cohen's $\kappa = 0.62$, $p < .001$), which showed to be substantial (according to Landis and Koch [21]). We then proceeded with coding the data from the second and third rounds (years 2019 and 2021). Few additional codes emerged in the second round. In the third round, no new codes emerged.

3 Results

In this section, we summarize the results of our analysis of the reports and functional testing of the submitted programs. Participants from the first round (year: 2018) were assigned three-digit numbers starting with *1*, while participants from the second round (year: 2019) were assigned numbers starting with *2*. Participants from the third round (year: 2021) were assigned numbers starting with *3*. Although we had provided the participants with a set of inspirational questions (see https://crocs.fi.muni.cz/public/papers/ifipsec2022), many of them preferred to give their own views on the libraries. In the following, we report the most salient themes of the reports: the diversity of library preferences and the most important usability factors with documentation and code snippets in a separate subsection due to the vast amount of comments concerning specifically this usability dimension.

3.1 Diverse Library Preferences

Following the analysis and coding of the submitted reports for all rounds (referring to altogether 60 participants), we set out to determine significant trends among the participants. Most surprisingly, no single favorite library emerged from the reports.

Although not explicitly asked to, several participants ordered the libraries in some order of preference. Each of the libraries was ranked first by the same amount of participants: four out of 60 participants ranked OpenSSL as their favorite library, while four participants preferred GnuTLS the most. Four other participants indicated that mbed TLS would be their library of choice. Apart from the rankings, participants also expressed positive and negative assessments of each library. For each of the libraries and each of the rounds, we will shortly provide an example of positive and negative quotes to illustrate the dichotomy of opinions that arose.

Note, however, that the composition of rankings and preferences differed between the rounds. In the third round, OpenSSL was neither ranked as a favorite library by any of the participants nor did it receive a positive overall assessment. On the contrary, GnuTLS did not receive a negative overall assessment in both, the second and the third round. Mbed TLS ranged between the other two libraries, with receiving both, positive and negative assessment in the first and second round, but not receiving an individual negative assessment in the third round. In round two and three, there were, however, participants who collectively assessed all three libraries negatively.

OpenSSL

"I found this API the easiest one to work with, but that might be just because I've experienced it before." (P101, OpenSSL)

"[OpenSSL] is very difficult to work with. The API is huge [...], many methods are generic and overall the API is quite low level. A lot of things have to be programmed manually [...]" (P104, OpenSSL)

"If I should choose just one library I'll use OpenSSL because of IMHO it's more common than others, a powerful command-line tool and there is good documentation as well." (P211, OpenSSL)

"Personally, this library [OpenSSL] had the worst API of these 3, considering mainly documentation and lack of examples." (P202, OpenSSL)

"The OpenSSL API was making me suicidal (like not really, but it is terrible)." (P308, OpenSSL)

GnuTLS

"The simplest library from given list. Fast development. Good documentation [...] Nice example programs in the package with sources." (P102, GnuTLS)

"The worst to work with was surely GnuTLS. It's required to create network connection manually before using it. Also most of the stuff has to be manually set, and documentation is not good." (P106, GnuTLS)

*"In my opinion, the most successful approach was done in **gnutls**. The validity of the chain was checked by **gnutls** itself as well as the correctness of the hostname."* (P212, GnuTLS)

"Maybe this [code length] is the main reason I liked the GnuTLS the most." (P303, GnuTLS)

mbed TLS

"I liked this one the most. It feels more "higher level" than OpenSSL [...] I also liked the documentation the most, I felt like I can find most of the answers quite quickly [...]" (P107, mbed TLS)

"Really hard to find example programs. Took lot of time to implement assignment. There is documentation, but for newbie really hard to find what required, short explanations for functions." (P102, mbed TLS)

"I have to point out that i found embed easiest to use and to setup" (P220, mbed TLS)

"I don't like EmbedTLS because there were no example in its documentation, and I had to use example code because they have only auto-generated doxygen." (P222, mbed TLS)

"I kind of liked the MbedTLS mainly because of the tutorial for the TLS client." (P303, mbed TLS)

3.2 Considered Usability Factors

While a clear list of usability factors emerged from the qualitative coding of the reports, we encountered a wide range of reasons for favoring one library over another. For example, documentation was used to argue for both liking and disliking it by different participants. Other reasons reflect what has already been pointed to by related work (see Section 5): helpfulness and availability of code examples, ease of use and readability of the documentation, and the API's level of abstraction have a great impact on the overall usability. We describe these and other usability factors that were mentioned in the following paragraphs.

