2018-03-01

A Large-Scale Analysis of How OpenSSL Is Used in Open-Source Software

Scott Jared Heidbrink
Brigham Young University

A Large-Scale Analysis of How OpenSSL Is Used in Open Source Software

Scott Jared Heidbrink

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Daniel Zappala, Chair
Eric Mercer
Ryan Farrell

Department of Computer Science

Brigham Young University

ABSTRACT

A Large-Scale Analysis of How OpenSSL Is Used in Open Source Software

Scott Jared Heidbrink
Department of Computer Science, BYU
Master of Science

As vulnerabilities become more common the security of applications are coming under increased scrutiny. In regards to Internet security, recent work discovers that many vulnerabilities are caused by TLS library misuse. This misuse is attributed to large and confusing APIs and developer misunderstanding of security generally. Due to these problems there is a desire for simplified TLS libraries and security handling. However, as of yet there is no analysis of how the existing APIs are used, beyond how incorrect usage motivates the need to replace them. We provide an analysis of contemporary usage of OpenSSL across 410 popular secure applications. These insights will inform the security community as it addresses TLS library redesign.

Keywords: TLS, SSL, API, source code analysis, static code analysis

ACKNOWLEDGMENTS

Thanks go to my advisor Dr. Daniel Zappala for allowing me into his lab and helping throughout the troubles of thesis work. Thanks also to Mark O'Neill for convincing me I needed to get a degree. Thanks to my wife, Caffreina, for understanding when I had late nights, and to Link for coming when I had first hoped to finish.

Table of Contents

List of Tables

Chapter 1

Introduction

Transport Layer Security (TLS) is the primary security protocol of the Internet, and especially the Web. Sadly, TLS libraries have been shown to be difficult to use and understand [7, 13]. This has led to developers misusing these libraries and leaving their applications vulnerable to the attacks TLS is meant to prevent. This problem has been prevalent across differing TLS libraries and languages [10]. The problem is exacerbated by the complexity of the TLS protocol and extensions, which has led to developers even misunderstanding the guarantees of TLS and how the protocol functions internally [6, 13]. The problems of this misunderstanding and misuse are only becoming more prevalent as the call for "TLS everywhere" grows [1]. Security incidents are becoming more publicized and developers are forced to adopt more and more security responsibilities without proper training and understanding of security considerations.

These problems have led to recent work in how to educate developers and users about security practices, and TLS library redesigns to provide simpler means for developers to use the TLS protocol [4, 10]. While these library redesigns have focused on the problem of authenticating TLS connections, many other problems exist as well [7, 16], which all seem to stem from how current TLS usage violates the layer abstraction for which it was originally designed [28]. We believe there is a better approach, through moving the responsibility of security from application developers to operating systems and system administrators who have been shown to have more attention and concern to security matters [23]. The approach

we consider most appealing is porting TLS functionality to the operating system using the same POSIX socket API with which network application developers are already familiar.

While some efforts attempted to simplify library APIs [4], there seems to be no consideration of whether developers are even taking advantage of the flexibility these TLS APIs offer. As security properties are determined by much of the flexibility of TLS [14, 17, 19, 25], we believe taking a comprehensive look at how TLS APIs are actually used is an important consideration in future libraries and TLS use. We seek in this work to perform this analysis and provide insight into what security decisions should be made by application developers or system administrators. Unless security is the focus of the application, application developers are often less concerned with security matters. Because of this, many security considerations are best left to the user (or system administrators) who are more concerned and familiar with their security needs [23]. While application developers may be in a better position, for example, to know what hostnames they need to connect to, system administrators may wish to limit who they trust to verify the identity of the hostnames. Further, when security vulnerabilities are found, such as insecure protocol versions, often application developers are slow, if ever, to update their application [10], but system administrators would be much more responsive in disabling the insecure protocol. We seek to more fully explore this relationship and how to best place security decisions in the hands of the appropriate party.

Our contributions are as follows:

- A detailed analysis of the OpenSSL API
- Analysis of how developers are using the OpenSSL API
- Recommendations on how to simplify the OpenSSL API by replacing it with the POSIX API

Chapter 2

Related Work

We separate related work into three categories. The first category contains efforts to measure and assess developer mistakes in their use of TLS libraries. The second category comprises efforts to replace or redesign TLS libraries in an effort to simplify them for developer use. Finally, the third category includes efforts to perform API Mining, which covers work relating to analyzing API usage in general.

2.1 TLS library failures

Previous analysis of application usage of TLS has largely focused on how applications fail to perform proper certificate validation. Georgiev et al. [13] provided one of the first large scale analyses of TLS usage in non-browser applications by performing white and black box testing. They found that many applications are vulnerable to MitM attacks due to improper certificate validation. While the study was not comprehensive across all applications, the paper did find vulnerabilities in a wide range of applications from banking to instant messengers, which used a range of TLS libraries as well. Their conclusions were libraries are confusing to developers, don't provide safe defaults, and are not adequately tested. They also found in some instances developers were intentionally misleading and had disabled certificate validation while assuring end users their communications were secure.

The work by Georgiev et al. involved a large amount of manual effort and, because of the nature of white and black box testing, is difficult to scale. To overcome this limitation, He et al. introduced an automated tool, SSLint [16], to perform static code analysis of how

popular Ubuntu packages use contemporary TLS libraries. The primary function of the tool was to create program dependence graphs (PDGs) for an application and compare those to a labeled graph of known correct API usage. Modeling correct usage was preferable to modeling incorrect usage due to TLS libraries' APIs having relatively few correct uses, while there are numerous ways to use them incorrectly. Using standard graph querying/matching algorithms, they were able to find 27 new vulnerabilities against 381 applications. While the signatures used for correct API usage only modeled certificate validation, the methodology applies to any modeled usage.

Fahl et al. [9] conducted a similar study as Georgeiv but within the mobile application ecosystem. They performed static code analysis on 13,500 Android applications and found that even on Android, which provides a more standardized TLS handling library through Java, developers face similar problems to desktop systems. They later extended this work and looked at 1,009 iOS applications as well and then conducted interviews with developers to try and find the root cause of improper verification [10]. They found various causes of such mistakes:

- Forgetting to re-enable proper verification when switching from development to production.
- Misunderstanding of vulnerabilities caused by improper verification.
- Accepting all self-signed certificates when a self-signed certificate is used
- Slow, if ever, updates when informed about security problems.

Sounthiraraj et al. [26] improved on Fahl's work, similar to He over Georgiev, by creating a more automated tool, SVM-Hunter, that combined static and dynamic analysis techniques. The tool takes further advantage of the homogeneous nature of the Android OS, from the unified Java secure socket API, to explicit permissions and installation methods. They were able to find 1,453 possible vulnerable Android apps, 726 of which were confirmed vulnerable.

Rather than just looking at proper certificate validation, Bhargaven et al. [7] looked more at vulnerabilities caused by improper TLS handling and its interaction with higher protocols. They even introduced a novel triple-handshake attack and showed how the cookie forcing attack can be used when applications don't properly consider TLS handling. Their recommendations require a change in how each application handles TLS, but when Fahl's work is considered, showing that developers are often unmotivated in properly updating their applications, this seems an ineffectual solution.

We differ from this work in that we aren't looking for TLS library failings specifically, but only use them as motivations for simpler TLS handling. We are also interested in correct TLS library usage and how this can continue to be handled by developers through a simpler API, or how it should be moved to system administrator control.

2.2 TLS library replacements

Amour et al. created libtlssep [4], which provides a simple certificate validation API by creating a decorator for normal socket and TLS communication that focuses on proper name verification and safe default behaviors. They were able to limit the API to 11 function calls that closely mimic normal POSIX socket calls. They were able to show that current applications using standard TLS libraries can be ported over with negligible performance degradation. Fedora provides a global configuration file that allows developers to defer selection of cipher-suites, cipher-suite parameters, TLS version, and whether to perform secure renegotiations to the system-wide settings [21].

Another approach was developed by Certshim [5], which seeks to enforce correct certificate verification by wrapping dynamically-linked security libraries and implementing proper verification handling before returning control to the application. TrustBase [24], improves on this approach by providing a kernel module that inspects all network traffic to identify TLS connections and perform proper certificate validation independent of application

logic. Trustbase also allows for a direct verification API that application developers can call to perform certificate verification for them.

As certificate validation has been a main failure for developers, it is unsurprising this has been the focus of most work in this area. The push by Fedora to modify OpenSSL and GNUTLS to allow a system-wide configuration of certain security decisions is most closely related to our work. This still requires developers to actively adopt this model and still leaves developers with the burden of using TLS-specific libraries. To the best of our knowledge, no work in this area has taken into consideration current usage of the API to discuss what TLS functionality should be provided to the developers, nor provide a discussion of whether a system administrator should instead be in control of this functionality. Nor has any work examined how applications use TLS libraries in a holistic approach.

2.3 API Mining

Work assessing APIs is focused on understanding complex and poorly documented APIs to allow developers to learn from other developer's usage. It usually focuses on learning how other developers have used the API and extracting useful information. Generally, the field of mining software repositories [18] seeks to understand changes in software and libraries and what that means for new changes. Most work in this area is to provide developers with examples and understanding of API usage when documentation is poor [3, 12, 29]. There is some work in developing simpler APIs by analyzing API usage to find frequent sequences or call patterns that can be abstracted [27]. In analyzing a TLS API, however, there are specific security guarantees to consider.

In contrast with this body of work, we hypothesize that the OpenSSL API should be simplified by wrapping it within the POSIX API and seek to discover how that can be accomplished. While applying this previous work to simplify the API could prove useful, we believe the POSIX API is already in a good position to take this role as application

programmers already need to understand this API. Thus we seek to understand how the OpenSSL API could instead be provided within the simpler POSIX API.

Chapter 3

Background

3.1 TLS/SSL

Transport Layer Security (TLS) originally was called Secure Socket Layer (SSL). SSL advanced through three versions, and then was named to TLS after version 3. Because of this, the terms SSL and TLS are often used interchangeably. TLS provides secure communication by establishing cryptographic primitives between a server and client through a handshake protocol. Simply speaking, a client connects to a server expressing the TLS protocol versions and what cryptography algorithms it wishes to use, to provide confidentiality and integrity to the connection. This "Cipher-Suite" selection usually involves authenticating the server through the use of certificates and the Certificate Authority public-key infrastructure. The Server responds with its certificate, the selected protocol version, ciphers and optionally requests client authentication. The appropriate cryptography primitives are generated through data sent in this exchange and both sides inform the other that they will now be encrypting all future data. Because TLS was designed to be customizable and extensible, there are numerous TLS extensions that can be added to this handshake. In order to support the flexibility provided by TLS, TLS libraries often present large and complex APIs to developers to work with.

3.2 OpenSSL

OpenSSL is one such library that is arguably the most popular. It allows applications developers to create SSL structures, which can then be customized to select which TLS

versions, extensions, private keys, certificates, or cipher suites to use. Due to presumed application requirements, OpenSSL provides access to these SSL structures directly, or through various other interfaces. One such popular interface is the `SSL_CTX` or SSL context. This interface allows an application to configure one context from which to create multiple SSL structures, which can then be further customized. When a connection is established this is then referred to as an SSL session, because it contains the actual information needed to secure the particular connection. OpenSSL also provides access directly to individual connections through a session structure. In total there are three different API call layers that a developer can access, `SSL_CTX` which are used to configure TLS parameters for multiple `SSL` structures, which in turn configure multiple `SSL_SESSION`s, which configure an individual connection. There is a fourth special case set of API calls `SSL_CONF_CTX` whose purpose is to configure an `SSL_CTX` through command-line options.

OpenSSL can also be used as a general purpose cryptography library, so it provides two main shared object libraries to use, "libcrypto" and "libssl". Because we are only concerned with TLS usage, our analysis focuses on the "libssl" object file.

3.3 Secure Socket API

We stipulate that developers should be relieved of the burden of security, thus freeing them to focus entirely on the unique functionality that their applications provide. Likewise, the operating system and system administrators should be empowered to easily control and customize the secure connections on their machines. This shift is not without precedent. Most operating systems already offer critical services to applications to reduce code redundancy and to ensure that the services are run in a manner that does not threaten system stability or security. For example, application developers on Linux and Windows are not expected to write their own TCP implementation for networking applications or to implement their own file system functionality when writing to a file. Fedora led an effort to create a system-wide "CryptoPolicy" configuration file [21]. Through changes in OpenSSL and GNUTLS, this

configuration file allows developers to defer certain security decisions to administrators. In line with this precedent, and with the problems with developer security competence and administrator control in mind, we seek to establish all of TLS as an operating system controlled service.

In this paper we evaluate the practicality of this Secure Socket API (SSA), a minimalist TLS API that reduces contemporary security APIs to the POSIX socket API. Not only does this seem to be the original goal of secure network programming [28], this interface allows network application developers to use an interface with which they are already familiar, all while allowing flexible administrator control.

Chapter 4

Methodology

The following methodology was undertaken as a means to analyze the OpenSSL library to inform its port to the SSA. It is also presented here as generic guidelines for others seeking to reproduce or extend our work to other security libraries such as GnuTLS or JavaSE.

1. Extract the exported symbols of the API.
2. Manually analyze the documentation of the API for each symbol that was exported.
3. Categorize each symbol with related symbols for further analysis. This allows the analysis to focus on specific security considerations in the context of their affiliated functions. This also prevents looking further into inconsequential API calls (e.g. new, free, read, write), which are trivially wrapped by the POSIX API.
4. Gather a collection of source code packages that depend on the security library under analysis.
5. Create dependence and flow graphs for each package gathered.
6. For each category of symbols, analyze usage of the symbols with the category using graphs from the previous step:
   (a) Determine what use cases a developer has for the employ of the feature cumulatively expressed by the category.
   (b) Analyze the usage of functions in the category to gain insight into developer needs, developer mistakes, and augment the set of use cases derived from the previous step.

