https://doi.org/10.48009/2_iis_2018_99-108
Issues in Information Systems
Volume 19, Issue 2, pp. 99-108, 2018

HEARTBLEED: A CASE STUDY

J.K. Harris, Georgia Southern University, jkharris@georgiasouthern.edu

ABSTRACT

On April 7th, 2014 a flaw in OpenSSL was simultaneously announced independently by researchers at Google and Codenomicon, a Finnish software testing company. The flaw was in an OpenSSL extension called Heartbeat that allowed for a buffer over-read of heap memory by any external client connected through OpenSSL possibly exposing the private keys used by OpenSSL. The flaw went undetected for over two years as the affected versions of OpenSSL dated back to March, 2012. It is estimated that the flaw affected over 500,000 servers around the world, including those at Google, Amazon, Cisco, Dell, Intel and Facebook (NetCraft, 2014). This paper reports an analysis of OpenSSL code using software metrics in versions of OpenSSL before, during, and after Heartbleed and their possible effects on software assurance, looks at the OpenSSL coding, design and testing standards and practices that contributed to the flaw and to the severity of the flaw and makes suggestions for future changes to the corresponding processes.

Keywords: Heartbleed, OpenSSL, Best Practices, Secure Coding, Secure Testing, Secure Design, FIPS 140-2, Software Metrics

INTRODUCTION: THE IETF AND OPENSSL

The Internet Engineering Task Force (IETF) was formed in 1986 as a platform to coordinate contractors for the U.S. Defense Advanced Projects Agency (DARPA) and the Internet core gateway system. The IETF was initially supported by the U.S. federal government, but since 1993, it has operated as a standards development agency under the auspices of the Internet Society, an international membership-based non-profit organization (IETF, February 2006).

The IETF mission is as follows:

- To identify operational and technical problems in the Internet and propose solutions to them.
- To specify the development and use of protocols and internet architecture and find solutions to technical problems.
- To allow smooth transfer of technology from the Internet Research Task Force (IRTF) to the international Internet community.
- To create a platform for the exchange of crucial information between users, researchers, vendors, operators, agency contractors and network managers.

The IETF has working groups that cover nine function areas. Each area is led by Area Directors, who together with the Chairman of IETF form the Internet Engineering Steering Group or the IESG. The IETF groups comprise of the following areas of function:

- Applications: User area Service
- IP: Next Generation
- Operational Requirements
- Security
- Transport
- Internet
- Network Management
- Routing

The IETF's request for comments (RFC's) contains a set of numbered documents defining standards for the most common Internet protocols. RFC 2246, defining the Transport Layer Security (TLS) Protocol version 1.0, was published in January of 1999 (IETF, 1999). TLS was established as a protocol whose main functions are to provide authentication and confidentiality during TCP connections. The TLS working group falls under the IETF Security group.

The OpenSSL Project was founded in December, 1998 to create a free set of encryption tools. It was a fork of an open source project called SSLeay developed by Eric Young and Tim Andrews. Just after they joined the RSA Corporation in 1998 they released SSLeay for development by the Open Source Community (Wikipedia, March 2018). Before publishing a standard as an RFC, the IETF likes to have a working version of the standard. The OpenSSL development team has worked closely with the IETF's TLS working group to produce working version(s) of TLS and its extentions.

THE HEARTBLEED FLAW

In April, 2006 the Datagram Transport Layer Security (DTLS) protocol was published by the IETF as RFC 4347 (IETF, March 2006). DTLS allowed for the encryption of User Datagram Protocol (UDP) packets. Since UDP is a connectionless protocol, the sending peer never receives acknowledgements from the receiving peer. The receiving peer could be offline or simply not listening. There is no way for the sender to know. The Heartbeat extension proposal was made to the IETF so that either end of TLS/DTLS connection could request a string of up to 64K-1 characters to be echoed back. This would allow either end of a connection to verify that the opposite endpoint was still listening. In January, 2012, working versions of this extention were implemented by the ODG in OpenSSL 1.0.1 beta versions (NLUUG, 2018). In February, 2012, RFC 6520, Transport Layer Security (TLS) and DTLS Heartbeat Extension was published by the IETF (IETF, 2012). Then in March, 2012, the Heartbeat extension appeared in the non-beta version of OpenSSL version 1.0.1 (NLUUG, 2018). A little over two years passed with version 1.0.1f being the current released version when on April 7th, 2014, researchers at Google and Codenomicon, a Finnish software testing company, announced a buffer over-read flaw in OpenSSL. Software testers at Codenomicon claimed they could read multiple 64KB segments of memory on a remote server running the affected code from a browser and were able to access "encryption keys, usernames and passwords plus a lot of other stuff that we don't want to mention" (Markowitz, 2014). The flaw was zero-day for over two years as the affected versions of OpenSSL dated back to March, 2012. It is estimated that the flaw affected over 500,000 servers around the world, including those at Google, Amazon and Facebook and possibly exposed much of the data encrypted by those servers over the lifetime of their private keys have far exceeded the two years the flaw existed (NetCraft, 2014). It also meant the thieves could have digitally signed documents with the compromised private keys or set up Web sites that would be authenticated as the original private key owner's Web site until those keys were revoked, essentially posing online as the owners themselves. Figure 1 shows the number of revoked certificates the days following the disclosure of the Heartbleed flaw (Zhang et al., 2018)).