Abstraction Level: In the first and third round of the study, many participants generally commented on the abstraction level of the libraries (only few commented it in round 2). Opinions diverged with some participants finding the abstraction levels appropriate — *"well encapsulated, as expected"* (P105, GnuTLS), *"appropriate"* (P303, GnuTLS), *"more encapsulated than the others"* (P101, mbed TLS), *"appropriate"* (P305, mbed TLS), *"best abstraction and usage"* (P106, OpenSSL), *"not that bad, but also not great"* (P308, OpenSSL). Others criticized the abstraction level of single libraries— *"[GnuTLS] seems to be more lower level than the previous ones"* (P107, GnuTLS), *"lower level than what I would expect for such a library"* (P307, OpenSSL), *"lower but [...] understandable if it is mainly for embedded devices"* (P303, mbed TLS), while even others were not satisfied with the abstraction level of all libraries— *"they mix low-level affairs (such as TCP sockets) with high-level interfaces (SNI, OCSP, certificate verification) and, most importantly, force the developer to write critical code by hand"* (P103, all libraries), *"There is less abstraction in each of the APIs than I would desire, even for C. What seemed like a mainstream security task turned into having to manually request everything."* (P311, all libraries).

In all three rounds, several participants specifically criticized the fact that they had to handle the socket connection manually in GnuTLS: *"when you want to connect to the specific port, you need to use your own socket and you need to implement the connection by yourself, which is extremely annoying,"* (P101, GnuTLS); *"In this part, GnuTLS was the worst of these 3, considering I had to manually open socket, fill the structs with information like the type of the socket, hostname, port and then associate the created socket with my session"* (P202, GnuTLS); *"But in gnutls, I had to go all the way down in the abstraction to the system call level to create a socket, perform a lookup of the ip address and finally create a connection since it provided no builtin way of doing this (at least not one I could find)."* (P312, GnuTLS)

Similarly, in all three rounds, several participants were confused by the OpenSSL BIO system: *"I found some things confusing: for example the BIO and SSL structs"* (P107), *"Many times I had some confusions regarding the BIO or SSL structures and functions"* (P224), *"First time i saw this, I had no idea what BIO was"* (P308). Thereby, developers are obviously left alone to realize that they have *"to understand the whole BIO object machinery, in order to correctly communicate with a server"* (P225), while *"there was no information about the*

fact that I need to create an underlying BIO for the TSL connection (I had to deduce this from the fact that the SSL object has no setter for ports)" (P307).

Error Handling and Return Values: While error reporting was mentioned multiple times, there was no consistent pattern in the reports. It included both positive comments: *"Thanks to the error handling, I could stop with implementation at any time and quickly verify that there's no problem so far."* (P101, OpenSSL), *"I found very helpful the error messages in gnutls."* (P215, GnuTLS); as well as negative ones: *"issue regarding not showing enough details about reasons for failing the verification was not resolved"* (P102, GnuTLS), *"this library was the worst thing to debug from the group"* (P310, OpenSSL).

In all three rounds of the study, OpenSSL was criticized for its inconsistent return values: *"Also, it does not match the typical bash/C convention for returned success/error code (usually 0 is success, but in OpenSSL 1 is success)"* (P104, OpenSSL); *"Semantics of return values is not consistent."* (P214, OpenSSL) *"For the whole process I had to keep looking up return values of functions in the documentation, because they differ for each call."* (P307, OpenSSL).

Entity Naming: Names of functions, parameters and constants were perceived by many positively across the libraries: *"[OpenSSL has] in the most of cases well named structures/types"* (P108, OpenSSL); *"very clear names for the functions"* (P219, GnuTLS); *"Usually the names are fine [...]"* (P302, all libraries).

