Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.27, NO.3, Jun. 2017 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2017.27.3.459

OpenSSL 기반 사용자 지정 암호 프로토콜 구현 방안*

임 준 휘,† 이 상 곤,‡ 이 훈 재, 인센티우스 크리스티안 안드리안토
동서대학교

Custom Cryptographic Protocol Implementation Method Based on OpenSSL*

JunHuy Lam,† Sang-Gon Lee,‡ Hoon-Jae Lee, Vincentius Christian Andrianto
Dongseo University

요 약

가장 널리 사용되는 오픈 소스 프로젝트 중 하나인 OpenSSL은 대부분의 웹 사이트, 서버 및 클라이언트를 보호하는 데 사용되는 암호화 라이브러리이다. OpenLLS을 사용하여 SSL 혹은 이것으로부터 나온 TLS 프로토콜로 안전하게 통신할 수 있다. 시스템의 지속적 보안성 유지를 위해 암호 프로토콜은 업데이트되고 개선되어야 하므로, 이 라이브러리는 새로운 암호화 방법을, 특히 대칭 암호화 프로토콜을, 시스템에 통합하여 구현하는 일이 간단하도록 작성되었다. 그러나 비대칭 암호화 프로토콜을 추가 할 때는 훨씬 더 복잡해진다. 본 논문에서는 OpenSSL 기반 사용자 지정 암호 프로토콜 구현 방안 도출을 위하여 OpenSSL 라이브러리의 세부 아키텍처를 파악하고 설명한다. 그리고 대칭키와 비대칭 암호화 기반 사용자 지정 프로토콜을 수용하기 위해 OpenSSL 라이브러리를 수정하는 방법을 제시한다.

ABSTRACT

One of the most widely-used open source project; OpenSSL is a cryptography library that is used to secure most web sites, servers and clients. One can secure the communication with the Secure Socket Layer (SSL) or its successor, Transport Layer Security (TLS) protocols by using the OpenSSL library. Since cryptography protocols will be updated and enhanced in order to keep the system protected, the library was written in such a way that simplifies the integration of new cryptographic methods, especially for the symmetric cryptography protocols. However, it gets a lot more complicated in adding an asymmetric cryptography protocol and no guide can be found for the integration of the asymmetric cryptography protocol. In this paper, we explained the architecture of the OpenSSL library and provide a simple tutorial to modify the OpenSSL library in order to accommodate custom protocols of both symmetric and asymmetric cryptography.

Keywords: OpenSSL architecture, custom security protocol, custom asymmetric protocol, custom symmetric protocol

I. Introduction

One of the most widely-used open source project; OpenSSL is a cryptography library that is used to secure most web sites, servers and clients in the Internet. One can secure the communication with Secure Socket Layer (SSL) or its successor, Transport Layer Security (TLS) protocols by using the OpenSSL library.

Since cryptography protocols will be updated and enhanced in order to keep the system protected, the library was written in a way that simplifies the

Received(03. 16. 2017), Modified(04. 26. 2017), Accepted(04. 27. 2017)

* 본 논문은 2017년도 영남지부 학술대회에 발표한 우수논문을 개선 및 확장한 것임

* 본 연구는 2016년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행되었음(Grant Number: 2014R1A1A2060021).

† 주저자, timljh@msn.com

‡ 교신저자, nok60@dongseo.ac.kr(Corresponding author)

integration of new cryptographic methods, especially for the symmetric cryptography protocols. However, it becomes a lot more complicated in modifying or adding an asymmetric cryptography protocol and no guide can be found for this integration.

Most of the OpenSSL guide provides tutorials or references on how to utilize, configure or deploy the library through command lines, the Application Program Interface (API) or configuration settings [1-3] but the guide on how to modify or enhance the library can hardly be found. This makes the coding of a new cryptographic protocol extremely difficult.