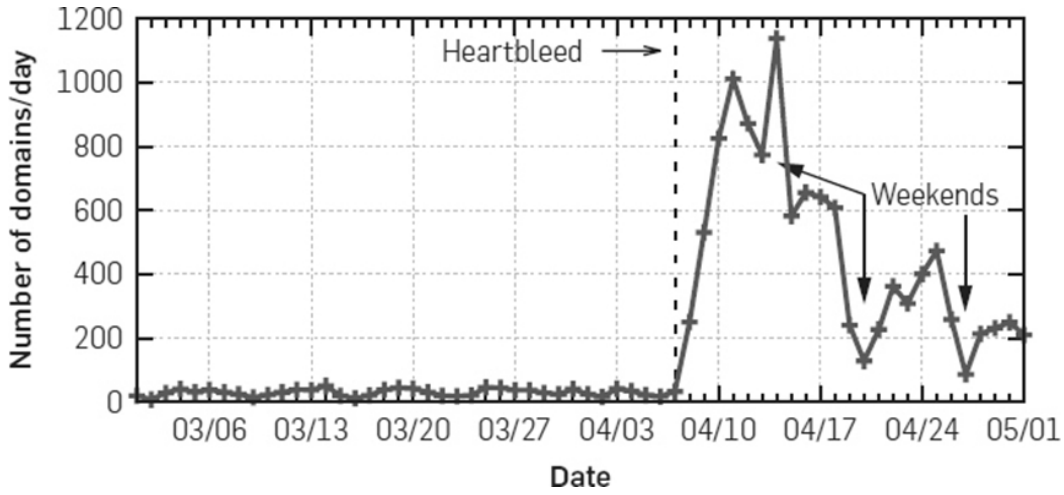

Figure 1. From (Zhang, et al., 2018), the Number of Certificates Revoked per Day Following the Heartbleed Disclosure

Applications, such as ssh and sftp clients that incorporated OpenSSL 1.0.1 through OpenSSL 1.0.1f were (and very often still are) vulnerable to Heartbleed. This includes many mobile banking apps that were developed using those versions (Jauregui, 2017). In January, 2017 it was reported that "Over 199,500 websites were still vulnerable to the Heartbleed OpenSSL bug. The countries most affected by Heartbleed still remain the United States, followed by Korea, China, Germany, France, Russian Federation, United Kingdom, India, Brazil and Italy." (Khandelwal, 2017). A more detailed description of the Heartbeat extension and the Heartbleed flaw are given in Kyatam S. et al. (2017) and a detailed timeline of the events that occurred before, during, and after the discovery of the Heartbleed flaw is given in Durumeric Z. et al. (2014).

AN ANALYSIS OF OPENSSL CODE USING SOFTWARE METRICS

Figures 2 and 3 show several software metrics in OpenSSL software for an early version of OpenSSL (version 0.9.6, September, 2000) and the Heartbleed version (version 1.0.1, March, 2012). These versions were from the achive in NLUUG (2018).