Yet, for all of the libraries, several participants also criticized the naming and some of them pointed out concrete examples where naming is ambiguous and calls for improvement:

> *"Openssl – SSL_write + SSL_write_ex, SSL_read + SSL_read_ex felt a little too similar, the distinction is a little unclear"* (P301, OpenSSL)

> *"**Similar yet different** Another problem I found is with OpenSSL, where the usage of CTX is different between SSL_CTX and X509_STORE _CTX."* (P326, OpenSSL)

> *"Just looking at the list of the function gives stuff like:*
> – *gnutls_pkcs11_privkey_generate*
> – *gnutls_pkcs11_privkey_generate2*
> – *gnutls_pkcs11_privkey_generate3*
> *This pattern is present in different places and some others are a shame as well. With 1116 function beginning by gnutls... on my system, that's already complicated enough to find the one I need to put 3 times the same name with a number at the end."* (P209, GnuTLS)

> *"Another thing was setting the minimal version of TLS to 1.2. Parameters for the function that provides this, which are* MBEDTLS_SSL_MAJOR _VERSION_3 *and* MBEDTLS_SSL_MINOR_VERSION_3 *are highly non intuitive."* (P101, mbed TLS)

3.3 Documentation and Code Samples

In all three rounds of the study, many of the comments in the written report were concerning documentation and code samples in it. To keep the programming task as realistic as possible, we intentionally left it to the participants to search for appropriate documentation. They were thus free to use the sources available on the official API websites or any other sources available through the Internet. For each library, some participants positively assessed the found documentation, while others criticized it as a whole or in certain aspects.

OpenSSL: The OpenSSL website [28] provides under the menu point "Docs" a link to frequently asked questions, the manual pages of all releases, a link to the *OpenSSL Cookbook*, and historic information on the OpenSSL FIPS Object Module. Additionally, there is a wiki available[9], the link to which is, however, hidden in the "Community" section of the OpenSSL website. The "Download" section of the OpenSSL website further mentions the OpenSSL Github Repository[10].

Several participants criticized that the documentation is *"spread across whole openssl page"* (P204) that one *"had to click through several links"* (P312), or that while there is *"a lot of documentation for OpenSSL, both manuals, API description and examples, it seemed [...] very fragmented"* (P322). Mostly likely referring to the manual pages, some participants criticized that the documentation contains *"only [a] list of functions"* (P102), that it is *"just an index of every function and it's your job to sort everything"* (P224), or *"provided functions to be too tainted by deprecated features"* (P226). Some other comments wrapped these findings up: *"it is really hard to find something if you don't precisely know what you are looking for"* (P107) or *"finding [the] proper function always took me some time"* (P303).

These issues can be interpreted as a result of a dynamically growing development effort with a variety of contributors, however, the nature of the project also fosters a variety of contributions that help developers in different ways, as mentioned positively by our participants: *"Simplest to get into i found OPENSSL, mostly because of the community i would say and also because the level of the tutorials"* (P220). Similarly, another participant pointed out: *"I find the documentation to be the best among all 3 libraries and there are lots of examples online"* (P207).

GnuTLS: The GnuTLS website [13] provides under the menu point "Documentation" links to the GnuTLS manual in several formats (HTML, PDF, EPUB), links to the GNU Guile bindings, and a link to frequently asked questions. The manual is a book-style document that does not only contain information on the API but also provides background on the TLS protocol and related matters (such as authentication and key management). It thus differs in style from the

[9] `https://wiki.openssl.org/index.php/Main_Page`
[10] `https://github.com/openssl/openssl`

OpenSSL manual pages which only provide a list of functions. The project's Gitlab page[11] is linked under the "Development (gitlab)" menu point on the GnuTLS website.

Several participants criticized the manual and documentation as being *"hideous and hard to orient"* (P107), *"most confusing, because the tutorial website was difficult to read"* (P220), or simply mentioning that the *"tutorial was too much verbose and long and it was very hard to find here what you want"* (P317). Yet, other participants relativized these points, by noting that the documentation is *"at first very overwhelming (there is a LOT) but once you get into it it's nicely done"* (P301) or mentioning that *"At first, I was scared because it was just like a markdown file (just a single page) [...]. But I think in the end it was good."* (P303)

In general, it seemed that participants were more positive towards the GnuTLS documentation, for instance, considering it as *"pretty straightforward"* (P218) and as *"amazingly structured"* (P306), or – as wrapped up by this participant: *"With GnuTLS you can have an overview of everything and dig deeper from that"* (P224).