(c) Make recommendations for inclusion or exclusion in the SSA. Since the SSA relies upon simple key-value settings using `getsockopt` and `setsockopt`, any included SSA feature should contain a compelling motivation and eliminate redundant functionality exported by the original API.

We now cover each of these steps in detail, as they were undertaken in our analysis of OpenSSL. Due to computational and manual time constraints we limited our analysis to a single popular TLS library, OpenSSL, and its uses within Ubuntu's standard repository of packages [2].

4.1 Extracting Symbols

To extract the OpenSSL TLS API symbols we used the GNU tool `nm` to extract defined symbols in the `libssl1.0.0` shared library. We then recursively used the GNU tool `grep` to augment this list with any macros that are defined using one of these extracted symbols within OpenSSL's source code header files.

4.2 API Documentation Analysis

Symbol extraction yielded 507 symbols in the OpenSSL TLS API. We assessed each of these symbols manually with their corresponding official API documentation to determine their nature and purpose. Symbols which had no corresponding official documentation were categorized by manual analysis of their use within dependent packages, and using third-party documentation where available.

4.3 Categorization of Symbols

We utilized conventional qualitative analysis to categorize each of these functions. The derived categories correspond to the TLS protocol itself: Version selection, Extension management, Certificate/PrivateKey management, Cipher suite selection, Session management,

and Certificate/Key validation. Five additional categories were created to include symbols outside this context: Configuration, Allocation, Connection management, Instrumentation, and Miscellaneous. The Configuration category covers library-specific function calls. For OpenSSL this category contains functions for workarounds for various bugs identified when interacting with different TLS implementations, but also provides access to disabling security critical extensions that are implicitly used by the library. The Allocation category covers the memory management functions such as init and free methods. The Connection management category encompasses all symbols that translate easily to the POSIX socket API, such as `SSL_read`. The Instrumentation category covers function calls that provide debugging or informational output.

4.4 Source Code Collection

Using the package manager Aptitude we employed the `rdepends` tool to find all packages that directly depend on the `libssl1.0.0` shared object provided by OpenSSL, which corresponded to the OpenSSL version 1.0.2, which was Ubuntu's default at the time of analysis.

4.5 Graph Generation

To compute the necessary control and dependence graphs on the large collection of source code we used the Joern tool [30].

4.5.1 Joern

Joern uses a "fuzzy parser for C/C++ based on the concept of island grammar". This fuzzy parser sacrifices accuracy to allow larger amounts of code to be analyzed and alleviate the need to have code that compiles. This is in contrast to other tools which we attempted to use that are more feature-rich and accurate, but computationally intractable to run on our dataset. Since many other static analysis tools hook a compiler's symbolic representation of a program, they allow for a more accurate trace of control and data dependencies, especially

with respect to pointer analysis. Our experiences with these other tools are cataloged in Section 6.

From the parser output Joern generates code property graphs which combines information from abstract syntax trees, control flow graphs, and program dependence graphs. The abstract syntax tree allows us to match API symbols to actual lines in source code. The control flow graphs enable analysis of the order of API symbol usage and augment Joern's fuzzy parsing by insuring output data dependencies precede API calls. Data dependence graphs allow us to analyze API parameter values and return value usage. It also provides control dependencies, which allows us to quickly assess whether given variables are utilized in program branching.

4.6 Symbol Analysis

We use Python's `igraph` package to perform appropriate queries on the output of Joern. For each API call we extract various properties which are then used to find relevant behaviors based on our analysis of the API's documentation. First, we find every graph node corresponding to an API function call. We then extract all control flow predecessors and dependents. This allows us to check, for example, if `SSL_get_verify_result` is not called in conjunction with `SSL_get_peer_certificate`, which is a violation of the documentation requirements. We only use control flow rather than control dependence analysis here due to empirical results showing it to be more accurate with Joern's parsing. We then find every parameter used by these calls and trace all data dependencies. This includes extracting Joern's rudimentary statement understanding for these dependencies. This allows us for example, to find data dependencies that are themselves assignment expressions and recursively trace those data dependencies as well. We then ensure each of these dependencies is a control flow predecessor. Ideally we would ensure there existed a path between the data dependence and call that didn't pass through any other data dependency, however for nontrivial graphs this became an intractable problem. We then perform a similar extraction for return values, performing

similar recursive dependency checks. We also extract the Joern statement type that is assigned to the call, which allows us to determine how the function call is used in the source code context. This is part of our return value analysis, because Joern doesn't parse a non-variable assigned call as having a return value, which is frequent behavior in C and C++.

4.7 Synthesis and Recommendations

The data collected from the previous steps enabled us to quickly assess how developers use OpenSSL. In conjunction with the official documentation, we were able to synthesize a holistic understanding of the salient features of OpenSSL, and how its individual functions can be condensed into a set that resembled the POSIX socket API. Our exploration of each symbol category, as well as our recommendations for feature inclusion to the SSA, are outlined in Section 5.3.

4.8 Limitations

Joern limits our analysis by sacrificing accuracy for speed, which allows us to analyze a greater number of packages but at the risk of omitting some uses of API symbols and dependencies. Joern does not evaluate preprocessor directives, so it's possible that OpenSSL function calls are hidden through an application's use of macros or other directives, which can sometimes be quite complex. This also limits analysis as it is possible to change almost the entirety of the program through preprocessor directives. For example, the package `ftpd-ssl` surrounds some function calls with #if 0, which would prevent anything until the #endif macro from being included in compilation. Joern also only analyzes data and control within functions, which prohibits the analysis of interactions with global variables or attempting to trace the origin of local parameter values. As a trivial example, if a function `my_func` calls `SSL_new(param)`, but `param` is itself a parameter passed to `my_func`, Joern will be unable to determine `param`'s value. In practice we found it rare that OpenSSL dependencies extend beyond function boundaries, and in cases where they do we attempted to manually analyze the source file

in question. We also avoid call ordering analysis, as these often do occur beyond function boundaries, and limit conclusions based solely on existence of these function calls within a program. For example, as discussed in Section 5.3.1.2 the use of `SSLv23` allows for the use of SSL version 3 unless a call is made to `set_options` with the `SSLv3_OP_NO_SSLv3` or `SSL_OP_NO_TLSv1` option, rather than insuring that this option is set before a connection is actually made, we only analyze if this call is ever made. This still leaves the potential that a package uses another library which in turn calls the appropriate OpenSSL call. Our analysis still constitutes a significant manual effort, because more complicated analyses, such as symbolic execution or accounting for preprocessor directives, were not conducted.

Furthermore, some packages may simply provide a wrapper API around OpenSSL for other packages to use. For example, it is conceivable that a library provides proper certificate verification under its own API, and applications utilizing that API would not be counted in our analysis of OpenSSL-utilizing applications. Thus we may be underestimating the individual usage counts of OpenSSL symbols. This would require more complex tools or significant manual effort to detect, and we consider this analysis beyond the scope of our work.

In addition, the practical use of dependencies between packages in Aptitude allow package maintainers to infer the existence of OpenSSL by explicitly depending on another package that depends on OpenSSL. Software can then directly interact with OpenSSL yet not appear on direct dependency lists of OpenSSL. While the `rdepends` tool allows for this recursive dependency tracing, analyzing the resultant thousands of packages was beyond our means.

Chapter 5

Results

In this chapter we report on salient results from our exploration of the OpenSSL API. We report on the details of our data collection, documentation analysis, and category exploration with accompanying recommendations for incorporating OpenSSL into the Secure Socket API (SSA), which is based on the POSIX socket API.

5.1 Data Collection

Using Aptitude's `rdepends` tool we found 882 software packages that depend on `libssl`, OpenSSL's library for TLS functionality. Among these 882 software packages there are 687 unique source code repositories, after removal of various Linux header repositories that differed in version. From these 687 repositories, we removed OpenSSL itself and non-C/C++ packages and were left with 410 software packages. Using the GNU `nm` tool we extracted 323 unique symbols from version 1.0.0 of `libssl`, which was the latest version for Ubuntu at the time of writing. Using these 323 symbols we recursively gathered 181 macros directly from OpenSSL's source code. This left us with a total of 504 unique TLS API calls.

Using Joern to parse the 410 source code repositories, we found 165 of these API calls were unused by any software package. We found a total of 1,438 C/C++ source files that use any OpenSSL TLS API function, of which only 860 files were unique. A total of 24,124 API calls to the OpenSSL TLS API are made from the 1,438 files.

5.2 API Documentation Analysis

Because OpenSSL allows configuration of a TLS connection at different levels, discussed in Section 3.2, we found that the API was fairly redundant. In total there are 93 API calls that contain an equivalent call at another level. In general the `SSL_CTX`-related calls are used the most, over `SSL`- and `SSL_SESSION`-related calls. This suggests that TLS is usually setup application-wide rather than by developers configuring individual connections.

5.3 Category Analysis

Category	No. of Symbols	No. of Calls
TLS Functionality		
Version selection	29	1306
Extension management	68	597
Cipher suite selection	39	1467
Certificate/PrivateKey management	73	2083
Certificate/Key validation	51	3164
Session management	61	1155
Configuration	19	1337
Other		
Allocation	33	6087
Connection management	41	5228
Miscellaneous	64	1468
Instrumentation	26	232

Table 5.2: Breakdown of OpenSSL's `libssl` symbols by purpose. Many categories deal with management of the TLS protocol itself, but some others are concerned with allocation of objects and configuration of the library itself

Through conventional quantitative analysis we categorized each of the symbols into the categories shown in Table 5.2. Table 5.2 presents the overview of our category breakdown, including the number of symbols, or API calls, per category and total number of calls to a symbol in this category. Appendix A presents the full category to symbol breakdown and Appendix B presents the breakdown of each API's number of calls.

For each category in the "TLS Functionality" section of Table 5.2 we first present possible use cases for which a developer needs to have control of this functionality. We then analyze each function's usage within the category for how developers actually use it. This often takes the form of tracing data dependencies of parameters to analyze what values are normally passed to it and tracing return values for how any results are handled. Based on the analyzed usage we summarize our recommendation regarding whether this functionality should remain with application developers or be handled differently and explain our reasoning for this recommendation.

For each category in the "Other" section we simply present our recommendations for the handling of these functions because their usage is based on implementation details of the OpenSSL library and aren't specific to the TLS protocol.

5.3.1 Version selection

When creating a TLS connection, application developers must specify to OpenSSL the desired protocol version to utilize by selecting a corresponding `SSL_METHOD`. Normally this is selected by calling the appropriate `*_method` API function, prepended with a shortened version name (e.g., TLSv1) and whether it is a generic, client, or server socket. If a generic method call is used, whether the socket should behave as a server or client must be specified later through calls to `SSL_set_connect_state`, `SSL_set_accept_state`, or inferred from direct calls to `SSL_accept` or `SSL_connect`. These method selections also specify the underlying transport protocol to use (TLS for TCP, DTLS for UDP). Currently there are eight different `SSL_METHOD` calls, excluding differentiating client, server, and generic distinctions: SSLv23, TLSv1_2, TLSv1_1, TLSv1, SSLv3, SSLv2, DTLS, DTLSv1_2, DTLSv1.

SSLv23 and DTLS are "version-flexible", meaning they will implement the highest version of the protocol that can be agreed upon between the server and client. The supported versions, however, can also be further reduced through calls to `SSL_CTX_set_options` or

Version	method	client_method	server_method	Total
SSLv23_*	114	186	158	458
TLSv1_2_*	14	26	20	60
TLSv1_1_*	9	24	19	52
TLSv1_*	39	72	48	159
SSLv3_*	13	42	28	83
DTLS_*	1	0	0	1
DTLSv1_2_*	1	1	1	3
DTLSv1_*	12	10	9	31

Table 5.3: Version Method usage

`SSL_set_options`. While SSLv2 and SSLv3 have been shown to be insecure, only SSLv2 has been disabled by default when using these methods [22].

5.3.1.1 Use Cases

To maintain compatibility between a server and clients, it may be necessary to specify a particular protocol version so that communication can be realized when a new protocol is released. In addition, security has increasingly become a legal issue. Applications may want fine-grained controls over the security guarantees of their products. As such, developers may not wish to upgrade to new protocol versions that haven't seemingly undergone the same scrutiny of previous versions.

5.3.1.2 Usage Analysis

Table 5.3 shows the breakdown of each TLS version usage. As Table 5.3 shows, `SSLv23` methods are the most commonly used, which default to the latest protocol and allow backwards comparability with many older versions. While OpenSSL documentation recommends use of these methods and subsequent limitation of protocol selection through the `*_set_options` API, hundreds of packages still use the version specific method selections. `TLSv1` methods, which restrict the protocol version to only TLS 1.0, are the next most common. SSLv3 and SSLv2 have been proven insecure and are no longer recommended for use. However, 83

packages directly use these methods and 98 additional packages possibly accept these insecure connections through using `SSLv23` without setting the `SSL_OP_NO_SSLv3` option through the `set_options` API calls. This indicates some real-world developer neglect, resulting in serious security vulnerabilities.

Because other TLS version methods contain both the major and minor versions (e.g. `TLSv1_1` and `TLSv1_2`), it is unclear whether developers incorrectly assume that `TLSv1` actually supports any TLS minor versions as well. Anecdotally, we believe this to be the case, as comments in some packages project this misunderstanding. For example, the package `balsa` has a comment justifying the use of the `TLSv1` method over `SSLv23`: "we could also enable SSLv3 but it doe (sic) not work very well with all servers." This seems to communicate the misunderstanding that the only difference between `SSLv23` and `TLSv1` is that `SSLv23` enables SSL 3.0. In fact `SSLv23` allows for negotiation of TLS 1.1 and 1.2 while `TLSv1` only allows TLS 1.0. This has also been misunderstood by developers asking questions on help websites [20]. DTLSv1 is also used more than the version-flexible DTLS, perhaps for similar reasons.

Through manual inspection we found some packages specify their desired TLS version through compile-time checks, which merely select the versions included with the local OpenSSL installation, or directly by the administrator through specification of compile-time parameters. These practices indicate that application developers are already attempting to defer their selection of TLS versions to the requirements of the system administrator.