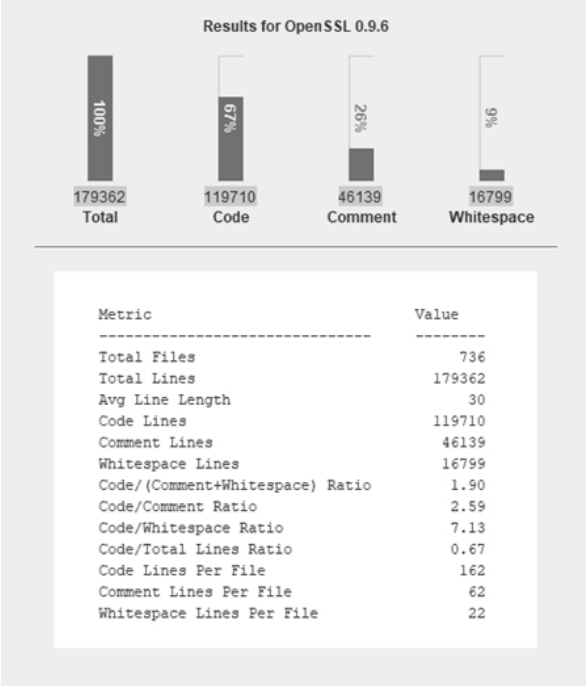

Figure 2. Software Metrics for OpenSSL v 0.9.6

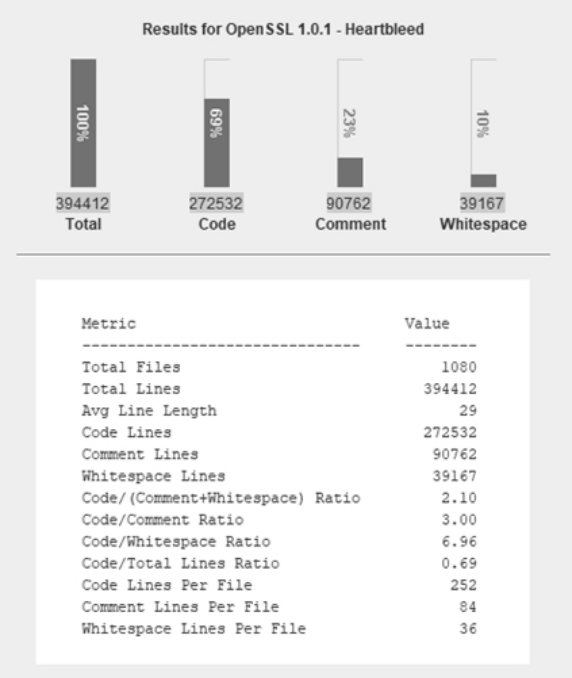

Figure 3. Software Metrics for OpenSSL v 1.0.1

The number of lines of code from OpenSSL version 0.9.6 to version 1.0.1 (the Heartbleed version) increased by about 128% from 119,710 to 272,532. The code/comment ratio went from 2.59 to 3.0 indicated decreased comment density (which is the ration of the number of lines of code to the number of lines of comments) within the code for the Heartbleed version. The main reason the code/comment ratio seems low is because the licensing agreements (there are two, the original SSLeay license and the OpenSSL license) are contained within each source file and are about 100 lines long. The code lines per file went from 162 to 252, indicating an increase of around 55%. Each of these metrics point to the fact that the complexity of code increased and the maintainabilty decreased, undoubtably as a result of "feature creep". Feature creep is the addition of new features to software. The changelog for OpenSSL between version 1.0.0h and the subsequent version 1.0.1 (the initial Heartbleed version) shows 55 changes with many of those changes implementing new features, including the Heartbeat extention (OpenSSL.org, May 2018). Figure 4 show the same software metrics for the current version of OpenSSL (released in April, 2018).