When it comes to code snippets, opinions were diverse with some participants praising the available examples – *"a couple somewhat helpful examples at the end of its manual"* (P103), *"beautiful commented example which had everything needed for homework"* (P204), *"In the documentation, I could find a nice example of a TLS client with an x509 certificate"* (P303) – and others being unable to find relevant ones *"[the documentation] doesn't provide any useful examples"* (P101), *"it took me ages to find some relevant examples"* (P309).

mbed TLS: The (now being deprecated) mbed TLS website [22] provides under the menu point "Dev corner" > "API reference" the Doxygen-generated API documentation. Moreover, under "Dev corner" > "High-level design", there is an overview of the API modules and their dependencies. Right on the homepage of the now being deprecated mbed TLS website, there is a link to the new website[12] where developers can find the link to the project's Github repository[13].

Several participants criticized the documentation as being *"not rich in details"* (P109), as containing only *"very brief descriptions and hardly any examples"* (P226), or simply as being *"difficult to navigate and also very few code examples"* (P301).

Participants pointed out several features that they are missing, such as man pages – *"lacking manpages"* (P203), *"that's a shame that mbedTLS does not provide man pages on my distribution (Arch Linux)"* (P209) – or an orientation help – *"where is the SEARCH feature on your website"* (P216), *"No real index"* (P204).

At the same time, participants also noted the features that they liked: *"their official web page provided detailed TLS tutorial where I found almost everything I*

[11] https://gitlab.com/gnutls/gnutls/blob/master/README.md
[12] https://www.trustedfirmware.org/projects/mbed-tls/
[13] https://github.com/ARMmbed/mbedtls

needed" (P109), *"nice dependency graphs on its website"* (P203), *"there are some git repositories linked on the websites to present some code examples"* (P220), *"I also like the visual part of the documentation"* (P303).

3.4 Functionality Analysis

To evaluate the functional correctness of the submitted solutions, we tested them as described in Section 2.3. Figure 1 summarizes the results of our testing.

		compilation	valid: example	valid: google*	valid: badssl*	expired	wrong host	self signed	untrusted root	revoked: badssl*	revoked: grc	TLS 1.0*	TLS 1.1*	TLS 1.2
LIBRARY	OpenSSL	0.9	0.7	0.5	0.6	0.5	0.4	0.5	0.5	0.1	0.1	0.8	0.8	0.7
	GnuTLS	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.2	0.5	0.6	0.6	0.6
	mbed TLS	0.8	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.1	0.0	0.3	0.3	0.5
ROUND	YEAR 1	0.9	0.6	na	na	0.5	0.5	0.5	0.5	na	0.0	na	na	0.4
	YEAR 2	0.8	0.4	0.3	0.4	0.4	0.3	0.4	0.4	0.2	0.2	0.4	0.4	0.4
	YEAR 3	0.9	0.8	0.7	0.7	0.7	0.6	0.7	0.7	0.1	0.3	0.7	0.7	0.8

Fig. 1. Overview of the functionality analysis results. Numbers indicate the rounded fraction of all programs that succeeded in the given test category. Darkness indicates the severity of the issue. Valid domains were tested using *example.com*, *google.com*, and *badssl.com*. Certificate flaws and TLS version support were tested using *badssl.com*. Revoked certificates were tested using *badssl.com* and *revoked.grc.com*. Categories not tested in year 1 are marked with an asterisk (*).

A few clear patterns were observed. Out of the 180 total programs (3×60), most compiled. Several failed to connect even to the valid domains, indicating that getting the connection right with the given libraries can present a challenge. Handling flawed certificates turned out to be similarly hard in all three libraries with a medium amount of successful programs. Even worse, revocation checks were especially hard to implement — in all three libraries only a negligible minority succeeded.

For the enforcement of the correct TLS version, there seems to be a difference between the libraries with OpenSSL handling this task more easily than GnuTLS and mbed TLS.

While overall numbers are discouraging, participants in round three were generally more successful than in the other two rounds and this applied equally to all libraries (but not for revocation checks). Whether this difference is due to the higher experience of the participants or whether due to an increase in usability of all three libraries is to be investigated in future studies. As for now, the results of our qualitative analysis show that at least the perceived usability (as manifested in the user satisfaction) did not increase.