To transfer any actual application data, OpenSSL requires that the programmer specify the connection role as a server or a client. Several functions provide this ability, some of which combine specifying the client or server nature of a connection in tandem with the TLS version (e.g., `SSLv23_client_method`). Optionally, developers can also specify this implicitly (e.g., `SSL_connect` indicates a client role) and explicitly as a standalone function (e.g., `SSL_set_connect_state`). Specifying these roles independently, in tandem with other settings, or implicitly all have the same effect and thus represent redundancy in OpenSSL.

5.3.1.3 Recommendations

We recommend that the SSA use the `SSLv23` method calls and by default remove all versions of TLS that are known to be insecure, through the use of option flags (e.g., `SSL_OP_NO_SSLv3`).

Our results indicate that 54% of calls to version method functions are those that default to the latest TLS version supported by the system, but also support backwards compatibility with lower versions supported by the system, for connections with outdated remote hosts. The OpenSSL documentation indicates that these methods should be used [11]. Of calls that specify a specific TLS version to use, only 16% utilize the latest version of TLS (1.2), which is 7% of all calls. A substantial portion of calls (19%) also directly specify the use of TLS 1.0 only through the use of `TLSv1_method` options. This particular version is old and slated for end of life in June 2018. Our analysis of source code comments surrounding the use of this version suggest that developers erroneously believe it selects the latest TLS version. The use of preprocessor macros to determine version selection also indicates that developers already wish this burden to be placed on system administrators. With all of these factors in mind, it becomes clear that overwhelmingly developers want the system to select the version for them, or are adopting lower versions that should not be used anyway. We therefore recommend that the SSA select the latest TLS versions by default, and that deviation from this be controlled by the system administrator through configuration. System Administrators should also be able to configure their SSA to disable other protocol versions, or even enable experimental versions, such as TLS 1.3, as desired. If necessary, developers can request specific protocol versions through the `setsockopt` call, which will return an `errno` if that version has been disabled. Moving this control to system administrators empowers them to secure their machines independent of application logic.

5.3.2 Extension management

TLS extensions are used for three main purposes:

- Adding additional security

Extension	Description
SNI	Server Name Indication allows a client to express to which hostname it is trying to connect, allowing servers to host more than one hostname per port
ALPN/NPN	Application Layer Protocol Negotiation allows application layer protocols to perform handshake features within TLS for improved speed, currently only HTTP 2.0 is supported. This updates the Next Protocol Negotiation (NPN) extension.
OCSP	Online Certificate Status Protocol provides information about the validity of the certificate
Session Tickets	Session Tickets allow for clients to cache the session information to be returned to the server for quicker connection times
SRTP	Secure Real-time Transport Protocol is used to secure the RTP protocol
SRP	Secure Remote Password is a way to provide mutual authentication through a traditional password mechanism over TLS
Heartbeats	Heartbeats are used to keep long-running connections open
PRF	Psuedo Random Function extension was created to support U.S. Government requirements in certain applications to maintain properties within the TLS key material
Serverinfo	The `serverinfo` calls allow for arbitrary server extensions to be added
Supported Curves	The supported curves extension is used to select parameters for use in Elliptic Curve Cryptography

Table 5.5: Explicit TLS extensions implemented by OpenSSL.

- Fixing discovered vulnerabilities
- Performance improvements

OpenSSL implicitly uses some TLS extensions, such as secure renegotiation, and provides developer access to the utilization of ten different TLS extensions explicitly along with a mechanism to add custom (or unsupported) extensions. It also exports control over some implicitly used extensions through the `options` API. The aforementioned ten extensions are outlined in Table 5.5.

Extension	Client Usage	Server Usage	Total
SNI	68	44	77
NPN	-	-	31
ALPN	18	28	29
OCSP	19	8	19
Session Ticket	1	15	15
SRTP	9	9	9
SRP	3	2	3
custom	0	1	1
Heartbeats	0	0	0
PRF	0	0	0
Serverinfo	0	0	0
Supported Curves	0	0	0

Table 5.6: Extension usage

5.3.2.1 Use Cases

Many Extensions, such as SNI, require application information and as such require application participation for their use. Removal of the secure renegotiation extension allows for backwards compatibility for peers who do not support this extension.

5.3.2.2 Usage Analysis

As shown in Table 5.6, the most utilized extension is Server Name Indication, present in 19% of applications. NPN and its successor ALPN follow, with their combined use found in 15% of applications. OCSP here and elsewhere is utilized by a handful of applications (19). The remaining extensions are used by very few applications (less than 10) and Heartbeats, PRF, Serverinfo, and Supported Curves are used by none of the applications in our set.

`SSL_CTX_set_alpn_protos` and `SSL_set_alpn_protos` return 0 on success, which reverses traditional return code conventions of OpenSSL. Because of this, at least five packages incorrectly interpret their return codes. Thirty-one packages still use NPN even when it has been updated by ALPN. There is only one instance of custom extension use, which is by the `haproxy` package to provide a certificate time stamp. Two extensions have an ability to provide a callback function for their handling (SRP and SNI).

5.3.2.3 Recommendations

Based on their popularity, we recommend that SNI, ALPN, OCSP, Session Ticket, STRP, and SRP be included in the SSA, with the others left out for now, due to lack of interest. Most of these extensions are get and set calls that pass in data, which trivially translates to `setsockopt` and `getsockopt` calls. Two extensions SNI and SRP allow for a callback interface, which we recommend be eliminated as part of the API.

SNI allows a server to correctly respond to client TLS connection requests with the appropriate certificates and configuration if hosting multiple entities. It also allows a client to perform proper certificate verification. The callback interface, which we recommend be eliminated, is only used for server configuration. This is because developers can specify through `setsockopt` the list of certificates and hostnames that are supported. The SSA can then use this list to subsequently negotiate the handshake appropriately. Any additional configuration needed by the developer can then be performed after the handshake by developers calling `getsockopt` to retrieve the negotiated hostname. For clients, recent work has already discussed implementation details of how to handle this in an application-independent manner [5, 24]. The hostname verification can then take place in isolation from the application as discussed in Section 5.3.5.3.

Since OCSP support is a certificate validation extension, we recommend that it be under the control of the administrator configuration and enabled by default. OCSP could more easily receive adoption if placed under control of the system administrator. The argument against inclusion of this extension is that it increases connection latency, due to the extra time required to query an OCSP responder. However, security-performance tradeoffs like these are best decided by administrators because they are most familiar with the requirements of the environment in which their machines operate. Each of the eight server packages read OCSP information from a configuration file. Under administrator control, this information could be centralized to allow a server to associate OCSP data for any certificate it manages, and any server application that wishes to use that certificate will automatically provide

OCSP information to its clients. For client applications, this extension could be enabled depending on the system administrators needs. OCSP pinning, or whether this extension must be present, could then be set by the application developer through `setsockopt`.

Session Tickets are rather simple for servers to implement as they only need to supply an encryption key to use. This can easily be set as either a system-wide option or application option with `setsockopt`. Providing it as a system wide key allows administrators greater control over the security of the connections they trust. Clients, however, have been slow to adopt this extension, most likely because of the overhead of writing an appropriate caching mechanism. Allowing the OS to handle this caching then alleviates application responsibility and allows system administrators to specify how long they trust these crypto primitives to be used.

SRTP OpenSSL does not implement SRTP itself or its corresponding RFC, and instead merely provides data payloads for the extension. As a result, applications using this functionality have to implement the logic themselves. However, the SSA could enable this same level of support very easily through `setsockopt` and `getsetsockopt`.

SRP can either be configured system-wide, where the SSA can thus perform validation independently of the developer, or provided by `setsockopt` for the username/password pairs to then be validated. Because this is only used by the Apache packages, it is unclear which is the best way forward. This would also eliminate the need for the callback interface.

The remaining extensions are used by no applications. Given our goal to simplify the developer API in conjunction with this lack of use, we recommend that the other extensions not be present in the SSA.

5.3.3 Session management

Establishing the cryptographic primitives through the TLS handshake requires multiple round trips, which can be relatively expensive for latency-sensitive applications.. Because of this, a TLS session can be saved by servers and resumed during a future reconnect by clients.

Extension	Total usage
SSL_SESS_CACHE_OFF	75
SSL_SESS_CACHE_SERVER	44
SSL_SESS_CACHE_BOTH	26
SSL_SESS_CACHE_NO_AUTO_CLEAR	23
SSL_SESS_CACHE_NO_INTERNAL	14
SSL_SESS_CACHE_NO_INTERNAL_LOOKUP	10
SSL_SESS_CACHE_CLIENT	9
SSL_SESS_CACHE_NO_INTERNAL_STORE	8

Table 5.7: Most used cache options

Storing the session means that the key material generated during the initial handshake does not have to be regenerated. While OpenSSL provides a default internal mechanism for this, it also exposes an API to allow developers to perform their own caching, or manipulate/monitor the internal caching. This allows applications to optimize caching mechanisms specific to their application.

5.3.3.1 Use Cases

The session handling API calls are almost exclusively used for customizing caching behavior. It is also possible to attach application data onto the SSL_SESSION to be saved and retrieved in the cache.

5.3.3.2 Usage Analysis

The default caching functionality is just `SSL_SESS_CACHE_SERVER`, which the majority of applications keep. There are 163 calls to change the cache mode from 110 applications. The individual options selected are shown in Table 5.7. As the options are supplied to the API as a bitmask, 19 of those 163 calls "change" it to the default by only setting `SSL_SESS_CACHE_SERVER`. 75 disable the cache completely and manual analysis of source code revealed, based on developer comments, that at least some packages do this in conjunction with disabling renegotiations. 31 packages provide callbacks for external caching mechanisms.

5.3.3.3 Recommendations

We recommend providing the same caching option selection through the `setsockopt` call and eliminating custom caching support.

Implementing the caching mode flags is trivial with the `setsockopt` call and actually can improve the `SSL_SESS_CACHE_CLIENT` by allowing the SSA to have the necessary details of the connection to select the appropriate cached session.

73% of analyzed applications do not make any changes to the default caching mechanisms of OpenSSL. Within the other 27%, the most common modification is to simply turn caching off entirely. The majority of the remaining uses are to turn off individual features or retain default settings. There are 31 packages that utilize an OpenSSL API that allows custom session cache handling via callback registration. Given the small population of applications using custom session management, and the difficulties involved in providing callbacks via the POSIX socket API, we recommend that the SSA avoid custom session cache management by applications. Because this may not be an acceptable long-term solution, we give a discussion of the complexities of callbacks, cache management, and potential pathways forward in Chapter 7

5.3.4 Certificate/PrivateKey management

TLS connections employ certificates and keys when performing the TLS handshake. Thus applications utilize various functions from OpenSSL to load, generate, use, and inspect certificates and keys.

5.3.4.1 Use Cases

Physical machines rarely represent a singular identity. As such, applications must select an appropriate identity to present to connected endpoints, through keys and certificates. Servers often require a certificate and private key to be specified by the user, or generate these in

some automated form. Loading these data from file or memory, generating them, verifying their potential uses, and other operations are performed using this group of API functionality.

5.3.4.2 Usage Analysis

Of the 73 API functions used for managing keys and certificates shown in Appendix A, 38 (54%) are unused. Another 17 (23%) are used by less than five software packages. The remaining functions are used heavily, with a combined call count of 2033 from hundreds of distinct packages. Most of these are used to either specify certificate or private key for the TLS connection. However, one is used to verify that a given private key corresponds to a particular certificate, and two are used to provide decryption passphrases to unlock private keys.

The OpenSSL documentation states that `SSL_CTX_use_certificate_chain_file` is the recommended way to load certificates, as it provides the union of functionality for related calls. However, 120 packages continue to use the `SSL_*_use_certificate` calls. Because these functions load certificates from files, and as we do not perform symbolic execution, we are unable to determine whether it is common practice to store the leaf certificate and certificate chain separately. We are also unable to analyze whether the `use_RSAPrivateKey` files are used to selectively use an RSA key within a file of multiple different kind of keys, or simply to load equivalently as if the `use_PrivateKey` calls were used.

5.3.4.3 Recommendations

Given that the majority of supplied functions go unused by any package, and that the overwhelming majority of uses are for specifying the pathnames or bytes of certificates and private keys, we recommend that corresponding `setsockopt` options be created for supplying private key and certificate data. The certificate assignment option should take both chains and leaf certificates as input, just as the OpenSSL-recommended `SSL_CTX_use_certificate_chain_file` function. The SSA should determine on behalf of the user whether the supplied values are

paths or encoded certificates, and support only PEM encoding since that is the de facto standard. Additionally, the SSA can check whether a supplied key is valid for supplied certificates on behalf of the developer, returning an error code that indicates this. These steps further simplify the API and relieve developers of the burden of implementing common functionality.

For certificate handling OpenSSL's documentation recommends simply using the `SSL_CTX_use_certificate_chain_file`, and so we recommend that the implementation of the SSA use this call, and the equivalent `SSL_CTX_use_PrivateKey_file`, and provide that functionality through `setsockopt` calls.

5.3.5 Certificate/Key validation

Under TLS, failure to properly validate a certificate presented by the other endpoint eliminates all authentication guarantees, subjecting the connection to potential MitM attacks. This topic has been the primarily focus of study for the evaluation of failures in TLS library use. Much of this work has found that applications often misuse validation functions or omit their use entirely. OpenSSL provides a default verification process that can be, often inadvertently, ignored, usually for the purpose of performing custom application-specific verification.

5.3.5.1 Use Cases

The typical client use case for validation of certificates is to validate the certificate sent by the server in the TLS handshake. It is common for developers to allow OpenSSL to perform certificate signing verification and other checks but perform hostname validation themselves, because the functions that perform this are not well known to developers. A typical server use case for validation of certificates is in the validation of client certificates and supplying a list of trusted certificate authorities to the client so that it may select an appropriate certificate to send. In addition, an application may wish to "hard-code" a certificate and ensure that the one received through the TLS handshake is *exactly* the same, as a means of strict validation.