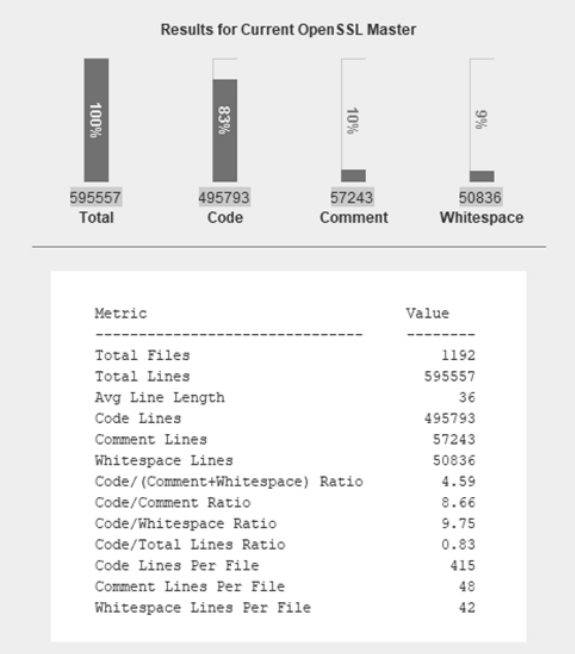

Figure 4. Software Metrics for OpenSSL 1.1.1-pre7-dev

From the initial Heartbleed version of OpenSSL to the current version, the total lines of code increased from 272,532 to 495,793, an increase of around 82%. The code/comment ratio increased from 3.0 to 8.66 indicating far smaller comment density in the current version. The lines of code per file increased from 252 to 415, around a 65% increase, even though the number of files also increased from 1080 to 1192. In fact, for the number of lines of code, the rate of increase increased. Between version 0.9.6 and the Heartbleed version 1.0.1 (11.5 years between versons), the increase in lines of code/year was 13,289. Between versions 1.0.1 (Heartbleed) and the current version 1.1.1-pre7-dev (5.5 years between versions) the increase in lines of code/year was 40,593, over three times the previous rate. This acceleration in lines of code is in spite of the fact that there was a major effort to refactor the OpenSSL code in 2015 after Heartbleed. In the years following Heartbleed, funding poured in to help the OpenSSL project. Several new full time programmers were added to the project. Without a doubt, this contributed to the acceration in coding and made feature creep worse. In the changelog between version 1.0.2h and the subsequent version 1.1.0 (released in August, 2016), there were 170 changes to OpenSSL, while the number of changes between version 1.0.0h and the subsequent Heartbleed version 1.0.1 (released in March 2012) was only 55 (OpenSSL.org, May 2018). Clearly on the former changes, there were more people on the ODG who needed things to do, hence the acceleration in added code.

After the Heartbleed flaw was made public, there were several prominent forks of the ODG's GIT repository. One of the prominent forks was named LibreSSL, created by OpenBSD founder Theo de Raadt. Several of the major Linux distributions switched their support from OpenSSL to LibreSSL (LibreSSL, 2018). Two of LibreSSL's goals specified on their Web page are:

- Modernize the OpenSSL codebase to make it easier to audit, understand and repair.

- Apply best-practice development processes.

Software metrics for LibreSSL's current repository on GitHub (LibreSSL 2.7.3) are shown in Figure 5. In the first week, the LibreSSL development group removed 90,000 lines of C code and 150,000 lines of content (Selzer, 2014).

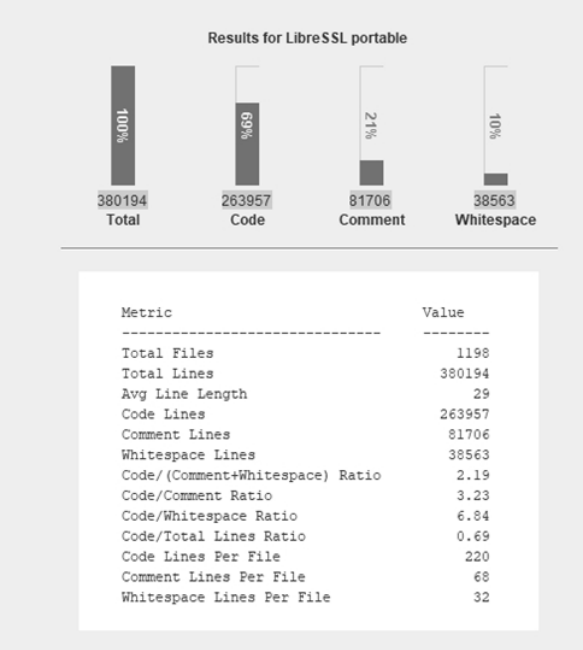

Figure 5. Software Metrics for LibreSSL 2.7.3

THE HEARTBLEED FLAW IN CODE

The Heartbleed flaw in the OpenSSL software was, for the most part, a result of the code shown in Figure 6 in a file named d1_both.c (NLUUG, 2018).