In this paper, we described and explained the detailed architecture of OpenSSL to ease the integration of new symmetric and asymmetric protocols for the future developers. Our contributions for this paper are listed as below,

- Explained the architecture of OpenSSL library
- Described the integration process of the custom symmetric and asymmetric cryptography protocols

An overview of OpenSSL library and related works can be found in Chapter 2. Chapter 3 describes the architecture of OpenSSL library for integration of custom protocol while the installation of the customized OpenSSL library can be found in Chapter 4. Finally, the paper will be concluded in Chapter 5.

II. Background

2.1 Overview of OpenSSL

OpenSSL is widely used to secure the Internet and hence the effect of a bug in its code can be severe. This was shown in the recent critical bug that exposes a serious security vulnerability; the Heartbleed. Heartbleed[4] is a critical bug found in OpenSSL library that allows attackers to read beyond the buffer's data or it was better known as buffer over-read.

By doing so, sensitive information that was protected by the secure channel established with OpenSSL library can be exposed to attackers and this can render secure channel useless. This critical bug causes havoc over the Internet as an estimation of 55% popular HyperText Transport Protocol (Secure) (HTTPS) web sites were secured by the vulnerable OpenSSL library[5]. Besides that, the vulnerable OpenSSL library was also used to secure other protocols such as e-mails, remote access sessions and etc.

This raises concern of secure coding and why does it take so long (more than 2 years) for the users/developers to discover this bug despite the project being open source. This is mainly due to the complicated structure[6] of the OpenSSL library that makes it difficult to understand even for software analyzing tools.

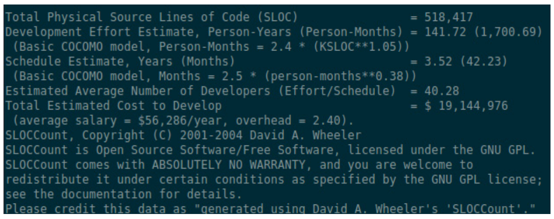
Kupsch and Miller[6] explained that the difficulties for the code analysis tools in identifying the vulnerability are as follow,

- The use of pointers (Chain of pointers led by the vulnerable function)
- The complexity of the execution path from the bigger allocation to its misuse
- The valid bytes of the TLS message are a subset of the allocated buffer
- The contents of the buffer do not appear to come directly from attackers

Besides that, the Heartbleed bug also

led to a forked version of OpenSSL due to the complicated and messy code base. The forked OpenSSL; LibreSSL[7] tries to clean up the code and make it easier for the developers to understand, audit and repair the library.