4 Discussion

The opinions on library usability appear to be diverse and sometimes conflicting: none of them was explicitly considered worst or best by a majority.

The most often mentioned usability factors include the quality of documentation and code samples, the abstraction level of the API, error handling and return values, and entity names. Properly checking for a revoked certificate turned out to be the most difficult part of the task and basic certificate checks were also not easy.

For all of the three libraries, there is space for improvement in different usability dimensions: OpenSSL should work on making developers aware of the BIO system and linking to suitable documentation. Moreover, return values showed to be confusing in this library (yet this is an issue that is hard to fix due to backward-compatibility reasons). Also, similar function names were sometimes perceived as confusing.

GnuTLS should support the developers in socket handling and explaining the differences between functions of similar naming. Apart from that, mixed comments on examples may hint towards a need to provide an easily accessible and unified resource for examples.

Similar to OpenSSL and GnuTLS, mbed TLS should address issues of hard-to-understand function names (e.g., from functions with similar names). When it comes to documentation, mbed TLS should work on navigability and additional resources (such as man pages).

While we discovered some options for improvement, API usability is most likely only one aspect that determines TLS client functionality and security. In real-world projects these factors may be further influenced by implementation constraints (such as compatibility with other systems, integration into legacy code, and library license models).

4.1 Study Limitations

As is the case with every study, various limitations may diminish the applicability of results. Our study was conducted in three rounds over the course of three years on real-world applications that might have changed during that time. While we could ensure in the task specification that students worked with similar versions of the API, we did not trace whether the (formal or informal) documentation changed. However, as we observed most of the themes appearing in all of the

three rounds, we determine the possibility that the documentation sources underwent significant changes as small. The comparison of the libraries may have been biased by prior experience with some of them (this was mentioned by some participants, but there was no common pattern). Furthermore, library order may have played a role (since the task was the same, participants' background knowledge increased with each subsequent library). Although graduate students are future IT professionals and junior developers constitute a non-negligible share of the developer population (almost 40% with 4 years or less of professional coding experience [30]), our sample still deviates from the actual developer population, and generalizations should thus be made with caution.

5 Related Work

Multiple studies have looked into the security of applications using TLS. Georgiev et al. [12] analyzed a representative sample of non-browser software applications and libraries that use TLS for secure Internet connections. Multiple libraries and apps were found to be broken. Authors argue the causes are poor API design, too many options presented to the developers, and bad documentation.

Fahl et al. performed an automated analysis of more than 14,000 Android and iOS applications [10,11] focusing on TLS certificate verification and man-in-the-middle attack possibilities. Many applications were found exploitable. Common issues included not verifying certificates, not checking the hostname, and allowing mixed content. They suggested a couple of technical countermeasures and argued for more developer education and simpler, usable tools to write secure applications. Similarly, Egele et al. [9] in their analysis of 11,000 Android applications found 88% to be vulnerable. The main assumed reasons included poor documentation and inappropriate defaults of the Java APIs.

To summarize, we see that security APIs are frequently misused, despite their crucial function. Although there is a plethora of usability analyses based on automatic code analysis as mentioned above, only a few studies had inspected the usability of security APIs from the point of user satisfaction.

For instance, Naiakshina et al. [24] conducted a qualitative study asking why developers store user passwords incorrectly. Their results reveal, among other things, that more usable APIs are not sufficient if secure defaults are not in place and that security is often secondary to functionality. Satisfaction was studied only marginally, in terms of expected and actual difficulty of the task.

Acar et al. [1] conducted a study comparing the usability of five Python security libraries. Apart from the functional correctness ("usable" libraries resulting more often in secure code), they tried to assess user satisfaction using the System Usability Scale (SUS) [6] and their own usability scale. Although SUS is widely known and used, it is not diagnostic so no specific conclusions could be drawn. Their own scale is diagnostic and consists of 11 questions combining the *Cognitive Dimensions framework* [32] with usability suggestions from Nielsen [27] and from Green and Smith [15]. Similar to our results, their results indicate that

documentation (and especially code examples in it) are of utmost importance and should be treated as a first-class requirement by library developers.