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

Figure 6. The Flawed Heartbeat Code

The statement "buffer = OPENSSL_malloc(1 + 2 + payload + padding);" creates memory from the heap for a one byte and two byte field along with space for the return string and padding (16 bytes) that represents the response record. The variable "bp" initially represents a pointer to the start of the Heartbeat's response message that is to be sent back as the echo to the Heartbeat requestor. The line "*bp++ = TLS1_HB_RESPONSE;" copies the one byte Heartbeat message type into the Heartbeat message type field. The line "s2n (payload, bp)" copies the payload length ("payload") into the payload length field pointed to by bp and increments bp to point to the next field. The statement "memcpy(bp, pl, payload);" copies "payload" bytes from requestor's string (pl) to the Heartbeat response record's echo string (now pointed to by bp). The flaw occurs in that the string pointed to by pl is never properly validated to match the payload length. Both of those values are supplied by the requestor. Instead of using a null terminated array of characters as the string to be echoed, the writer chose to use an array of characters with a separate variable determining the length. In that way, the return record's length could easily be determined without having to "count" the characters

in the string to be echoed. However, the writer forgot to validate the value for the length with the actual number of character in the string, essentially leaving out one line of code (i.e. with this method you have to count the number of character to validate anyway!). Not validating user input is a mistake done all the time by students in intro programming classes, except the consequences in this case were far more severe. The requestor can state the payload length as 64K-1 (the variable "payload" is an unsigned two byte integer) and the actual string to be echoed as a one-byte string. The Heartbeat code would then allocate 64K-1 bytes dynamically from the heap, copy the requestor's one-byte string into the replying string and return an echoed string of length 64K-1 bytes. The echoed string will contain the values of those 64K-2 memory cells that were never sanitized or copied over, i.e. whatever was placed before in those memory cells. As the software testers at Codenomicon showed, those cells can contain, among other things, the private keys used in OpenSSL (Markowitz, 2014). The patched version added the lines of code shown in Figure 7 to validate the payload length:

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->relent)
return 0;
/* silently discard */

hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
return 0;
/* silently discard per RFC 6520 sec. 4 */
pl = p;
```

Figure 7. The Patch for the Heartbleed Code

which validates the actual length of the requestor' s payload length to make sure it is not zero and that it matches the length of the string to be echoed (1 byte for the message type, two bytes for the length, "payload" bytes for the echoed string and 16 bytes of padding).

CODING ISSUES

Within the 12 lines of code in Figures 6 and 7 (not counting comments) are: nine non-descriptive identifiers ("bp", "pl", "buffer", "s", "s3" "hbtype", "rrec", "relent", and "p"), three non-descriptive C macros ("s2n", "n2s" and OPENSSL_malloc), one of which has several levels of definition (OPENSSL_malloc), two instances of unsafe pointer arithmetic (*bp++,*p++), and an unsafe function to copy memory (memcpy). Even the file name is obscure (d1_both.c). All of these illustrate a lack of coding standards being applied for OpenSSL development. Most of the remaining 272,527 lines of OpenSSL code in the Heartbleed version (and all other versions) were similar in style. The ODG's policies on coding style (OpenSSL.org, April 12, 2018) run contrary to many current day best practices for software development. Regarding the naming of variables, they state, "C is a Spartan language, and so should your naming be. Do not use longnames like ThisVariableIsATemporaryCounter. Use a name like tmp, which is much easier to write, and not more difficult to understand". In the chapter on commenting they state, "Comments are good, but there is also a danger of over-commenting. NEVER try to explain HOW your code works in a comment. It is much better to write the code so that it is obvious, and it's a waste of time to explain badly written code". Most of the OpenSSL code that was reviewed for this paper violated the latter practice as OpenSSL code clearly was not written for readibility. Readibility was almost always sacrificed for the sake of terse code and fewer clock cycles, which reflects a typical C programmer's mentality. Readibility and maintainability are going to become more of an issue as the ODG changes. The previous ODG lead programmer, who was by far the most knowledgeable team member and largest contributer and who had been with the project since its inception, retired from the team in October, 2017 (Caswell, 2017).

DESIGN ISSUES

OpenSSL is designed to implement the TLS protocol, which is under the auspices of the IETF. RFC 6520, the TLS/DTLS Heartbeat Extension, specifies the requirements for the Heartbeat extention to TLS. According to RFC 6520, the reason Heartbeat was being designed for TLS is stated as "TLS is based on reliable protocols, but there is not necessarily a feature available to keep the connection alive without continuous data transfer" (IETF, 1999). While this statement is true, there has been considerable debate in the past on whether or not responsibility for maintaining the status of a connection should be the responsibility of the application. If application developers wish to close an idle connection, the application can be written to do so. Many programmers religiously believe that checking for idle peers is a function of the application and not one for the transport layer. This is most likely the reason why the original TCP standard, RFC 793 or UDP standard, RFC 768, did not feature a "heartbeat".
In any case, the ODG's implementation of a risky protocol (Heartbeat) that echoes input coming from a remote peer using the same shared process memory space that is storing the private key for TLS/DTLS connections is poor security design. The risks involved in of comprimizing TLS data are much greater than compromizing DTLS data. TLS data includes banking transactions, purchases, and many other highly secured transactions. DTLS is primarily for less risky services, such as secure streaming (Interestingly enough, Netflix, who is the number one Internet bandwidth hog, is currently using TLS and not DTLS to do its streaming). DTLS and TLS should be run as separate processes, each running in their own secure space with their own set of private keys. Compartmentalization is a key component of secure systems, one that was not taken into account by the OpenSSL designers. The much riskier nature of a private key compromize for TLS data versus DTLS data should have pointed to a compartmentalized design. This also should have been a security consideration by the IETF in the DTLS RFC 4347. As such, Heartbeat, which was primarily designed to check for a non-responsive peer in a (typically) streaming DTLS connection, compromized the data for both TLS and DTLS over the lifetime of the private key.

During the time that the Heartbleed flaw was a zero-day vulnerability (over two years), it is believed that most compromized servers were not applying perfect forward secrecy for TCP transactions. Perfect forward secrecy in OpenSSL uses the private key only for authentication, not for the symmetric key exchange. The symmetric key exchange is typically done using the Diffie-Hellman Ephemeral (DHE) or Ellyptical-Curve Diffie-Hellman Ephemeral (ECDHE) key exchanges. If DHE/ECDHE is used to exchange keys and if the private key is subsequently comprimized, it is very difficult for a man-in-the-middle capturing traffic between two peers to determine the symmetric keys that were exchanged. Even if most servers implemented perfect forward secrecy, most SSL implementations are also courteous, in that the server follows the clients preferences. The server will usually select the first client preference that it (the server) supports. Since most clients, at the time of Heartbleed, tended to put the non-DHE/ECDHE cipher suites first, DHE/ECDHE cipher suites were most likely not chosen and so it is believed that most servers, even though they implemented perfect forward secrecy, used public/private keypairs to exchange symmetric keys. Therefore, if the private key was compomized, so were the all of the one-time symmetric keys that were exchanged using that private key along with the data they encrypted.