In the first week alone, LibreSSL managed to remove half of the code[8] from the library. However, it is no longer compatible with the project that developed with OpenSSL and it also supports less operating system (OS) compared to OpenSSL. Fig. 1 shows the source code analysis of the OpenSSL library by using the sloccount[9] tool and the estimated effort and time required to develop the library. It shows the size of the OpenSSL library is enormous given the long history of the open source cryptography library.



```
Total Physical Source Lines of Code (SLOC)                = 518,417
Development Effort Estimate, Person-Years (Person-Months) = 141.72 (1,700.69)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 3.52 (42.23)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 40.28
Total Estimated Cost to Develop                           = $ 19,144,976
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Fig. 1. Source code analysis of OpenSSL library by using sloccount tool

2.2 Related Works

Armknecht et al. [3] implemented a trusted channel based on TLS specification by using the OpenSSL library. They claimed that it helps in ensuring the integrity of an endpoint software in order to prevent attacks that compromise the endpoints through malicious code injection.

However, their work is only a proof of concept (PoC) prototype instead of an actual implementation in code. They also do not explain on the integration of their work with OpenSSL library in term of coding.

Yilek et al.[10] explained the significance of the OpenSSL code and how it can compromises security when a bad code was introduced. It allows attackers to make use of the loophole or bug and then compromise the secure channel.

Despite that, they do not show the proper way to integrate any new protocols to the OpenSSL library. They only surveyed on the existing SSL/TLS-enabled web servers whether the administrators patched up their servers or not.

Jurkiewicz and Niemiec[11] tried to implement a symmetric block cipher with OpenSSL, they provide a simple guide on how to code for the symmetric block cipher with OpenSSL. However, they do not provide any guide on the asymmetric ciphers and their guide on the symmetric block cipher also missed out a few key points such as the libcrypto.num, libssl.num and buildinfo files.

III. The Architecture of OpenSSL

In general, OpenSSL library can be divided into two sub-libraries as illustrated in Fig.

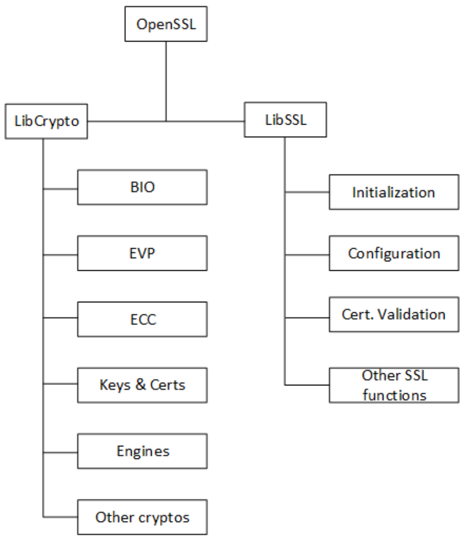


Fig. 2. The Architecture of OpenSSL library

2, namely the libCrypto and libSSL libraries.

LibCrypto library consists of the generic cryptographic functions which include the parameters initialization, random number generator, error handling, other crypto functions and configurations of the cryptographic functions.

This component of OpenSSL library is located under the directory of "crypto". LibCrypto library can be further divided into sub-modules,

- Basic Input/Output (BIO)
- Envelope (EVP)
- Elliptic Curve Cryptography (ECC)
- Keys & Certificates
- Engines
- Other crypto functions (Diffie Hellman, Bignum and etc)

LibSSL library consists of the SSL/TLS protocols implementation that make use of the LibCrypto library to establish the secure channel. This includes the channel/session establishment, management, resumption, authentication and configuration. Similar to LibCrypto, LibSSL library can be further divided into sub-modules,

- Connection/Session initialization
- Connection configuration
- Certificate validation
- Other SSL/TLS functions (Session resumption, Datagram TLS (DTLS), Alternative key exchange and etc)

3.1 LibCrypto

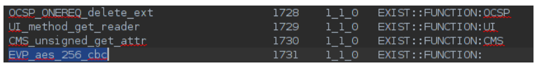
LibCrypto generally covered the symmetric cryptographic functions and general purpose cryptographic functions. All of the LibCrypto library's function pointers are listed in a file named, libcrypto.num. This file labels all of the symmetric cryptographic functions as well as the rest of the cryptographic functions with sequential numbers. This is one of the required file that was not mentioned by Jurkiewicz and Niemiec[11] as stated in the related works.

In order to add a new symmetric cryptographic function to the LibCrypto library, the first step will be adding a new number that will be pointed by the function name to util/libcrypto.num. Then, the header of the cipher can be added under include/openssl/cipher_name.h. Since symmetric cipher's code can be identical but in reverse order, the encrypt mode is denoted as 1 in this header file while the decrypt mode is denoted as 0.

For a new symmetric crypgraphic function, the crypto/evp/c_allc.c should be updated with the new cipher along with its name while for a new message digest algorithm, crypto/evp/c_alld.c should be updated with the new digest function along with its name. The EVP module of LibCrypto consists of a list of function calls for each of the cipher and message digest algorithms. If new properties such as different maximum key length is required, it can be added or modified in crypto/evp/evp.h.

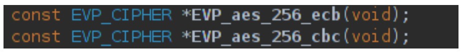
Fig. 3 shows part of the libcrypto.num file that contains the function pointer of EVP function for Advanced Encryption Standard (AES) 256 Cipher Block Chaining (CBC) mode. This function pointer is then lead to the evp.h and evp/c_allc.h header files as shown in Fig. 4 and Fig. 5.

Lastly, the detailed implementation of the cipher will be added under the directory crypto/cipher_name. OpenSSL



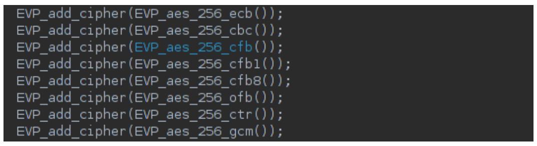
```
OCSP_ONEREQ_delete_ext                    1728    1_1_0   EXIST::FUNCTION:OCSP
UI_method_get_reader                      1729    1_1_0   EXIST::FUNCTION:UI
CMS_unsigned_get_attr                     1730    1_1_0   EXIST::FUNCTION:CMS
EVP_aes_256_cbc                           1731    1_1_0   EXIST::FUNCTION:
```

Fig. 3. Part of the libcrypto.num file