5.3.5.2 Usage Analysis

Immediately we note that 11 packages turn off validation entirely using `SSL_VERIFY_NONE` and specifying no callback for custom validation. Five of these packages include server behavior, so such action may be benign as this disables TLS client authentication, which most servers perform at the application layer using a different mechanism (e.g., login forms). However, the remaining six perform no verification whatsoever, indicating the presence of a man-in-the-middle vulnerability. A total of 7 packages use `SSL_get_verify_result`, but neglect to ensure `SSL_get_peer_certificate` returns a valid certificate. Neglecting this call is documented as a bug in the OpenSSL documentation, as receiving no certificate results in a success return value.

5.3.5.3 Recommendations

Certificate verification is crucial to TLS security and has been shown to have many problems in practice under existing TLS library use. Even if developers have a reason for custom verification they often perform it improperly. Recent work has described how to handle verification in an application-independent manner and under the control of administrator preferences [5, 24]. We therefore recommend that applications leave validation to the SSA, which defers to administrator preferences and secure defaults. We make this suggestion with one caveat: if an application would like to validate a certificate based on a hard-coded one, then it can supply that certificate to the SSA via `setsockopt`. This allows the SSA to maintain absolute control of validation, while also having the option of granting custom validation to applications, if the administrator allows it, by selecting to honor the request to validate using the provided certificate or not.

5.3.6 Cipher suite selection

A cipher suite is a set of cryptographic algorithms to be used by the TLS connection. These algorithms serve as the basis of the security guarantees of confidentiality, authentication, and integrity for the TLS connection.

5.3.6.1 Use Cases

Infamously, and currently controversially, there are numerous legal considerations when deciding appropriate ciphersuites. Previously the U.S. government mandated that any encryption over a certain key length be export restricted, and currently world governments have called for weakened encryption for all apps to aide in monitoring criminal activity. Applications have also historically borne the burden of legal issues with respect to security breaches and thus developers and companies have an incentive to ensure sufficient algorithms and security properties are utilized. For TLS, these issues are largely dependent on cipher suite selection.

5.3.6.2 Usage Analysis

Our analysis tools limit our ability to trace three of the main ways in which the cipher list is set: environment variables, configuration files, and from other functions. Automated analysis is limited to intra-procedural analysis so when the cipher list is passed in as a function parameter we are unable to trace it backward to find its definition. 208 (51%) of our studied packages contain code that sets the ciphers used by OpenSSL directly using the `*_set_cipher_list` family of functions. From the 334 calls to the `*_set_cipher_list` API calls we were only able to extract the set of ciphers given in Table 5.8. Of note is the use of `eNULL`, `NULL`, `COMPLEMENTOFALL`, `RC4`, and `MD5`. `eNULL`, `NULL` and `COMPLEMENTOFALL` all enable the "NULL" cipher which offers no encryption. The RC4 and MD5 algorithms are known to be too vulnerable for use in modern cryptography. This is either unknown to the developers of these applications, which undermines the security of TLS, or is an option that

Cipher Parameter	Usage
ALL	11
ADH:@STRENGTH	10
NULL	10
ALL:!LOW:!EXP:!MD5:!MD2	8
ADH	7
aNULL	6
eNULL	4
ALL:!ADH:!LOW:!EXP:!MD5:@STRENGTH	4
ALL:COMPLEMENTOFALL	3
HIGH:!EXPORT:!aNULL@STRENGTH	3
ALL:!eNULL:!EXP:!SSLv2:+ADH@STRENGTH	3
HIGH	3
DEFAULT:!EXP:!LOW	2
AES256-GCM-SHA384:AES256-SHA	2
DEFAULT	3
AES128-SHA	2
DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2:@STRENGTH	2
HIGH:!EXPORT:!aNULL@STRENGTH	2
ALL:!ADH:!LOW:!EXlikP:!MD5:@STRENGTH	1
ALL:!aNULL:!eNULL:!LOW:!EXPORT40:!RC4	1
DEFAULT:!KRB5	1
RC4-MD5	1
EECDH+aRSA+AESGCM:EECDH+aECDSA+AESGCM:EDH+aRSA+AESGCM	1
HIGH:MEDIUM:!RC4:!SRP:!PSK:!RSA:!aNULL@STRENGTH	1
SRP	1
SSLv3:TLSv1:HIGH:!LOW:!MEDIUM:!EXP:!NULL:!aNULL@STRENGTH	1
ALL:!ADH:!LOW:!EXP:!MD5:!RC4:!SHA1:!ECDH:!ECDSA:@STRENGTH	1
ALL:!ADH:!LOW:!EXP:!DES-CBC3-SHA:@STRENGTH	1
eNULL:ALL:!ADH:RC4+RSA:+SSLv2	1

Table 5.8: Cipher Suite choices

the package allows the user to select, such as the package `coturn`, which provides the ability to enable these as a commandline argument to allow a "secure" connection with the eNULL cipher. We next manually analyzed a sample of packages for which we cannot extract exact cipher suite settings. We found that many retrieve their settings for these functions from environment variables and configuration files.

5.3.6.3 Recommendations

We recommend that cipher suite selection be left to system-wide configuration files such as those implemented through Fedora's crypto policy file [21].

Cipher Suite selection is critical to the security guarantees of the TLS connection. The use of statically assigned values for cipher suites indicated insecure practices by developers, suggesting a need to have this option controlled by the system. Packages that did not set these values statically either set them through admin-configurable settings or adopt the defaults from the OpenSSL installation, thus reinforcing our recommendation that this be a parameter set by the system and its administrator and not by applications. Another problem with setting these values statically by the application is that as various algorithms are found to be vulnerable to attack, static values may not be updated to reflect new secure practice. However, the SSA could allow applications to disallow the use of specific algorithms through the use of `setsockopt`. That is, the SSA could export a restrictive API for cipher suite selection, but should not export a permissive one.

5.3.7 Configuration

OpenSSL allows for various controls of the internal and external behavior of the library. These OpenSSL options allow applications to request modifications to TLS connection handling by the library. Many of these options are workarounds for known bugs when OpenSSL interacts with remote hosts using a different TLS library implementation. For example, the option flag `SSL_OP_MICROSOFT_BIG_SSLV3_BUFFER` is used to handle an Internet Explorer violation of the SSLv3 specification.

Setting OpenSSL modes allows for control of internal library functionality, such as when internal buffers are released or whether to automatically perform renegotiations.

5.3.7.1 Use Cases

OpenSSL's options mechanism allows for control of miscellaneous features. This includes workarounds for poor implementations, further limiting TLS protocol selection, or even removing security extensions that have been deemed necessary.

5.3.7.2 Usage Analysis

Of the calls in this category, 830 (62%) are to two similar functions: `SSL_set_options` and `SSL_CTX_set_options`. These functions allow applications to employ bug workarounds for interaction with other TLS implementations and also allow the disabling of certain protocol use. The top four utilized options shown in Table 5.9 are used to disable vulnerable TLS features and older versions (compression, SSLv2, and SSLv3), and to enable "all" bug workarounds.

An additional 337 (25%) of calls to this category utilize various flags for the `SSL_set_mode` and `SSL_CTX_set_mode` functions. As shown in Table 5.10, 41% set a flag that makes I/O operations on a socket block if the handshake has not yet completed, 56% set flags that modify the `SSL_write` function to behave more like `write`, and 14% use a flag that reduces the memory footprint of idle TLS connections.

Also present are 32 calls (2%) to functions that change whether or not OpenSSL will attempt to read as many bytes input as possible during read operations.

Through manual inspection we find that many options are set by conditional pre-processor macros that are triggered based on compilation arguments, suggesting that many developers are leaving these decisions to administrators already.

The remaining functions in this category are used to clear and retrieve the settings described. Despite some of these options and modes "no longer having any effect", as described in the OpenSSL documentation, many continue to be used. While this continued use is harmless, it suggests that despite API changes developers are slow to update their code.

5.3.7.3 Recommendations

Due to discovered attacks against TLS [8, 15] compression has been removed in future TLS specs and we recommend the `SSL_OP_NO_COMPRESSION` option always be set. Given that the uses of this category are primarily bug workarounds and restricting the use of outdated protocols, and that many of these are already set through compilation flags, we recommend leaving such configurations to the administrator. Modes and other configuration settings in this category tend to control subtleties of read and write operations. Under the SSA, I/O semantics are largely determined by the existing POSIX socket standard and are ignored for the purposes of this work.

5.3.8 Allocation

There are 33 functions that are called a total of 6,087 times that are responsible for the assignment, allocation, and deallocation of common TLS objects (e.g. `SSL_CTX`) and various library globals. Under the POSIX API the only object with which the programmer interacts is a socket descriptor, and thus this category of OpenSSL TLS functionality has no analog under the SSA.

5.3.9 Connection management

The connection management category contains functions that perform connection and I/O operations on sockets. All of these have direct counterparts within the POSIX socket API, or have combinations of symbols that emulate the behavior, such as `SSL_connect` (`connect`), and `SSL_Peek` (`recv` with `MSG_PEEK` flag). Another example is that of `SSL_get_error`, which when called returns a value similar to `errno`. These functions are therefore mapped to their POSIX counterparts, and as such are already trivially present in the SSA.

5.3.10 Instrumentation

A group of 26 functions are responsible for providing debug information and statistics for TLS connection properties. Since all TLS operations are handled by the SSA, such functions no longer need to be developer-facing.

5.3.11 Miscellaneous

The miscellaneous group contains functions that assign various data structures to others, extract information from internal data structures, and those that are pending deprecation, such as functions that interact with compression methods. It also contains some functions that are used for translating strings from config files and commandline arguments into TLS context settings. Since all TLS operations are handled by the SSA, such functions no longer need to be developer-facing.

Option Flag	Total usage
SSL_OP_NO_SSLv2	271
SSL_OP_ALL	226
SSL_OP_NO_SSLv3	203
SSL_OP_NO_COMPRESSION	121
SSL_OP_NO_TICKET	95
SSL_OP_CIPHER_SERVER_PREFERENCE	78
SSL_OP_NO_TLSv1_1	75
SSL_OP_NO_TLSv1_2	73
SSL_OP_NO_TLSv1	72
SSL_OP_SINGLE_DH_USE	72
SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS	46
SSL_OP_SINGLE_ECDH_USE	38
SSL_OP_NO_SESSION_RESUMPTION_ON_RENEGOTIATION	35
SSL_OP_TLS_BLOCK_PADDING_BUG	25
~SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS	32
SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION	9
SSL_OP_MICROSOFT_BIG_SSLV3_BUFFER	5
SSL_OP_MSIE_SSLV2_RSA_PADDING	5
SSL_OP_TLS_D5_BUG	5
SSL_OP_COOKIE_EXCHANGE	5
SSL_OP_MICROSOFT_SESS_ID_BUG	5
SSL_OP_SSLEAY_080_CLIENT_DH_BUG	5
SSL_OP_SSLREF2_REUSE_CERT_TYPE_BUG	5
SSL_OP_NETSCAPE_CHALLENGE_BUG	5
SSL_OP_NO_SSL_MASK	4
SSL_OP_NO_QUERY_MTU	3
SSL_OP_TLS_ROLLBACK_BUG	3
~SSL_OP_NO_SSLv3	2
~SSL_OP_NO_SSLv2	2
SSL_OP_EPHEMERAL_RSA	2
SSL_OP_SAFARI_ECDHE_ECDSA_BUG	1
SSL_MODE_RELEASE_BUFFERS	1
TLS_PROTOCOL_SSLv2	1
SSL_OP_NO_DTLSv1_2	1
~SSL_OP_ALLOW_UNSAFE_LEGACY_RENEGOTIATION	1
SSL_MODE_ENABLE_PARTIAL_WRITE	1
~SSL_OP_NETSCAPE_REUSE_CIPHER_CHANGE_BUG	1
SSL_OP_NO_DTLSv1	1
~SSL_OP_CIPHER_SERVER_PREFERENCE	1
~SSL_OP_NO_TLSv1_2	1
~SSL_OP_NO_TLSv1_1	1
~SSL_OP_TLS_BLOCK_PADDING_BUG	1
SSL_OP_NETSCAPE_DEMO_CIPHER_CHANGE_BUG	1

Table 5.9: Individual Options selected

Mode Flag	Total usage
SSL_MODE_AUTO_RETRY	139
SSL_MODE_ENABLE_PARTIAL_WRITE	119
SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER	79
SSL_MODE_RELEASE_BUFFERS	48
SSL_MODE_NO_AUTO_CHAIN	6
SSL_MODE_SEND_FALLBACK_SCSV	3
SSL_MODE_SMALL_BUFFERS	2
SSL_MODE_HANDSHAKE_CUTTHROUGH	1

Table 5.10: Individual Modes selected

Chapter 6

Failures

The main goals of our project were to be able to trace data and control dependencies from a set API across a large amount of source code. This generally falls under the term static analysis, of which there are a multitude of tools with which to work.

Many popular static analysis tools focus on finding specific code vulnerabilities or potential problematic coding practices. Another set provide simple software metrics. Both of these static analysis tools were ill-suited for our work as they don't provide generalized code understanding for our custom analysis. This left us with a small set of static analysis frameworks from which we could choose.

The most popular of these, Frama-C, clang, and Code Surfer, all required hooking into the compiler to process source code. This allows them to do powerful analysis such as pointer analysis, macro expansion, and even evaluate the preprocessor, all of which would have been useful in our analysis. Because we were limiting our analysis to open source Ubuntu packages, we could easily hook into the Debian build commands with which they were designed to be run. The first problem with these kinds of tools is scalability. These tools could not perform the level of analysis we required at the scale we required. For instance, when analyzing the 882 software packages under Code Surfer we were unable to complete the `lite`-configuration analysis while running it on a 12 core, hyperthreaded, machine with 256 GB of RAM, despite it being run for a week. This is often due to coarse grained controls over limiting speed vs. functionality. This is also due to the apparent goal of these tools being to provide understanding for a single project rather than many. Attempting to analyze many projects

is poorly-documented, has an unfriendly interface, and often adversely affects performance. Even Joern, when analyzing a large set of projects, creates unnecessary graph edges in its backend database, which resulted in slow-downs when querying the database. To overcome this, we used Joern to export its code property graphs then built custom tools to parse and analyze them using python's `igraph` library. Of the tools that required compilation of source code, we found no tool that provided understanding of code removed through preprocessor directives, nor analysis of preprocessing directives themselves. While this is understandable, in our case we found that this commonly removed many use cases of the OpenSSL API. Using Joern allowed us to identify all possible uses of the OpenSSL API, rather than those that were manifest after compilation.