The second design issue in OpenSSL that contributed to Heartbleed was that OpenSSL ran Heartbeat by default. Servers and other applications that upgraded to OpenSSL 1.0.1 automatically had Heartbeat running on their systems for both TLS and DTLS connections. This again was a design failure in RFC 6520 by the IETF as there were no security considerations in RFC 6520 regarding the fact that Heartbeat should not run by default. The fact that there was no security considerations concerning the extentions not running by default, along with the lack of compartmentalized design (separating TLS and DTLS) is an indication that among the IETF TLS working group there was little or no security risk analysis done for Heartbeat.

TESTING ISSUES

From the OpenSSL.org "FIPS-140" Web page (OpenSSL.org, April 10, 2018): "OpenSSL itself is not validated. Instead a special carefully defined software component called the OpenSSL FIPS Object Module has been created. This Module was designed for compatibility with OpenSSL so that products using the OpenSSL API can be converted to use validated cryptography with minimal effort." At the time that Heartbleed was zero-day, the OpenSSL FIPS Object Module (FIPSOM) was FIPS 140-2 validated (OpenSSL.org, April 25, 2018). FIPS 140-2 is a computer security certification awarded by the Nation Institute of Standards and Technology (NIST) Cryptographic Module Validation Program. "To be FIPS 140-2 certified or validated, the software (and hardware) must be independently validated by one of 13 NIST specified laboratories. The process takes weeks. Sometimes the software fails and must be fixed and then the testing process repeated. This takes time and money." (Vamosi, 2014). The validation includes an exhaustive set of tests to make sure the associated algorithms are cryptographically secure. However, the non-FIPSOM OpenSSL code was never validated as secure. This included the Heartbleed code. Many companies that installed FIPSOM into OpenSSL have partially validated OpenSSL code labeled as "compliant". FIPS 140-2 validation of FIPSOM was likely one of the

main reasons that Heartbleed was zero-day for such a long period of time. Open Source software depends on the Open Source Community to find flaws. The beta versions of OpenSSL 1.0.1 (with the Heartbleed flaw) were public for 2 months before the initial release of the non-beta version. During that time, nobody in the Open Source community discovered the Heartbleed flaw. It is doubtful that anyone even tried. Those companies that had the resources to do testing (Google, Facebook, Amazon, and others) had a false sense of security believing that since they installed FIPSOM, their OpenSSL code was secure because it was "FIP 140-2 compliant". Since the Heartbleed version of OpenSSL with FIPSOM was FIPS 140-2 compliant, what could go wrong? As was spectacularly highlighted by Heartbleed, rigourously testing only a portion of code is not sufficient. As is seen time and time again (Ariane-5, Therac-25, Patriot Missile) when it comes to computer programs, one small crack in the dam can cause the entire dam to collapse.

The testing standards used by the ODG are not well publically documented. There is a Web page in their OpenSSK wiki that discusses unit testing for OpenSSL (OpenSSL.org, April 20th, 2018). However, given that the OpenSSL changelog contained 55 changes between the initial Heartbleed OpenSSL version 1.0.1 and the previous OpenSSL version 1.0.0h (OpenSSL.org, May 2018) with only one full-time programmer working on the project, it is doubtful there was much security testing on the non-FIPSOM code. It is possible that automated testing tools were used; however, it is also doubtful that automated testing tools would find a flaw such as Heartbleed. Concerning static analysis testing tools (i.e. those that do not execute the code), according to Carvalho M. et al. (2014), "OpenSSL's complex organization exceeded the ability of all of these tools to find the vulnerability". Dynamic analysis tools (i.e. those that would execute the code to determine flaws) would have been equally ineffective. According to Wheeler D. (2014), "Widely used dynamic and hybrid analysis approaches aren't designed to find vulnerabilities like Heartbleed. Most fuzz-testing approaches look for a crash, which may be caused by a buffer overwrite but typically not by a buffer overread." Wheeler also stated "Human review of every untrusted input field, to ensure that each one was validated, would have caught Heartbleed." This would have required a great deal of man-hours to perform and was clearly down the list of things to do for the OpenSSL development team.

CONCLUSIONS