```
const EVP_CIPHER *EVP_aes_256_ecb(void);
const EVP_CIPHER *EVP_aes_256_cbc(void);
```

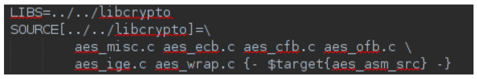
Fig. 4. Part the evp.h file



```
EVP_add_cipher(EVP_aes_256_ecb());
EVP_add_cipher(EVP_aes_256_cbc());
EVP_add_cipher(EVP_aes_256_cfb());
EVP_add_cipher(EVP_aes_256_cfb1());
EVP_add_cipher(EVP_aes_256_cfb8());
EVP_add_cipher(EVP_aes_256_ofb());
EVP_add_cipher(EVP_aes_256_ctr());
EVP_add_cipher(EVP_aes_256_gcm());
```

Fig. 5. Part of the evp/c_allc file

manages the source files with the a file named buildinfo under each directory, it lists all of the source files that will be added to the generated makefile and compile accordingly. Make sure the buildinfo file is updated with the addition of the new source files, Jurkiewicz and Niemiec[11] also failed to mention this buildinfo file that is needed to be modified in order to add new source files into the library. Fig. 6 shows the buildinfo file for the AES module, it needs to include all of the source files for the AES implementation in order to be compiled.



```
LIBS=../../libcrypto
SOURCE[../../libcrypto]=\
        aes_misc.c aes_ecb.c aes_cfb.c aes_ofb.c \
        aes_ige.c aes_wrap.c {- $target{aes_asm_src} -}
```

Fig. 6. Part of the buildinfo file for AES module

3.2 LibSSL

LibSSL on the other hand covers the asymmetric cryptographic functions. Similar to the symmetric cryptography, the SSL/TLS function pointers of the LibSSL library are listed in a file named, util/libssl.num. The newly added function in the LibSSL library had be to updated into this list of function pointers too. This is another required file that was not mentioned by Jurkiewicz and Niemiec[11] as stated in the related works as they were not working on asymmetric cryptography protocol.

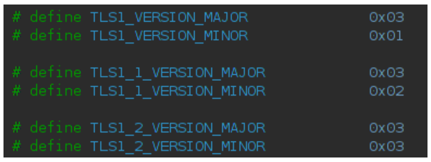
In the case of SSL/TLS protocols, the cipher identity, name and code number are listed in include/openssl/ssl.h and include/openssl/tls1.h depending on the protocol used. Besides that, the function headers can be found in the two header files too. These can be seen in Fig. 7 and 8 below.

If the new asymmetric cipher is not based on SSL or TLS protocols, a new header file can be added under include/openssl/new_asymmetric_cipher.h.

Unlike symmetric cryptography, the developer might need more than one function here for the client and server modes. Besides that, a general function can also be used for the procedures in which both the client and server modes share.

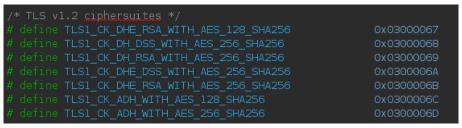
Therefore, in most cases, at least three functions can be declared here in order to add a new symmetric cryptographic function. For example, new_asymmetric_crypto_method, new_asymmetric_crypto_client_method and new_asymmetric_crypto_server_method.

Fig. 9 shows part of the libssl.num file that contains the TLS_client_method. Besides that, the other function pointers of the SSL/TLS protocol can also be found



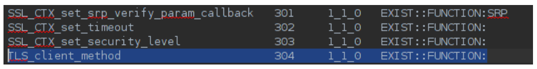
```
# define TLS1_VERSION_MAJOR             0x03
# define TLS1_VERSION_MINOR             0x01