Chapter 7

Future Work

The next logical step in our work would be to apply the recommendations in creating the SSA. One of the main reasons for suggesting wrapping the functionality into the POSIX API is that most operating systems already have a set of system calls that implement this API. This would allow for easy adoption of the SSA in most operating systems. For example, in Linux this could be accomplished by adding a kernel module to register additional protocols into the networking structure, which Linux already supports. The actual TLS work can then be offloaded to a userspace application that utilizes a traditional TLS library. Then based on the recommendations from Section 5 functionality can be provided to developers through the `setsockopt` and `getsockopt` system calls, and to administrators through global, or application specific, configuration files. The format of these configuration files are also left to future work.

Another area for future work is in cache management of TLS sessions. While the majority of analyzed packages used OpenSSL's internal caching mechanism, 31 packages provided callbacks that allow for custom cache management. Of the packages we manually analyzed, we found that this functionality is often used to transfer cache information to and from other processes, for load balancing purposes. For instance, when a client reconnects to a server and the client is balanced to a different process (or machine) the server can retrieve session information from the process that handled the client previously. While OpenSSL uses the term "callbacks" for developer-supplied cache management functions, these should not be confused with callbacks in asynchronous programming - they are merely custom

functions that are called during the normal synchronous flow of operation. Providing this custom behavior directly to developers through the POSIX API raises a difficult security and architectural dilemma, as allowing applications to supply arbitrary code to system services can be dangerous. We believe that the best way forward is to not allow developers to handle session management directly, but provide this load balancing behavior internally in the SSA that can be enabled by an administrator. For instance, multiple SSA instances on other machines could synchronize their cache information to speed up connections with load-balanced clients. A benefit beyond the existing OpenSSL API with this approach is that differing server applications can make use of existing session data, rather than a single application. This would allow administrators in network-constrained situations to increasingly eliminate TLS handshake overhead.

We also see our analysis work extending in two directions. First, other popular TLS libraries, such as GNUtls or JavaSE, could be analyzed using our methodology. While our work is sufficient to demonstrate the practicality of the SSA, expanding it to these other libraries would provide insight into other possible developer use cases. Second, being able to use more powerful analysis tools would greatly improve out results. As discussed in Section 6 Joern best fit our purposes, however improvements in other tools, or Joern, would aide in analysis. First, an understanding of preprocessor directives would have greatly reduced the manual effort in understanding the code. If these directives were included as part of the control and data dependencies we could have quantified the dependence of API calls based on compiler directives. Second, intra-procedural analysis would have also increased our accuracy and conceptually seems a straightforward path to add to Joern. Third, while there are tools that hook directly into compiler understanding of source code, they provide a coarse grained control over speed and accuracy. For example, in our case we were only looking to analyze a small subset of the code within the compiled application, and no tool allows for pre-limiting the analysis to a subset of the code. As the TLS using source files represented a very small percentage of the total number of source files within a package, being able to restrict analysis

to this subset could have substantially decreased run-times. Finally, while compiler level understanding of source code is certainly desirable, configuring and building the source code often involves removing a significant amount of source code and removing potential use cases.

Chapter 8

Conclusion

We performed analysis of over 400 Ubuntu packages that depend on the OpenSSL TLS/SSL API to determine how current usage supports a new TLS handling paradigm. Since recent work has shown developers often fail in proper TLS coding, we proposed wrapping TLS functionality within the already familiar POSIX API. Current usage not only would allow for this transition, but in many cases seems to be the desire of developers and administrators. Many critical decisions when creating a TLS connection, such as version control and cipher suite selection, are already programmed to be configurable through configuration files and preprocessing macros. This suggests that application developers already wish to place security decisions in the hands of the users and system administrators. Further, continued use and support of obsolete and insecure calls and options supports the notion that developers are slow to update applications to security threats, thus supporting our decision to allow administrator control. While there are cases where a developer may need the fine-grained control that a TLS library can provide, in general providing this as a system service will allow developers to easily adopt secure practices with an already familiar API and empower administrators to better control the security of their machines.

Appendix A

Category Breakdown

Version selection		
Function name	No. of Using Packages	No. of Total Calls
DTLS_client_method	0	0
DTLS_method	1	1
DTLS_server_method	0	0
DTLSv1_2_client_method	1	1
DTLSv1_2_method	1	1
DTLSv1_2_server_method	1	1
DTLSv1_client_method	10	11
DTLSv1_method	12	14
DTLSv1_server_method	9	9
SSL_CTX_set_ssl_version	2	9
SSL_get_ssl_method	3	3
SSL_get_version	47	94
SSL_set_ssl_method	8	12
SSL_version	13	25
SSLv23_client_method	186	284
SSLv23_method	114	236
SSLv23_server_method	158	201
SSLv3_client_method	42	50
SSLv3_method	13	13
SSLv3_server_method	28	29
TLSv1_1_client_method	24	26
TLSv1_1_method	9	9
TLSv1_1_server_method	19	20
TLSv1_2_client_method	26	28
TLSv1_2_method	14	15
TLSv1_2_server_method	20	21
TLSv1_client_method	72	86
TLSv1_method	39	54
TLSv1_server_method	48	53

Extension management		
Function name	No. of Using Packages	No. of Total Calls
SRP_Calc_A_param	0	0
SRP_generate_client_master_secret	0	0
SRP_generate_server_master_secret	0	0
SSL_CTX_add_client_custom_ext	0	0
SSL_CTX_add_server_custom_ext	1	1
SSL_CTX_get_tlsext_ticket_keys	1	1
SSL_CTX_set_alpn_protos	17	20
SSL_CTX_set_alpn_select_cb	28	31
SSL_CTX_set_next_proto_select_cb	13	25
SSL_CTX_set_next_protos_advertised_cb	23	28
SSL_CTX_set_srp_cb_arg	2	2

SSL_CTX_set_srp_client_pwd_callback	0	0
SSL_CTX_set_srp_password	1	1
SSL_CTX_set_srp_strength	0	0
SSL_CTX_set_srp_username	1	1
SSL_CTX_set_srp_username_callback	2	2
SSL_CTX_set_srp_verify_param_callback	0	0
SSL_CTX_set_tlsext_opaque_prf_input_callback	0	0
SSL_CTX_set_tlsext_opaque_prf_input_callback_arg	0	0
SSL_CTX_set_tlsext_servername_arg	16	18
SSL_CTX_set_tlsext_servername_callback	39	50
SSL_CTX_set_tlsext_status_arg	10	14
SSL_CTX_set_tlsext_status_cb	15	21
SSL_CTX_set_tlsext_ticket_key_cb	12	14
SSL_CTX_set_tlsext_ticket_keys	1	1
SSL_CTX_set_tlsext_use_srtp	9	15
SSL_CTX_use_serverinfo	0	0
SSL_CTX_use_serverinfo_file	0	0
SSL_extension_supported	0	0
SSL_get0_alpn_selected	26	45
SSL_get0_next_proto_negotiated	23	44
SSL_get_selected_srtp_profile	7	7
SSL_get_servername	42	69
SSL_get_servername_type	0	0
SSL_get_shared_curve	0	0
SSL_get_srp_N	0	0
SSL_get_srp_g	0	0
SSL_get_srp_userinfo	2	2
SSL_get_srp_username	2	6
SSL_get_srtp_profiles	0	0
SSL_get_tlsext_heartbeat_pending	0	0
SSL_get_tlsext_status_exts	3	3
SSL_get_tlsext_status_ids	0	0
SSL_get_tlsext_status_ocsp_resp	6	8
SSL_heartbeat	0	0
SSL_select_next_proto	21	21
SSL_set1_curves	0	0
SSL_set1_curves_list	0	0
SSL_set1_sigalgs	0	0
SSL_set1_sigalgs_list	0	0
SSL_set_alpn_protos	1	1
SSL_set_cert_cb	0	0
SSL_set_cert_flags	0	0
SSL_set_session_ticket_ext	3	3
SSL_set_session_ticket_ext_cb	3	6
SSL_set_srp_server_param	2	2
SSL_set_srp_server_param_pw	0	0
SSL_set_tlsext_heartbeat_no_requests	0	0

SSL_set_tlsext_host_name	68	101
SSL_set_tlsext_opaque_prf_input	0	0
SSL_set_tlsext_status_exts	0	0
SSL_set_tlsext_status_ids	0	0
SSL_set_tlsext_status_ocsp_resp	18	18
SSL_set_tlsext_status_type	10	16
SSL_set_tlsext_use_srtp	0	0
SSL_srp_server_param_with_username	0	0
dtls1_process_heartbeat	0	0
tls1_process_heartbeat	0	0

Session management		
Function name	No. of Using Packages	No. of Total Calls
BIO_ssl_copy_session_id	0	0
DTLSv1_get_timeout	5	5
DTLSv1_handle_timeout	5	7
PEM_read_SSL_SESSION	1	2
PEM_read_bio_SSL_SESSION	1	1
PEM_write_SSL_SESSION	0	0
PEM_write_bio_SSL_SESSION	1	1
SSL_CTX_add_session	2	2
SSL_CTX_flush_sessions	6	6
SSL_CTX_get_session_cache_mode	10	10
SSL_CTX_get_timeout	9	9
SSL_CTX_remove_session	19	52
SSL_CTX_sess_get_cache_size	4	4
SSL_CTX_sess_get_get_cb	0	0
SSL_CTX_sess_get_new_cb	0	0
SSL_CTX_sess_get_remove_cb	0	0
SSL_CTX_sess_set_cache_size	37	43
SSL_CTX_sess_set_get_cb	31	31
SSL_CTX_sess_set_new_cb	31	34
SSL_CTX_sess_set_remove_cb	29	31
SSL_CTX_sessions	0	0
SSL_CTX_set_generate_session_id	0	0
SSL_CTX_set_session_cache_mode	110	163
SSL_CTX_set_session_id_context	72	96
SSL_CTX_set_timeout	50	60
SSL_SESSION_get0_peer	0	0
SSL_SESSION_get_compress_id	4	4
SSL_SESSION_get_ex_data	4	10
SSL_SESSION_get_ex_new_index	4	5
SSL_SESSION_get_id	15	35
SSL_SESSION_get_time	11	13
SSL_SESSION_get_timeout	9	9
SSL_SESSION_new	2	3

SSL_SESSION_print	4	4
SSL_SESSION_print_fp	0	0
SSL_SESSION_set1_id_context	0	0
SSL_SESSION_set_ex_data	3	6
SSL_SESSION_set_time	2	2
SSL_SESSION_set_timeout	8	8
SSL_add_session	0	0
SSL_copy_session_id	13	21
SSL_flush_sessions	0	0
SSL_get0_session	10	40
SSL_get1_session	20	35
SSL_get_default_timeout	4	4
SSL_get_session	46	105
SSL_get_time	0	0
SSL_get_timeout	0	0
SSL_has_matching_session_id	1	1
SSL_remove_session	0	0
SSL_session_reused	39	71
SSL_set_generate_session_id	0	0
SSL_set_session	48	68
SSL_set_session_id_context	15	24
SSL_set_session_secret_cb	4	8
SSL_set_time	0	0
SSL_set_timeout	4	4
d2i_SSL_SESSION	22	36
d2i_SSL_SESSION_bio	2	2
i2d_SSL_SESSION	24	79
i2d_SSL_SESSION_bio	1	1

Certificate/Key validation		
Function name	No. of Using Packages	No. of Total Calls
SSL_CTX_add_client_CA	17	18
SSL_CTX_get0_certificate	4	6
SSL_CTX_get0_param	1	1
SSL_CTX_get_cert_store	86	166
SSL_CTX_get_client_CA_list	14	20
SSL_CTX_get_client_cert_cb	0	0
SSL_CTX_get_extra_chain_certs	10	15
SSL_CTX_get_max_cert_list	0	0
SSL_CTX_get_verify_callback	12	12
SSL_CTX_get_verify_depth	9	9
SSL_CTX_get_verify_mode	20	29
SSL_CTX_load_verify_locations	217	445
SSL_CTX_select_current_cert	0	0
SSL_CTX_set0_verify_cert_store	0	0
SSL_CTX_set1_verify_cert_store	0	0

SSL_CTX_set_cert_store	25	44
SSL_CTX_set_cert_verify_callback	34	46
SSL_CTX_set_client_CA_list	92	116
SSL_CTX_set_client_cert_cb	10	10
SSL_CTX_set_default_verify	0	0
SSL_CTX_set_default_verify_paths	84	124
SSL_CTX_set_psk_client_callback	3	3
SSL_CTX_set_psk_server_callback	5	5
SSL_CTX_set_trust	0	0
SSL_CTX_set_verify	241	445
SSL_CTX_set_verify_depth	73	115
SSL_CTX_use_psk_identity_hint	2	2
SSL_add_client_CA	3	3
SSL_get0_param	2	2
SSL_get_certificate	45	72
SSL_get_client_CA_list	6	6
SSL_get_ex_data_X509_STORE_CTX_idx	81	92
SSL_get_peer_cert_chain	64	79
SSL_get_peer_certificate	245	550
SSL_get_psk_identity	2	2
SSL_get_psk_identity_hint	0	0
SSL_get_verify_callback	0	0
SSL_get_verify_depth	6	6
SSL_get_verify_mode	21	24
SSL_get_verify_result	159	326
SSL_load_client_CA_file	86	106
SSL_set0_verify_cert_store	0	0
SSL_set1_param	0	0
SSL_set1_verify_cert_store	0	0
SSL_set_client_CA_list	2	2
SSL_set_psk_client_callback	3	5
SSL_set_psk_server_callback	0	0
SSL_set_verify	83	174
SSL_set_verify_depth	19	57
SSL_set_verify_result	9	27
SSL_use_psk_identity_hint	0	0