It would be easy to say that the Heartbleed flaw was the result of a small programming error that went unnoticed. However, after looking closer at the circumstances, it is a perfect illustration of why best practices in software development and project management should be followed. Feature creep is an inevitable result of a project that has been continuing for over 20 years. Programming teams that work on the same project for long periods of time feel compelled to "improve" software through the addition of new features without proper consideration of the effects. These include: increased complexity making it more difficult to make future changes, the security risks associated with making those changes, and the increased cost of maintaining the code. With critical software, first and foremost, any proposed changes need a thorough risk analysis to determine if those changes are warranted. The changes need to be designed taking those risks into account and the code needs to be written to take into account readability and maintainability. Finally, the software needs to be properly tested using secure testing standards that follow best practices before it is made generally available. Perhaps if in the future more Open Source developers and the Open Source Community follow these suggestions, there will be fewer future "Heartbleeds".

REFERENCES

Carvalho, M., DeMott, J., Ford, R., & Wheeler, D. (2014). Heartbleed 101, *IEEE Security & Privacy, 12*(4), 63 – 67.

Caswell, M. (2017). OpenSSL Blog., retrieved at https://www.openssl.org/blog/blog/2017/10/24/steve-henson/ on 4/20/2018.

Durumeric Z., Li F., Kasten J., Amann J., Beekman J., Payer M., Weaver N., Adrian D., Paxson V., Bailey M., J. & Halderman A. (2014). The Matter of Heartbleed, November 2014 IMC '14: Proceedings of the 2014 Conference on Internet Measurement.

GitHub (April 14, 2018). LibreSSL, retrieved at https://github.com/libressl-portable/openbsd on 4/14/2018.

GitHub (April 22, 2018). OpenSSL GitHub, retrieved at https://github.com/openssl/openssl on 4/22/2018.

IETF (February, 2006). Internet Engineering Task Force, retrieved at https://www.ipv6.com/organizations/internet-engineering-task-force-ietf on 4/12/2018.

IETF (1999). RFC 2246 The TLS Protocol Version 1.0, January 1999, retrieved at https://www.ietf.org/rfc/rfc2246.txt on 4/12/2018

IETF (March, 2006). RFC 4347 Datagram Transport Layer Security, retrieved at https://www.ietf.org/rfc/rfc4347.txt 0n 3/5/2018.

IETF (2012). RFC 6520: TLS and DTLS Heartbeat Extention, February 2012, retrieved at https://www.ietf.org/rfc/rfc6520.txt on 3/5/2018.

IETF (2018). Transport Layer Security Working Group Charter, retrieved at https://datatracker.ietf.org/wg/tls/charter/ on 4/12/2018.

Jauregui, P. (2017). Exploiting Mobile Banking with HeartBleed Vulnerability, retrieved at https://p16.praetorian.com/blog/exploiting-mobile-banking-with-heartbleed-vulnerability on 4/15/2018.

Kyatam, S., Alhayajneh, A., & Hayajneh, T. (2017). Heartbleed Attacks Implementation and Vulnerability. 2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 1 – 6.

Khandelwal, S. (2017). Over 199,500 Websites Are Still Vulnerable to Heartbleed OpenSSL Bug, January 2017, retrieved at https://thehackernews.com/2017/01/heartbleed-openssl-vulnerability.html on 4/12/2018.

LibreSSL (April 2018). LibreSSL Home Page, April 2018, retrieved at http://www.libressl.org/ on 4/10/2018.

Markowitz, E. (2014). Behind the Scenes: The Crazy 72 Hours Leading Up to the Heartbleed Discovery, April 2014, retrieved at http://www.vocativ.com/tech/hacking/behind-scenes-crazy-72-hours-leading-heartbleed-discovery/ on 4/4/2018.

NetCraft, (2014). Half a million widely trusted websites vulnerable to Heartbleed bug, April 8th, 2014, retrieved at https://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html on 4/12/2018.

NLUUG (2018). OpenSSL Source Distribution Area, retrieved at http://ftp.nluug.nl/security/openssl/ on 2/15/2018.

OpenSSL.org (April 12th, 2018). Coding Style, retrieved at https://www.openssl.org/policies/codingstyle.html on 4/12/2018.

OpenSSL.org (April 10th, 2018). FIPS-140, retrieved at https://www.openssl.org/docs/fips.html on 4/10/2018.

OpenSSL.org (April 25th, 2018) FIPS-140 User Guide, retrieved at https://www.openssl.org/docs/fips/UserGuide-1.2.pdf_on 4/25/2018

OpenSSL.org (March 2012). OpenSSL Changelog, retrieved at https://www.openssl.org/news/changelog.html#x32 on 2/15/2018.

OpenSSL.org (April 20th 2018) Unit Testing, retrieved at https://wiki.openssl.org/index.php/Unit_Testing on 4/20/2018.

OpenSSL Validation Services (April). OpenSSL FIPS Object Module SE version 2.0.16, April, 2017. Retrieved at https://www.openssl.org/docs/fips/SecurityPolicy-2.0.13.pdf on 4/20/2018.

Seltzer, L. (2014). OpenBSD forks, prunes, fixes OpenSSL, April 2014, retrieved at https://www.zdnet.com/article/openbsd-forks-prunes-fixes-openssl/ on 4/12/2018.

Vamosi, R. (2014). FIPS Validated vs FIPS Compliant, What's The Difference?, July 2014, retrieved at https://www.mocana.com/blog/fips-validated-vs-fips-compliant-whats-the-difference on 4/12/2018.

Wheeler, D. (2014). Preventing Heartbleed, *Computer, 47*(8), 80 – 83.

Wikipedia (February, 2018). IETF, retrieved at https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force on 2/15/2018.

Wikipedia (March, 2018). OpenSSL, retrieved at https://en.wikipedia.org/wiki/OpenSSL on March 21st, 2018.

Yadron, D. (2014). Internet Security Relies on Very Few, Wall Street Journal, April 2014, retrieved at https://web.archive.org/web/20140426085925/https://online.wsj.com/news/articles/SB10001424052702303873604579495362672447986 on 3/15/2018.

Zhang, L., Choffnes, D., Dumitraş, T., Levin, D., Mislove, A., Schulman, A., Wilson, C. (2018). Analysis of SSL Certificate Reissues and Revocations in the Wake of Heartbleed Communications of the ACM, March, 61(3), 109-116.