# define TLS1_1_VERSION_MAJOR           0x03
# define TLS1_1_VERSION_MINOR           0x02

# define TLS1_2_VERSION_MAJOR           0x03
# define TLS1_2_VERSION_MINOR           0x03
```

Fig. 7. Part of the tls1.h file that contains the version numbers



```
/* TLS v1.2 ciphersuites */
# define TLS1_CK_DHE_RSA_WITH_AES_128_SHA256            0x03000067
# define TLS1_CK_DH_DSS_WITH_AES_256_SHA256             0x03000068
# define TLS1_CK_DH_RSA_WITH_AES_256_SHA256             0x03000069
# define TLS1_CK_DHE_DSS_WITH_AES_256_SHA256            0x0300006A
# define TLS1_CK_DHE_RSA_WITH_AES_256_SHA256            0x0300006B
# define TLS1_CK_ADH_WITH_AES_128_SHA256                0x0300006C
# define TLS1_CK_ADH_WITH_AES_256_SHA256                0x0300006D
```

Fig. 8. Part of the tls1.h file that contains the cipher suites



```
SSL_CTX_set_srp_verify_param_callback   301     1_1_0   EXIST::FUNCTION:SRP
SSL_CTX_set_timeout                     302     1_1_0   EXIST::FUNCTION:
SSL_CTX_set_security_level              303     1_1_0   EXIST::FUNCTION:
TLS_client_method                       304     1_1_0   EXIST::FUNCTION:
```

Fig. 9. Part of the libssl.num file

within the file.

After initializing these functions pointers, the developer can then proceed to code for the detailed implementation of the functions under ssl/new_cipher_lib.c. Once the code was written, the developer should add the newly created files to the buildinfo file so that they will be included in the generated makefile for compilation.

IV. Installation

If any third party library is required for the custom protocol, makes sure to include it in the makefile generator under the directory "Configurations" according to the OS used. To differentiate the customized version of OpenSSL library and easily verify the correct version was installed, the version name can be modified in the header file: include/openssl/opensslv.h. The definition of OPENSSL_VERSION_TEXT can also be modified to any identifiable name for the developer.

Typically, the "sudo make install" command can be used to install the compiled source code as part of the system libraries in Linux. Run the command "openssl version" to check whether the OpenSSL library was installed properly, the output should has the same version name the developer set in the OPENSSL_VERSION_TEXT as described earlier.

Once installed, the customized OpenSSL library can be used by any software within the system simply by including the customized library file. By default, the installed binary file is located at /usr/local/bin while the library files are located at /usr/local/lib. Lastly, the configuration file of the installed OpenSSL library is located at /usr/lib/ssl.

V. Conclusion

Most of the existing studies or analysis explain how to use the OpenSSL library but none of them actually explain how to enhance or modify the library with Jurkiewicz and Niemiec[11] being the only exception. Even that, they only explain the integration of a symmetric block cipher and do not include any guide on the integration of a custom asymmetric cipher. They also missed out a few key points such as the libcrypto.num, libssl.num and buildinfo files.