Certificate/PrivateKey management		
Function name	No. of Using Packages	No. of Total Calls
SSL_CTX_add0_chain_cert	0	0
SSL_CTX_add1_chain_cert	0	0
SSL_CTX_add_extra_chain_cert	51	57
SSL_CTX_build_cert_chain	2	2
SSL_CTX_check_private_key	172	267
SSL_CTX_clear_cert_flags	0	0
SSL_CTX_clear_chain_certs	0	0

SSL_CTX_clear_extra_chain_certs	5	5
SSL_CTX_get0_chain_certs	1	1
SSL_CTX_get0_privatekey	1	1
SSL_CTX_get_extra_chain_certs_only	0	0
SSL_CTX_set0_chain	0	0
SSL_CTX_set0_chain_cert_store	0	0
SSL_CTX_set1_chain	0	0
SSL_CTX_set1_chain_cert_store	0	0
SSL_CTX_set1_client_certificate_types	0	0
SSL_CTX_set1_client_sigalgs	0	0
SSL_CTX_set1_client_sigalgs_list	0	0
SSL_CTX_set1_param	0	0
SSL_CTX_set1_sigalgs	0	0
SSL_CTX_set1_sigalgs_list	0	0
SSL_CTX_set_cert_cb	5	5
SSL_CTX_set_cert_flags	0	0
SSL_CTX_set_client_cert_engine	1	1
SSL_CTX_set_current_cert	2	4
SSL_CTX_set_default_passwd_cb	132	235
SSL_CTX_set_default_passwd_cb_userdata	82	119
SSL_CTX_set_max_cert_list	0	0
SSL_CTX_use_PrivateKey	71	89
SSL_CTX_use_PrivateKey_ASN1	0	0
SSL_CTX_use_PrivateKey_file	236	475
SSL_CTX_use_RSAPrivateKey	12	19
SSL_CTX_use_RSAPrivateKey_ASN1	2	2
SSL_CTX_use_RSAPrivateKey_file	21	28
SSL_CTX_use_certificate	74	100
SSL_CTX_use_certificate_ASN1	0	0
SSL_CTX_use_certificate_chain_file	169	284
SSL_CTX_use_certificate_file	111	200
SSL_add0_chain_cert	0	0
SSL_add1_chain_cert	0	0
SSL_add_dir_cert_subjects_to_stack	2	2
SSL_add_file_cert_subjects_to_stack	0	0
SSL_build_cert_chain	0	0
SSL_certs_clear	0	0
SSL_check_chain	0	0
SSL_check_private_key	16	22
SSL_clear_cert_flags	0	0
SSL_clear_chain_certs	0	0
SSL_get0_certificate_types	0	0
SSL_get0_chain_certs	0	0
SSL_get_max_cert_list	0	0
SSL_get_privatekey	20	22
SSL_get_shared_sigalgs	0	0
SSL_get_sigalgs	0	0

SSL_select_current_cert	0	0
SSL_set0_chain	0	0
SSL_set0_chain_cert_store	0	0
SSL_set1_chain	1	1
SSL_set1_chain_cert_store	0	0
SSL_set1_client_certificate_types	0	0
SSL_set1_client_sigalgs	0	0
SSL_set1_client_sigalgs_list	0	0
SSL_set_current_cert	2	2
SSL_set_max_cert_list	1	3
SSL_use_PrivateKey	25	36
SSL_use_PrivateKey_ASN1	3	6
SSL_use_PrivateKey_file	11	17
SSL_use_RSAPrivateKey	4	4
SSL_use_RSAPrivateKey_ASN1	3	3
SSL_use_RSAPrivateKey_file	5	5
SSL_use_certificate	21	37
SSL_use_certificate_ASN1	3	3
SSL_use_certificate_file	17	26

Cipher suite selection		
Function name	No. of Using Packages	No. of Total Calls
SSL_CIPHER_description	17	21
SSL_CIPHER_find	1	1
SSL_CIPHER_get_bits	52	98
SSL_CIPHER_get_id	4	5
SSL_CIPHER_get_name	71	117
SSL_CIPHER_get_version	39	49
SSL_CTX_need_tmp_RSA	10	12
SSL_CTX_set1_curves	0	0
SSL_CTX_set1_curves_list	0	0
SSL_CTX_set_cipher_list	197	301
SSL_CTX_set_ecdh_auto	25	31
SSL_CTX_set_tmp_dh	95	125
SSL_CTX_set_tmp_dh_callback	35	39
SSL_CTX_set_tmp_ecdh	69	78
SSL_CTX_set_tmp_ecdh_callback	3	3
SSL_CTX_set_tmp_rsa	14	24
SSL_CTX_set_tmp_rsa_callback	37	76
SSL_get0_ec_point_formats	0	0
SSL_get0_raw_cipherlist	0	0
SSL_get1_curves	0	0
SSL_get_cipher	44	80
SSL_get_cipher_bits	29	48
SSL_get_cipher_list	11	16
SSL_get_cipher_name	51	72

SSL_get_cipher_version	18	24
SSL_get_ciphers	16	21
SSL_get_current_cipher	90	179
SSL_get_server_tmp_key	4	4
SSL_get_shared_ciphers	4	4
SSL_need_tmp_RSA	0	0
SSL_set_cipher_list	24	33
SSL_set_ecdh_auto	0	0
SSL_set_pref_cipher	0	0
SSL_set_tmp_dh	0	0
SSL_set_tmp_dh_callback	3	3
SSL_set_tmp_ecdh	0	0
SSL_set_tmp_ecdh_callback	0	0
SSL_set_tmp_rsa	0	0
SSL_set_tmp_rsa_callback	3	3

Configuration		
Function name	No. of Using Packages	No. of Total Calls
SSL_CTX_clear_mode	0	0
SSL_CTX_clear_options	23	51
SSL_CTX_get_default_read_ahead	0	0
SSL_CTX_get_mode	4	4
SSL_CTX_get_options	31	37
SSL_CTX_get_read_ahead	0	0
SSL_CTX_set_default_read_ahead	1	1
SSL_CTX_set_mode	120	222
SSL_CTX_set_options	242	758
SSL_CTX_set_read_ahead	17	26
SSL_clear_mode	1	1
SSL_clear_options	10	14
SSL_get_mode	10	19
SSL_get_options	11	12
SSL_get_read_ahead	0	0
SSL_renegotiate_abbreviated	0	0
SSL_set_mode	66	115
SSL_set_options	43	72
SSL_set_read_ahead	5	5

Miscellaneous		
Function name	No. of Using Packages	No. of Total Calls
OPENSSL_1.0.0	0	0
OPENSSL_1.0.1	0	0
OPENSSL_1.0.1d	0	0
OPENSSL_1.0.2	0	0
OPENSSL_1.0.2g	0	0

SSL_COMP_add_compression_method	3	4
SSL_COMP_free_compression_methods	0	0
SSL_COMP_get_compression_methods	34	39
SSL_COMP_get_name	17	26
SSL_COMP_set0_compression_methods	0	0
SSL_CONF_CTX_clear_flags	0	0
SSL_CONF_CTX_finish	4	4
SSL_CONF_CTX_free	4	10
SSL_CONF_CTX_new	4	4
SSL_CONF_CTX_set1_prefix	0	0
SSL_CONF_CTX_set_flags	4	10
SSL_CONF_CTX_set_ssl	0	0
SSL_CONF_CTX_set_ssl_ctx	4	4
SSL_CONF_cmd	4	8
SSL_CONF_cmd_argv	0	0
SSL_CONF_cmd_value_type	2	2
SSL_CTX_callback_ctrl	0	0
SSL_CTX_ctrl	4	4
SSL_CTX_get_app_data	20	63
SSL_CTX_get_ex_data	28	78
SSL_CTX_get_ex_new_index	22	44
SSL_CTX_get_quiet_shutdown	0	0
SSL_CTX_get_ssl_method	0	0
SSL_CTX_set_app_data	18	33
SSL_CTX_set_cookie_generate_cb	3	4
SSL_CTX_set_cookie_verify_cb	3	4
SSL_CTX_set_ex_data	27	62
SSL_CTX_set_purpose	8	17
SSL_CTX_set_quiet_shutdown	9	9
SSL_SESSION_get_app_data	0	0
SSL_SESSION_set_app_data	0	0
SSL_callback_ctrl	0	0
SSL_clear	39	91
SSL_ctrl	0	0
SSL_export_keying_material	12	12
SSL_get_SSL_CTX	45	69
SSL_get_app_data	49	166
SSL_get_current_compression	20	37
SSL_get_current_expansion	2	2
SSL_get_ex_data	52	115
SSL_get_ex_new_index	43	65
SSL_get_finished	6	7
SSL_get_peer_finished	6	7
SSL_get_peer_signature_nid	0	0
SSL_get_quiet_shutdown	0	0
SSL_get_secure_renegotiation_support	3	7
SSL_get_shutdown	51	95

SSL_num_renegotiations	4	6
SSL_renegotiate_pending	3	6
SSL_set_SSL_CTX	39	52
SSL_set_app_data	47	65
SSL_set_ex_data	60	106
SSL_set_purpose	0	0
SSL_set_quiet_shutdown	29	34
SSL_set_shutdown	48	89
SSL_set_trust	0	0
SSL_test_functions	0	0
sk_SSL_CIPHER_find	2	8
sk_SSL_CIPHER_find_ex	0	0

	Allocation	
Function name	No. of Using Packages	No. of Total Calls
BIO_f_ssl	8	10
BIO_new_buffer_ssl_connect	1	1
BIO_new_ssl	15	21
BIO_new_ssl_connect	10	13
ERR_load_SSL_strings	8	25
OpenSSL_add_ssl_algorithms	22	24
SSL_CTX_SRP_CTX_free	0	0
SSL_CTX_SRP_CTX_init	0	0
SSL_CTX_free	302	986
SSL_CTX_new	372	919
SSL_SESSION_free	29	93
SSL_SRP_CTX_free	0	0
SSL_SRP_CTX_init	0	0
SSL_dup	3	5
SSL_dup_CA_list	1	2
SSL_free	344	1032
SSL_get_fd	33	60
SSL_get_rbio	54	122
SSL_get_rfd	3	5
SSL_get_wbio	35	85
SSL_get_wfd	1	1
SSL_library_init	334	507
SSL_load_error_strings	343	531
SSL_new	355	769
SSL_set_bio	141	253
SSL_set_fd	223	471
SSL_set_rfd	23	36
SSL_set_wfd	22	35
SSLeay_add_ssl_algorithms	54	76
lh_SSL_SESSION_free	0	0
lh_SSL_SESSION_new	0	0
ssl3_setup_buffers	2	3

ssl_init_wbio_buffer	2	2

Connection management		
Function name	No. of Using Packages	No. of Total Calls
BIO_ssl_shutdown	10	10
DTLS_get_link_min_mtu	0	0
DTLS_set_link_mtu	0	0
DTLSv1_listen	0	0
SSL_CTX_set_max_send_fragment	2	2
SSL_accept	192	256
SSL_alert_desc_string	0	0
SSL_alert_desc_string_long	41	66
SSL_alert_type_string	1	1
SSL_alert_type_string_long	31	54
SSL_connect	267	450
SSL_do_handshake	50	93
SSL_get_error	309	1247
SSL_get_state	20	55
SSL_in_accept_init	6	6
SSL_in_before	4	4
SSL_in_connect_init	7	9
SSL_in_init	19	20
SSL_is_init_finished	38	99
SSL_is_server	1	1
SSL_peek	38	50
SSL_pending	105	168
SSL_read	332	600
SSL_renegotiate	12	19
SSL_rstate_string	0	0
SSL_rstate_string_long	0	0
SSL_set_accept_state	116	154
SSL_set_connect_state	124	182
SSL_set_max_send_fragment	1	1
SSL_set_mtu	2	4
SSL_set_state	3	6
SSL_shutdown	271	672
SSL_state	4	9
SSL_state_string	10	59
SSL_state_string_long	42	255
SSL_want	9	9
SSL_want_nothing	0	0
SSL_want_read	20	41
SSL_want_write	15	27
SSL_want_x509_lookup	0	0
SSL_write	328	599

Instrumentation		
Function name	No. of Using Packages	No. of Total Calls
SSL_CTX_get_info_callback	0	0
SSL_CTX_sess_accept	6	6
SSL_CTX_sess_accept_good	7	7
SSL_CTX_sess_accept_renegotiate	6	6
SSL_CTX_sess_cache_full	6	6
SSL_CTX_sess_cb_hits	6	6
SSL_CTX_sess_connect	5	5
SSL_CTX_sess_connect_good	5	5
SSL_CTX_sess_connect_renegotiate	5	5
SSL_CTX_sess_hits	7	7
SSL_CTX_sess_misses	7	7
SSL_CTX_sess_number	8	8
SSL_CTX_sess_timeouts	7	7
SSL_CTX_set_info_callback	54	89
SSL_CTX_set_msg_callback	4	4
SSL_CTX_set_msg_callback_arg	1	1
SSL_cache_hit	2	2
SSL_clear_num_renegotiations	0	0
SSL_get_info_callback	0	0
SSL_set_debug	0	0
SSL_set_info_callback	17	25
SSL_set_msg_callback	14	15
SSL_set_msg_callback_arg	14	15
SSL_set_tlsext_debug_arg	0	0
SSL_set_tlsext_debug_callback	0	0
SSL_total_renegotiations	5	6