Nadi et al. [23] performed multiple separate usability studies in the context of Java's cryptographic APIs. One of the studies with developers identified several shortcomings replicated by later studies, as well as ours: lack of documentation and tutorials, unsatisfactory abstraction level, and lack of direct support for common tasks. User satisfaction was not considered explicitly. A different empirical study by Acar et al. [4] with GitHub users creating security-related code only included questions on self-reported success, task difficulty, solution security, previous experience, and demographics, not asking directly about user satisfaction with the API.

In summary, although TLS usually forms a crucial part of product security, misuse seems to be quite common. Previous research focused mainly on the effectiveness and efficiency components of usability, overwhelmingly in the context of cryptographic primitives. However, user satisfaction and specifics of TLS were usually understated—a gap which we intend to fill with our work.

6 Conclusion and Future Work

We conducted a three-fold exploratory study trying to identify usability issues of common TLS library APIs. 60 master-level students attempted to implement a simple TLS client using OpenSSL, GnuTLS, and mbed TLS.

We did not find evidence that any of the tested libraries was preferred by the participants, as they had multiple conflicting expectations. Common usability aspects mentioned by the participants included the quality of documentation (including the sample code snippets provided in it), the overall API abstraction level (especially complaining about network socket handling in GnuTLS and the BIO system in OpenSSL), return values consistency (especially in OpenSSL) and entity naming (where they found examples of similarly named functions in all three libraries).

Examining the effectiveness of the produced solutions, checking the revocation status of certificates turned out to be very difficult (with few successful participants) and basic certificate checks were also not overly easy.

The study suggests multiple directions for future work. Firstly, it may be beneficial for library developers to investigate the expectations of programmers using their interfaces. Secondly, user satisfaction with the APIs seems to be rather complex and a proper measuring methodology is still lacking. Key aspects of user satisfaction concerning APIs should be identified, synthesizing the existing API usability principles and user perceptions to answer questions such as: How do I write usable documentation? What error reporting is considered usable by the users? What is the right level of abstraction the users expect? Thirdly, further studies should be conducted to answer the question: "How can we teach developers to efficiently set up a TLS client in different libraries while taking into account diverse real-world constraints?".

Acknowledgments: This research was supported by the ERDF project *CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence* (No. CZ.02.1.01/0.0/0.0/16_019/0000822). We would like to thank Red Hat Czech for support and all students of the course for participating in this research. Thanks also go to Pavol Žáčik for helping to confirm different API functionality aspects.

References

1. Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M.L., Stransky, C.: Comparing the usability of cryptographic APIs. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 154–171 (2017). https://doi.org/10.1109/sp.2017.52
2. Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L., Stransky, C.: You get where you're looking for: The impact of information sources on code security. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 289–305 (2016). https://doi.org/10.1109/sp.2016.25
3. Acar, Y., Fahl, S., Mazurek, M.L.: You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In: 2016 IEEE Cybersecurity Development (SecDev). pp. 3–8. IEEE (2016)
4. Acar, Y., Stransky, C., Wermke, D., Mazurek, M.L., Fahl, S.: Security developer studies with GitHub users: Exploring a convenience sample. In: Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017). pp. 81–95. USENIX Association, Santa Clara, CA (2017)
5. Memorable site for testing clients against bad SSL configs (2022), https://badssl.com/
6. Brooke, J.: SUS: A quick and dirty usability scale. Usability Evaluation in Industry **189**(194), 4–7 (1996)
7. Cairns, K., Steel, G.: Developer-resistant cryptography. In: A W3C/IAB workshop on Strengthening the Internet Against Pervasive Monitoring (STRINT) (2014)
8. Dietrich, C., Krombholz, K., Borgolte, K., Fiebig, T.: Investigating system operators' perspective on security misconfigurations. In: 25th ACM Conference on Computer and Communications Security. ACM (October 2018)
9. Egele, M., Brumley, D., Fratantonio, Y., Kruegel, C.: An empirical study of cryptographic misuse in Android applications. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 73–84. CCS '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516693
10. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: Why Eve and Mallory love Android: An analysis of android SSL (in)security. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 50–61. ACM (2012). https://doi.org/10.1145/2382196.2382204
11. Fahl, S., Harbach, M., Perl, H., Koetter, M., Smith, M.: Rethinking SSL development in an appified world. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 49–60. CCS '13, ACM, New York, NY, USA (2013). https://doi.org/10.1145/2508859.2516655
12. Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., Shmatikov, V.: The most dangerous code in the world: Validating SSL certificates in non-browser software. In: Proceedings of the 2012 ACM conference on Computer and Communications Security. pp. 38–49. ACM (2012). https://doi.org/10.1145/2382196.2382204