Our analysis and study on the OpenSSL can provide insights for developers or researchers that attempt to include or experiment a new cryptography protocol within the OpenSSL library. Hopefully, it can also helps to reduce the learning curve for new developers of OpenSSL library.

References

[1] Ivan Ristic, OpenSSL Cookbook, 2nd Ed., Feisty Duck Limited, UK, March 2015.

[2] Kenneth Ballard, "Secure programming with OpenSSL API," IBM developerWorks, June 2012.

[3] F. Armknecht, Y. Gasmi, and et.al., "An Efficient Implementation of Trusted Channels based on OpenSSL," Proceedings of the 3rd ACM workshop on Scalable trusted computing, pp. 41-50, Oct. 2008.

[4] M. Carvalho, J. DeMott, R. Ford, and D. Wheeler, "Heartbleed 101," IEEE Security & Privacy, 12(4), 63-67, July. 2014.

〔5〕 Z, Durumeric, J. Kasten and et al., ″The Matter of Heartbleed,″ Proceedings of the 2014 Conference on Internet Measurement Conference, pp 475-488, Nov. 2014.

〔6〕 J.A. Kupsch and B.P. Miller, “Why Do Software Assurance Tools Have Problems Finding Bugs Like Heartbleed?,” Continuous Software Assurance Marketplace, 22 Apr. 2014. Web. https://www.swampinabox.org/doc/SWAMP-WP003-Heartbleed.pdf

〔7〕 LibreSSL. OpenBSD Foundation. Web. https://www.libressl.org/goals.html

〔8〕 Jon B. ″OpenSSL code beyond repair, claims creator of ″LibreSSL″ fork″, Ars Technica, Apr. 2014.

〔9〕 D.A. Wheeler, ″Sloccount, 2008,″ Web. http://www. dwheeler. com/sloccount

〔10〕 S. Yilek, E. Rescorla, and et al., ″When private keys are public: results from the 2008 Debian OpenSSL vulnerability,″ Proceedings of the 9th ACM SIGCOMM conference on Internet measurement, pp.15-27, Nov. 2009.

〔11〕 P. Jurkiewicz and M. Niemiec, ″Implementation of a New Cipher in OpenSSL Environment the Case of INDECT Block Cipher,″ International Journal of Computer and Communication Engineering, 5(1), pp. 41-49, Jan. 2016.

〈저자소개〉

임 준 휘 (Jun-Huy Lam) 학생회원
2006년 7월: 말레이시아 Multimedia University 졸업
2010년 7월: 동서대학교 유비쿼터스IT학과 공학석사
2012년 8월~2014년 8월: GHL System Berhad, Malaysia (소프트웨어 엔지니어)
2014년 9월~현재: 동서대학교 대학원 유비쿼터스IT학과 박사과정
〈관심분야〉 SDN, 네트워크 보안, 소프트웨어 개발, 블록체인

이 상 곤 (Sang-gon Lee) 종신회원
1986년 2월: 경북대학교 전자공학과 졸업
1988년 2월: 경북대학교 전자공학과 석사
1993년 2월: 경북대학교 전자공학과 박사
1991년 3월~1997년 2월: 창신대학교 전자통신과 조교수
1997년 3월~현재: 동서대학교 컴퓨터공학부 교수
〈관심분야〉 암호이론, 암호프로토콜 및 네트워크 응용, 소프트웨어정의네트워크, 블록체인

이 훈 재 (Hoon-jae Lee) 종신회원
1985년 2월: 경북대학교 전자공학과 졸업
1987년 2월: 경북대학교 전자공학과 석사
1998년 2월: 경북대학교 전자공학과 박사
1987년 3월~1998년 2월: 국방과학연구소 선임연구원/팀장
1998년 3월~2002년 2월: 경운대학교 조교수
2002년 3월~현재: 동서대학교 컴퓨터공학부 교수
〈관심분야〉 암호이론, 네트워크보안, 부채널공격, 포렌식

빈센티우스 크리스티안 안드리안토(Vincentius Christian Andrianto) 학생회원
2015년 7월: 인도네시아 Petra Christian University 전자공학과 졸업
2015년 9월~현재: 동서대학교 대학원 유비쿼터스IT학과 석사과정
〈관심분야〉 SDN, 네트워크 보안, 인공지능, 보안 소프트웨어 개발