Appendix B

Total Usage

API function	No. of using packages	No. of total calls
SSL_CTX_new	372	919
SSL_new	355	769
SSL_free	344	1032
SSL_load_error_strings	343	531
SSL_library_init	334	507
SSL_read	332	600
SSL_write	328	599
SSL_get_error	309	1247
SSL_CTX_free	302	986
SSL_shutdown	271	672
SSL_connect	267	450
SSL_get_peer_certificate	245	550
SSL_CTX_set_options	242	758
SSL_CTX_set_verify	241	445
SSL_CTX_use_PrivateKey_file	236	475
SSL_set_fd	223	471
SSL_CTX_load_verify_locations	217	445
OPENSSL_free	214	1356
SSL_CTX_set_cipher_list	197	301
SSL_accept	192	256
SSLv23_client_method	186	284
SSL_CTX_check_private_key	172	267
SSL_CTX_use_certificate_chain_file	169	284
SSL_get_verify_result	159	326
SSLv23_server_method	158	201
SSL_set_bio	141	253
SSL_CTX_set_default_passwd_cb	132	235
SSL_set_connect_state	124	182
SSL_CTX_set_mode	120	222
SSL_set_accept_state	116	154
SSLv23_method	114	236
SSL_CTX_use_certificate_file	111	200
SSL_CTX_set_session_cache_mode	110	163
SSL_pending	105	168
SSL_CTX_set_tmp_dh	95	125
SSL_CTX_set_client_CA_list	92	116
SSL_get_current_cipher	90	179
SSL_CTX_get_cert_store	86	166
SSL_load_client_CA_file	86	106
SSL_CTX_set_default_verify_paths	84	124
SSL_set_verify	83	174
SSL_CTX_set_default_passwd_cb_userdata	82	119
SSL_get_ex_data_X509_STORE_CTX_idx	81	92
SSL_CTX_use_certificate	74	100
SSL_CTX_set_verify_depth	73	115

SSL_CTX_set_session_id_context	72	96
TLSv1_client_method	72	86
SSL_CIPHER_get_name	71	117
SSL_CTX_use_PrivateKey	71	89
SSL_CTX_set_tmp_ecdh	69	78
SSL_set_tlsext_host_name	68	101
SSL_set_mode	66	115
SSL_get_peer_cert_chain	64	79
SSL_set_ex_data	60	106
SSL_CTX_set_info_callback	54	89
SSL_get_rbio	54	122
SSLeay_add_ssl_algorithms	54	76
SSL_CIPHER_get_bits	52	98
SSL_get_ex_data	52	115
SSL_CTX_add_extra_chain_cert	51	57
SSL_get_cipher_name	51	72
SSL_get_shutdown	51	95
SSL_CTX_set_timeout	50	60
SSL_do_handshake	50	93
SSL_get_app_data	49	166
SSL_set_session	48	68
SSL_set_shutdown	48	89
TLSv1_server_method	48	53
SSL_get_version	47	94
SSL_set_app_data	47	65
SSL_get_session	46	105
SSL_get_SSL_CTX	45	69
SSL_get_certificate	45	72
SSL_get_cipher	44	80
SSL_get_ex_new_index	43	65
SSL_set_options	43	72
SSL_get_servername	42	69
SSL_state_string_long	42	255
SSLv3_client_method	42	50
SSL_alert_desc_string_long	41	66
SSL_CIPHER_get_version	39	49
SSL_CTX_set_tlsext_servername_callback	39	50
SSL_clear	39	91
SSL_session_reused	39	71
SSL_set_SSL_CTX	39	52
TLSv1_method	39	54
SSL_is_init_finished	38	99
SSL_peek	38	50
SSL_CTX_sess_set_cache_size	37	43
SSL_CTX_set_tmp_rsa_callback	37	76
SSL_CTX_set_tmp_dh_callback	35	39
SSL_get_wbio	35	85

SSL_COMP_get_compression_methods	34	39
SSL_CTX_set_cert_verify_callback	34	46
SSL_get_fd	33	60
SSL_CTX_get_options	31	37
SSL_CTX_sess_set_get_cb	31	31
SSL_CTX_sess_set_new_cb	31	34
SSL_alert_type_string_long	31	54
SSL_CTX_sess_set_remove_cb	29	31
SSL_SESSION_free	29	93
SSL_get_cipher_bits	29	48
SSL_set_quiet_shutdown	29	34
SSL_CTX_get_ex_data	28	78
SSL_CTX_set_alpn_select_cb	28	31
SSLv3_server_method	28	29
SSL_CTX_set_ex_data	27	62
SSL_get0_alpn_selected	26	45
TLSv1_2_client_method	26	28
SSL_CTX_set_cert_store	25	44
SSL_CTX_set_ecdh_auto	25	31
SSL_use_PrivateKey	25	36
SSL_set_cipher_list	24	33
TLSv1_1_client_method	24	26
i2d_SSL_SESSION	24	79
SSL_CTX_clear_options	23	51
SSL_CTX_set_next_protos_advertised_cb	23	28
SSL_get0_next_proto_negotiated	23	44
SSL_set_rfd	23	36
OpenSSL_add_ssl_algorithms	22	24
SSL_CTX_get_ex_new_index	22	44
SSL_set_wfd	22	35
d2i_SSL_SESSION	22	36
SSL_CTX_use_RSAPrivateKey_file	21	28
SSL_get_verify_mode	21	24
SSL_select_next_proto	21	21
SSL_use_certificate	21	37
SSL_CTX_get_app_data	20	63
SSL_CTX_get_verify_mode	20	29
SSL_get1_session	20	35
SSL_get_current_compression	20	37
SSL_get_privatekey	20	22
SSL_get_state	20	55
SSL_want_read	20	41
TLSv1_2_server_method	20	21
SSL_CTX_remove_session	19	52
SSL_in_init	19	20
SSL_set_verify_depth	19	57
TLSv1_1_server_method	19	20

SSL_CTX_set_app_data	18	33
SSL_get_cipher_version	18	24
SSL_set_tlsext_status_ocsp_resp	18	18
SSL_CIPHER_description	17	21
SSL_COMP_get_name	17	26
SSL_CTX_add_client_CA	17	18
SSL_CTX_set_alpn_protos	17	20
SSL_CTX_set_read_ahead	17	26
SSL_set_info_callback	17	25
SSL_use_certificate_file	17	26
SSL_CTX_set_tlsext_servername_arg	16	18
SSL_check_private_key	16	22
SSL_get_ciphers	16	21
BIO_new_ssl	15	21
SSL_CTX_set_tlsext_status_cb	15	21
SSL_SESSION_get_id	15	35
SSL_set_session_id_context	15	24
SSL_want_write	15	27
SSL_CTX_get_client_CA_list	14	20
SSL_CTX_set_tmp_rsa	14	24
SSL_set_msg_callback	14	15
SSL_set_msg_callback_arg	14	15
TLSv1_2_method	14	15
SSL_CTX_set_next_proto_select_cb	13	25
SSL_copy_session_id	13	21
SSL_version	13	25
SSLv3_method	13	13
DTLSv1_method	12	14
SSL_CTX_get_verify_callback	12	12
SSL_CTX_set_tlsext_ticket_key_cb	12	14
SSL_CTX_use_RSAPrivateKey	12	19
SSL_export_keying_material	12	12
SSL_renegotiate	12	19
SSL_SESSION_get_time	11	13
SSL_get_cipher_list	11	16
SSL_get_options	11	12
SSL_use_PrivateKey_file	11	17
BIO_new_ssl_connect	10	13
BIO_ssl_shutdown	10	10
DTLSv1_client_method	10	11
SSL_CTX_get_extra_chain_certs	10	15
SSL_CTX_get_session_cache_mode	10	10
SSL_CTX_need_tmp_RSA	10	12
SSL_CTX_set_client_cert_cb	10	10
SSL_CTX_set_tlsext_status_arg	10	14
SSL_clear_options	10	14
SSL_get0_session	10	40

SSL_get_mode	10	19
SSL_set_tlsext_status_type	10	16
SSL_state_string	10	59
DTLSv1_server_method	9	9
SSL_CTX_get_timeout	9	9
SSL_CTX_get_verify_depth	9	9
SSL_CTX_set_quiet_shutdown	9	9
SSL_CTX_set_tlsext_use_srtp	9	15
SSL_SESSION_get_timeout	9	9
SSL_set_verify_result	9	27
SSL_want	9	9
TLSv1_1_method	9	9
BIO_f_ssl	8	10
ERR_load_SSL_strings	8	25
SSL_CTX_sess_number	8	8
SSL_CTX_set_purpose	8	17
SSL_SESSION_set_timeout	8	8
SSL_set_ssl_method	8	12
SSL_CTX_sess_accept_good	7	7
SSL_CTX_sess_hits	7	7
SSL_CTX_sess_misses	7	7
SSL_CTX_sess_timeouts	7	7
SSL_get_selected_srtp_profile	7	7
SSL_in_connect_init	7	9
SSL_CTX_flush_sessions	6	6
SSL_CTX_sess_accept	6	6
SSL_CTX_sess_accept_renegotiate	6	6
SSL_CTX_sess_cache_full	6	6
SSL_CTX_sess_cb_hits	6	6
SSL_get_client_CA_list	6	6
SSL_get_finished	6	7
SSL_get_peer_finished	6	7
SSL_get_tlsext_status_ocsp_resp	6	8
SSL_get_verify_depth	6	6
SSL_in_accept_init	6	6
DTLSv1_get_timeout	5	5
DTLSv1_handle_timeout	5	7
SSL_CTX_clear_extra_chain_certs	5	5
SSL_CTX_sess_connect	5	5
SSL_CTX_sess_connect_good	5	5
SSL_CTX_sess_connect_renegotiate	5	5
SSL_CTX_set_cert_cb	5	5
SSL_CTX_set_psk_server_callback	5	5
SSL_set_read_ahead	5	5
SSL_total_renegotiations	5	6
SSL_use_RSAPrivateKey_file	5	5
SSL_CIPHER_get_id	4	5

SSL_CONF_CTX_finish	4	4
SSL_CONF_CTX_free	4	10
SSL_CONF_CTX_new	4	4
SSL_CONF_CTX_set_flags	4	10
SSL_CONF_CTX_set_ssl_ctx	4	4
SSL_CONF_cmd	4	8
SSL_CTX_ctrl	4	4
SSL_CTX_get0_certificate	4	6
SSL_CTX_get_mode	4	4
SSL_CTX_sess_get_cache_size	4	4
SSL_CTX_set_msg_callback	4	4
SSL_SESSION_get_compress_id	4	4
SSL_SESSION_get_ex_data	4	10
SSL_SESSION_get_ex_new_index	4	5
SSL_SESSION_print	4	4
SSL_get_default_timeout	4	4
SSL_get_server_tmp_key	4	4
SSL_get_shared_ciphers	4	4
SSL_in_before	4	4
SSL_num_renegotiations	4	6
SSL_set_session_secret_cb	4	8
SSL_set_timeout	4	4
SSL_state	4	9
SSL_use_RSAPrivateKey	4	4
SSL_COMP_add_compression_method	3	4
SSL_CTX_set_cookie_generate_cb	3	4
SSL_CTX_set_cookie_verify_cb	3	4
SSL_CTX_set_psk_client_callback	3	3
SSL_CTX_set_tmp_ecdh_callback	3	3
SSL_SESSION_set_ex_data	3	6
SSL_add_client_CA	3	3
SSL_dup	3	5
SSL_get_rfd	3	5
SSL_get_secure_renegotiation_support	3	7
SSL_get_ssl_method	3	3
SSL_get_tlsext_status_exts	3	3
SSL_renegotiate_pending	3	6
SSL_set_psk_client_callback	3	5
SSL_set_session_ticket_ext	3	3
SSL_set_session_ticket_ext_cb	3	6
SSL_set_state	3	6
SSL_set_tmp_dh_callback	3	3
SSL_set_tmp_rsa_callback	3	3
SSL_use_PrivateKey_ASN1	3	6
SSL_use_RSAPrivateKey_ASN1	3	3
SSL_use_certificate_ASN1	3	3
SSL_CONF_cmd_value_type	2	2

SSL_CTX_add_session	2	2
SSL_CTX_build_cert_chain	2	2
SSL_CTX_set_current_cert	2	4
SSL_CTX_set_max_send_fragment	2	2
SSL_CTX_set_srp_cb_arg	2	2
SSL_CTX_set_srp_username_callback	2	2
SSL_CTX_set_ssl_version	2	9
SSL_CTX_use_RSAPrivateKey_ASN1	2	2
SSL_CTX_use_psk_identity_hint	2	2
SSL_SESSION_new	2	3
SSL_SESSION_set_time	2	2
SSL_add_dir_cert_subjects_to_stack	2	2
SSL_cache_hit	2	2
SSL_get0_param	2	2
SSL_get_current_expansion	2	2
SSL_get_psk_identity	2	2
SSL_get_srp_userinfo	2	2
SSL_get_srp_username	2	6
SSL_set_client_CA_list	2	2
SSL_set_current_cert	2	2
SSL_set_mtu	2	4
SSL_set_srp_server_param	2	2
d2i_SSL_SESSION_bio	2	2
sk_SSL_CIPHER_find	2	8
ssl3_setup_buffers	2	3
ssl_init_wbio_buffer	2	2
BIO_new_buffer_ssl_connect	1	1
DTLS_method	1	1
DTLSv1_2_client_method	1	1
DTLSv1_2_method	1	1
DTLSv1_2_server_method	1	1
PEM_read_SSL_SESSION	1	2
PEM_read_bio_SSL_SESSION	1	1
PEM_write_bio_SSL_SESSION	1	1
SSL_CIPHER_find	1	1
SSL_CTX_add_server_custom_ext	1	1
SSL_CTX_get0_chain_certs	1	1
SSL_CTX_get0_param	1	1
SSL_CTX_get0_privatekey	1	1
SSL_CTX_get_tlsext_ticket_keys	1	1
SSL_CTX_set_client_cert_engine	1	1
SSL_CTX_set_default_read_ahead	1	1
SSL_CTX_set_msg_callback_arg	1	1
SSL_CTX_set_srp_password	1	1
SSL_CTX_set_srp_username	1	1
SSL_CTX_set_tlsext_ticket_keys	1	1
SSL_alert_type_string	1	1