13. GnuTLS: Transport layer security library (2022), https://www.gnutls.org/
14. Google transparency report: HTTPS encryption on the web (2021), https://transparencyreport.google.com/https
15. Green, M., Smith, M.: Developers are not the enemy!: The need for usable security APIs. IEEE Security & Privacy **14**, 40–46 (09 2016). https://doi.org/10.1109/msp.2016.111
16. Hazhirpasand, M., Ghafari, M., Krüger, S., Bodden, E., Nierstrasz, O.: The impact of developer experience in using java cryptography (2019)
17. Iacono, L.L., Gorski, P.L.: I do and I understand. Not yet true for security APIs. So sad. In: Proceedings of the 2nd European Workshop on Usable Security. EuroUSEC '17, Internet Security, Reston, VA (2017). https://doi.org/10.14722/eurousec.2017.23015
18. Krombholz, K., Busse, K., Pfeffer, K., Smith, M., von Zezschwitz, E.: "if https were secure, i wouldn't need 2fa" - end user and administrator mental models of https. In: S&P 2019 (May 2019), https://publications.cispa.saarland/2788/
19. Krombholz, K., Mayer, W., Schmiedecker, M., Weippl, E.: "I have no idea what I'm doing" – On the usability of deploying HTTPS. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 1339–1356 (2017)
20. Krüger, S., Späth, J., Ali, K., Bodden, E., Mezini, M.: CrySL: An extensible approach to validating the correct usage of cryptographic APIs. In: Millstein, T. (ed.) 32nd European Conference on Object-Oriented Programming (ECOOP 2018). Leibniz International Proceedings in Informatics (LIPIcs), vol. 109, pp. 10:1–10:27. Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.ECOOP.2018.10
21. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics **33**(1), 159–174 (1977). https://doi.org/10.2307/2529310
22. mbed TLS (formerly known as PolarSSL) (2022), https://tls.mbed.org
23. Nadi, S., Krüger, S., Mezini, M., Bodden, E.: Jumping through hoops: Why do java developers struggle with cryptography apis? In: Proceedings of the 38th International Conference on Software Engineering. pp. 935–946. ACM (2016)
24. Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S., Smith, M.: Why do developers get password storage wrong?: A qualitative usability study. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 311–328. CCS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134082
25. Naiakshina, A., Danilova, A., Tiefenau, C., Smith, M.: Deception task design in developer password studies: Exploring a student sample. In: Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018). pp. 297–313. USENIX Association, Baltimore, MD (Aug 2018), https://www.usenix.org/conference/soups2018/presentation/naiakshina
26. Nemec, M., Klinec, D., Svenda, P., Sekan, P., Matyas, V.: Measuring popularity of cryptographic libraries in Internet-wide scans. In: Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC). pp. 162–175. ACM Press, New York, NY, USA (2017). https://doi.org/10.1145/3134600.3134612
27. Nielsen, J.: Usability Engineering. Academic Press (1993)
28. OpenSSL: Cryptography and SSL/TLS toolkit (2022), https://www.openssl.org/
29. Saldaña, J.: The coding manual for qualitative researchers. SAGE Publishing, Thousand Oaks, CA, USA, 3rd edn. (2015)
30. Stackoverflow developer survey (2021), https://insights.stackoverflow.com/survey/2021

31. Tahaei, M., Vaniea, K.: A survey on developer-centred security. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 129–138. IEEE (2019)
32. Wijayarathna, C., Arachchilage, N.A.G., Slay, J.: A generic cognitive dimensions questionnaire to evaluate the usability of security APIs. In: Tryfonas, T. (ed.) Human Aspects of Information Security, Privacy and Trust. pp. 160–173. Springer International Publishing (2017)