SSL_clear_mode	1	1
SSL_dup_CA_list	1	2
SSL_get_wfd	1	1
SSL_has_matching_session_id	1	1
SSL_is_server	1	1
SSL_set1_chain	1	1
SSL_set_alpn_protos	1	1
SSL_set_max_cert_list	1	3
SSL_set_max_send_fragment	1	1
i2d_SSL_SESSION_bio	1	1
BIO_ssl_copy_session_id	0	0
DTLS_client_method	0	0
DTLS_get_link_min_mtu	0	0
DTLS_server_method	0	0
DTLS_set_link_mtu	0	0
DTLSv1_listen	0	0
OPENSSL_1.0.0	0	0
OPENSSL_1.0.1	0	0
OPENSSL_1.0.1d	0	0
OPENSSL_1.0.2	0	0
OPENSSL_1.0.2g	0	0
OPENSSL_freeFunc	0	0
OPENSSL_free_locked	0	0
PEM_write_SSL_SESSION	0	0
SRP_Calc_A_param	0	0
SRP_generate_client_master_secret	0	0
SRP_generate_server_master_secret	0	0
SSL_COMP_free_compression_methods	0	0
SSL_COMP_set0_compression_methods	0	0
SSL_CONF_CTX_clear_flags	0	0
SSL_CONF_CTX_set1_prefix	0	0
SSL_CONF_CTX_set_ssl	0	0
SSL_CONF_cmd_argv	0	0
SSL_CTX_SRP_CTX_free	0	0
SSL_CTX_SRP_CTX_init	0	0
SSL_CTX_add0_chain_cert	0	0
SSL_CTX_add1_chain_cert	0	0
SSL_CTX_add_client_custom_ext	0	0
SSL_CTX_callback_ctrl	0	0
SSL_CTX_clear_cert_flags	0	0
SSL_CTX_clear_chain_certs	0	0
SSL_CTX_clear_mode	0	0
SSL_CTX_get_client_cert_cb	0	0
SSL_CTX_get_default_read_ahead	0	0
SSL_CTX_get_extra_chain_certs_only	0	0
SSL_CTX_get_info_callback	0	0
SSL_CTX_get_max_cert_list	0	0

SSL_CTX_get_quiet_shutdown	0	0
SSL_CTX_get_read_ahead	0	0
SSL_CTX_get_ssl_method	0	0
SSL_CTX_select_current_cert	0	0
SSL_CTX_sess_get_get_cb	0	0
SSL_CTX_sess_get_new_cb	0	0
SSL_CTX_sess_get_remove_cb	0	0
SSL_CTX_sessions	0	0
SSL_CTX_set0_chain	0	0
SSL_CTX_set0_chain_cert_store	0	0
SSL_CTX_set0_verify_cert_store	0	0
SSL_CTX_set1_chain	0	0
SSL_CTX_set1_chain_cert_store	0	0
SSL_CTX_set1_client_certificate_types	0	0
SSL_CTX_set1_client_sigalgs	0	0
SSL_CTX_set1_client_sigalgs_list	0	0
SSL_CTX_set1_curves	0	0
SSL_CTX_set1_curves_list	0	0
SSL_CTX_set1_param	0	0
SSL_CTX_set1_sigalgs	0	0
SSL_CTX_set1_sigalgs_list	0	0
SSL_CTX_set1_verify_cert_store	0	0
SSL_CTX_set_cert_flags	0	0
SSL_CTX_set_default_verify	0	0
SSL_CTX_set_generate_session_id	0	0
SSL_CTX_set_max_cert_list	0	0
SSL_CTX_set_srp_client_pwd_callback	0	0
SSL_CTX_set_srp_strength	0	0
SSL_CTX_set_srp_verify_param_callback	0	0
SSL_CTX_set_tlsext_opaque_prf_input_callback	0	0
SSL_CTX_set_tlsext_opaque_prf_input_callback_arg	0	0
SSL_CTX_set_trust	0	0
SSL_CTX_use_PrivateKey_ASN1	0	0
SSL_CTX_use_certificate_ASN1	0	0
SSL_CTX_use_serverinfo	0	0
SSL_CTX_use_serverinfo_file	0	0
SSL_SESSION_get0_peer	0	0
SSL_SESSION_get_app_data	0	0
SSL_SESSION_print_fp	0	0
SSL_SESSION_set1_id_context	0	0
SSL_SESSION_set_app_data	0	0
SSL_SRP_CTX_free	0	0
SSL_SRP_CTX_init	0	0
SSL_add0_chain_cert	0	0
SSL_add1_chain_cert	0	0
SSL_add_file_cert_subjects_to_stack	0	0
SSL_add_session	0	0

SSL_alert_desc_string	0	0
SSL_build_cert_chain	0	0
SSL_callback_ctrl	0	0
SSL_certs_clear	0	0
SSL_check_chain	0	0
SSL_clear_cert_flags	0	0
SSL_clear_chain_certs	0	0
SSL_clear_num_renegotiations	0	0
SSL_ctrl	0	0
SSL_extension_supported	0	0
SSL_flush_sessions	0	0
SSL_get0_certificate_types	0	0
SSL_get0_chain_certs	0	0
SSL_get0_ec_point_formats	0	0
SSL_get0_raw_cipherlist	0	0
SSL_get1_curves	0	0
SSL_get_info_callback	0	0
SSL_get_max_cert_list	0	0
SSL_get_peer_signature_nid	0	0
SSL_get_psk_identity_hint	0	0
SSL_get_quiet_shutdown	0	0
SSL_get_read_ahead	0	0
SSL_get_servername_type	0	0
SSL_get_shared_curve	0	0
SSL_get_shared_sigalgs	0	0
SSL_get_sigalgs	0	0
SSL_get_srp_N	0	0
SSL_get_srp_g	0	0
SSL_get_srtp_profiles	0	0
SSL_get_time	0	0
SSL_get_timeout	0	0
SSL_get_tlsext_heartbeat_pending	0	0
SSL_get_tlsext_status_ids	0	0
SSL_get_verify_callback	0	0
SSL_heartbeat	0	0
SSL_need_tmp_RSA	0	0
SSL_remove_session	0	0
SSL_renegotiate_abbreviated	0	0
SSL_rstate_string	0	0
SSL_rstate_string_long	0	0
SSL_select_current_cert	0	0
SSL_set0_chain	0	0
SSL_set0_chain_cert_store	0	0
SSL_set0_verify_cert_store	0	0
SSL_set1_chain_cert_store	0	0
SSL_set1_client_certificate_types	0	0
SSL_set1_client_sigalgs	0	0

SSL_set1_client_sigalgs_list	0	0
SSL_set1_curves	0	0
SSL_set1_curves_list	0	0
SSL_set1_param	0	0
SSL_set1_sigalgs	0	0
SSL_set1_sigalgs_list	0	0
SSL_set1_verify_cert_store	0	0
SSL_set_cert_cb	0	0
SSL_set_cert_flags	0	0
SSL_set_debug	0	0
SSL_set_ecdh_auto	0	0
SSL_set_generate_session_id	0	0
SSL_set_pref_cipher	0	0
SSL_set_psk_server_callback	0	0
SSL_set_purpose	0	0
SSL_set_srp_server_param_pw	0	0
SSL_set_time	0	0
SSL_set_tlsext_debug_arg	0	0
SSL_set_tlsext_debug_callback	0	0
SSL_set_tlsext_heartbeat_no_requests	0	0
SSL_set_tlsext_opaque_prf_input	0	0
SSL_set_tlsext_status_exts	0	0
SSL_set_tlsext_status_ids	0	0
SSL_set_tlsext_use_srtp	0	0
SSL_set_tmp_dh	0	0
SSL_set_tmp_ecdh	0	0
SSL_set_tmp_ecdh_callback	0	0
SSL_set_tmp_rsa	0	0
SSL_set_trust	0	0
SSL_srp_server_param_with_username	0	0
SSL_test_functions	0	0
SSL_use_psk_identity_hint	0	0
SSL_want_nothing	0	0
SSL_want_x509_lookup	0	0
dtls1_process_heartbeat	0	0
lh_SSL_SESSION_free	0	0
lh_SSL_SESSION_new	0	0
sk_SSL_CIPHER_find_ex	0	0
tls1_process_heartbeat	0	0

References

[1] Let's encrypt: Delivering SSL/TLS everywhere. `https://letsencrypt.org/2014/11/18/announcing-lets-encrypt.html`. Accessed: 2018-01-09.

[2] Repositories. `https://help.ubuntu.com/community/Repositories`. Accessed: 2018-01-10.

[3] Acharya, M., Xie, T., Pei, J., and Xu, J. Mining api patterns as partial orders from source code: from usage scenarios to specifications. In *ACM International Symposium on Foundations of Software Engineering (FSE)* (2007), ACM, pp. 25–34.

[4] Amour, L. S., and Petullo, W. M. Improving application security through TLS-library redesign. In *Security, Privacy, and Applied Cryptography Engineering (SPACE)* (2015), Springer, pp. 75–94.

[5] Bates, A., Pletcher, J., Nichols, T., Hollembaek, B., Tian, D., Butler, K. R., and Alkhelaifi, A. Securing SSL certificate verification through dynamic linking. In *ACM Conference on Computer and Communications Security (CCS)* (2014), pp. 394–405.

[6] Bhargavan, K., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.-Y., and Zanella-Béguelin, S. Proving the TLS handshake secure (as it is). In *International Cryptology Conference (CRYPTO)* (2014), Springer, pp. 235–255.

[7] Bhargavan, K., Lavaud, A. D., Fournet, C., Pironti, A., and Strub, P. Y. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *IEEE Symposium on Security and Privacy (SP)* (2014), IEEE, pp. 98–113.

[8] Duong, T., and Rizzo, J. The CRIME attack. In *Presentation at ekoparty Security Conference* (2012).

[9] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., and Smith, M. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM Conference on Computer and Communications Security (CCS)* (2012), ACM, pp. 50–61.

[10] Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. Rethinking SSL development in an appified world. In *ACM Conference on Computer and Communications Security (CCS)* (2013), ACM, pp. 49–60.

[11] Foundation, O. S. 1.0.2 manpages. https://www.openssl.org/docs/man1.0.2/ssl/SSL_CTX_new.html. Accessed: 15 December 2017.

[12] Fowkes, J., and Sutton, C. Parameter-free probabilistic API mining across GitHub. In *ACM International Symposium on Foundations of Software Engineering (FSE)* (2016), ACM, pp. 254–265.

[13] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and Shmatikov, V. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security (CCS)* (2012), ACM, pp. 38–49.

[14] Giesen, F., Kohlar, F., and Stebila, D. On the security of TLS renegotiation. In *ACM Conference on Computer and Communications Security (CCS)* (2013), ACM, pp. 387–398.

[15] Gluck, Y., Harris, N., and Prado, A. BREACH: Reviving the CRIME attack. *Unpublished manuscript* (2013).

[16] He, B., Rastogi, V., Cao, Y., Chen, Y., Venkatakrishnan, V., Yang, R., and Zhang, Z. Vetting SSL usage in applications with SSLint. In *IEEE Symposium on Security and Privacy (SP)* (2015), IEEE, pp. 519–534.

[17] Jager, T., Kohlar, F., Schäge, S., and Schwenk, J. On the security of TLS-DHE in the standard model. In *International Cryptology Conference (CRYPTO)*. Springer, 2012, pp. 273–293.

[18] Kagdi, H., Collard, M. L., and Maletic, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software: Evolution and Process 19*, 2 (2007), 77–131.

[19] Krawczyk, H., Paterson, K. G., and Wee, H. On the security of the TLS protocol: A systematic analysis. In *International Cryptology Conference (CRYPTO)*. Springer, 2013, pp. 429–448.

[20] Malinen, J. TLS session ticket extension problem when using the ssl23_client_hello method. https://mta.openssl.org/pipermail/openssl-dev/2015-July/002162.html. Accessed: 15 January, 2018.

[21] Mavrogiannopoulos, N. Fedora system-wide crypto policy. http://fedoraproject.org/wiki/Changes/CryptoPolicy. Accessed: 15 December 2017.

[22] Mller, B. This is POODLE bites: Exploiting the SSL 3.0 fallback. https://www.openssl.org/~bodo/ssl-poodle.pdf.

[23] Oliveira, D., Rosenthal, M., Morin, N., Yeh, K.-C., Cappos, J., and Zhuang, Y. It's the psychology stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *Annual Computer Security Applications Conference (ACSAC)* (2014), ACM, pp. 296–305.

[24] O'Neill, M., Heidbrink, S., Ruoti, S., Whitehead, J., Bunker, D., Dickinson, L., Hendershot, T., Reynolds, J., Seamons, K., and Zappala, D. Trustbase: An architecture to repair and strengthen certificate-based authentication. In *USENIX Security Symposium* (2017).

[25] Paterson, K. G., Ristenpart, T., and Shrimpton, T. Tag size does matter: Attacks and proofs for the TLS record protocol. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)* (2011), vol. 7073, Springer, pp. 372–389.

[26] Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., and Khan, L. SMV-hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in Android apps. In *Network and Distributed System Security Symposium (NDSS)* (2014).

[27] Wang, J., Dang, Y., Zhang, H., Chen, K., Xie, T., and Zhang, D. Mining succinct and high-coverage api usage patterns from source code. In *IEEE Conference on Mining Software Repositories (MSR)* (2013), IEEE, pp. 319–328.

[28] Woo, T. Y., Bindignavle, R., Su, S., and Lam, S. S. SNP: An interface for secure network programming. In *USENIX Summer* (1994), pp. 45–58.

[29] Xie, T., and Pei, J. Mapo: Mining api usages from open source repositories. In *International Workshop on Mining Software Repositories* (2006), ACM, pp. 54–57.

[30] Yamaguchi, F., Golde, N., Arp, D., and Rieck, K. Modeling and discovering vulnerabilities with code property graphs. In *IEEE Symposium on Security and Privacy (SP)* (2014), IEEE, pp. 590–604.