SECURE SOFTWARE DESIGN – FINAL PAPER

CSOL 560.FINAL PAPER. CHAD NELLEY

Abstract

An examination of OpenSSL vulnerabilities, testing methodologies and a look at LibreSSL as an alternative to OpenSSL.

Chad Nelley

Nelley.chad@gmail.com; chad@nelleyconsulting.com

CHAD NELLEY
FINAL PAPER

CSOL560 – Secure Software Design

OVERVIEW & INTRODUCTION

In this paper I will summarize 4 OpenSSL vulnerabilities, examine the merits of LibreSSL as a potential alternative to OpenSSL and examine a proper testing methodology to be applied to Cybersecurity software development best practices.

The Objective – Analysis of OpenSSL and a review of Secure Testing Modalities

Over the years much has been documented and explored with regards to the inherent flaws of the OpenSSL platform. This paper will not be revolutionary in its concepts, hypotheses or framing of the current challenges associated with the OpenSSL framework but rather will seek to briefly explain 4 vulnerabilities and how current testing methodologies applied to the go-forward development of OpenSSL or any other platform, for that matter, can help to reduce risk and promote a more secure approach to systems and architectures that are handling sensitive data and information.

Deliverables for this Paper as defined in the Assignment:

- 4 Specific Vulnerabilities that have impacted OpenSSL
- A review of Testing methodologies & products
- A look at LibreSSL as an alternative application to OpenSSL

DELIVERABLES:

Four Specific Vulnerabilities in OpenSSL

For purposes of this course and final paper, I chose 4 specific vulnerabilities from the OpenSSL library of known bugs. Two mainstream, well documented, issues and two lesser known and lesser documented know issues.

Vulnerability 1: DROWN

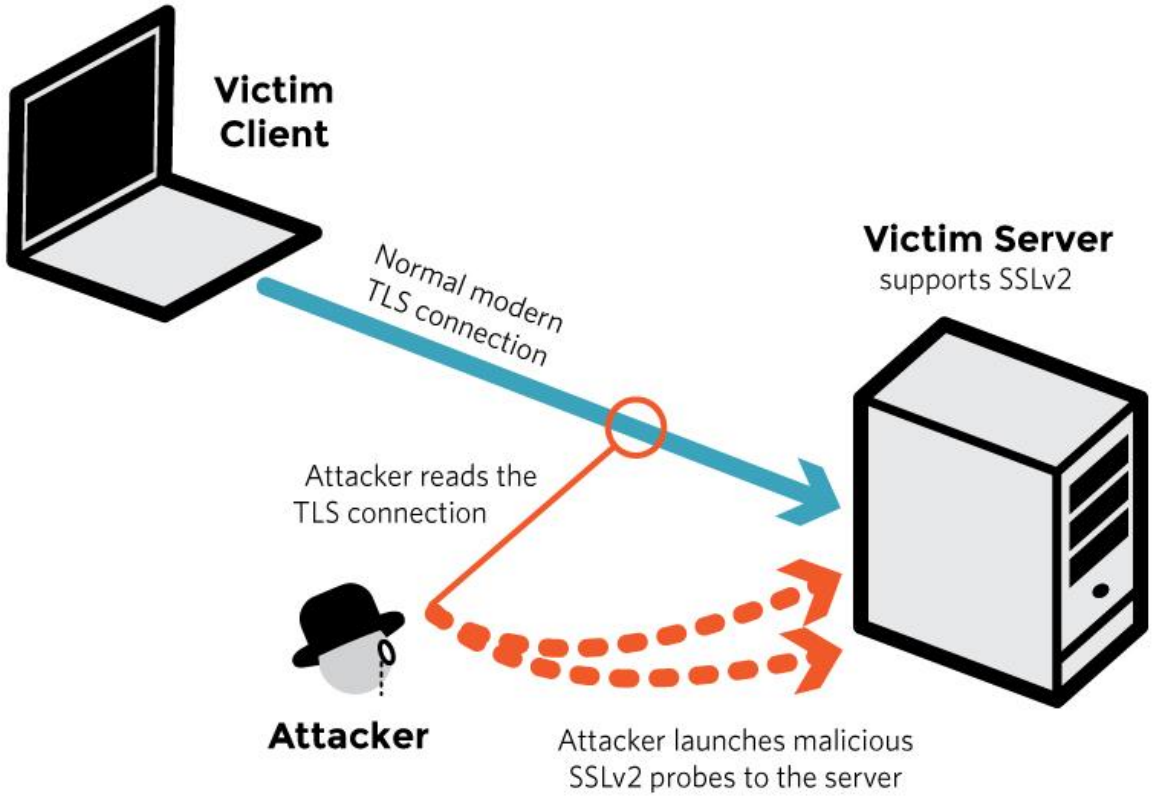
DROWN is a serious vulnerability that affects HTTPS and other services that rely on SSL and TLS, some of the essential cryptographic protocols for Internet security. These protocols allow everyone on the Internet to browse the web, use email, shop online, and send instant messages without third-parties being able to read the communication.

DROWN allows attackers to break the encryption and read or steal sensitive communications, including passwords, credit card numbers, trade secrets, or financial data. Our measurements indicate 33% of all HTTPS servers are vulnerable to the attack.

DROWN enables attackers to intervene in any communication between users and the server. This typically includes, but is not limited to, usernames and passwords, credit card numbers, emails, instant messages, and sensitive documents. Under some common scenarios, an attacker can also impersonate a secure website and intercept or change the content the user sees.

Websites, mail servers, and other TLS-dependent services are at risk for the DROWN attack. It is thought that even after the fix was announced and information for patching was made available, that nearly 15% of the top one million domains are still vulnerable to a DROWN attack.

Below is an illustration on how a DROWN attack works:



A server is vulnerable to DROWN if:

- It allows SSLv2 connections. This is surprisingly common, due to misconfiguration and inappropriate default settings. Our measurements show that 17% of HTTPS servers still allow SSLv2 connections.

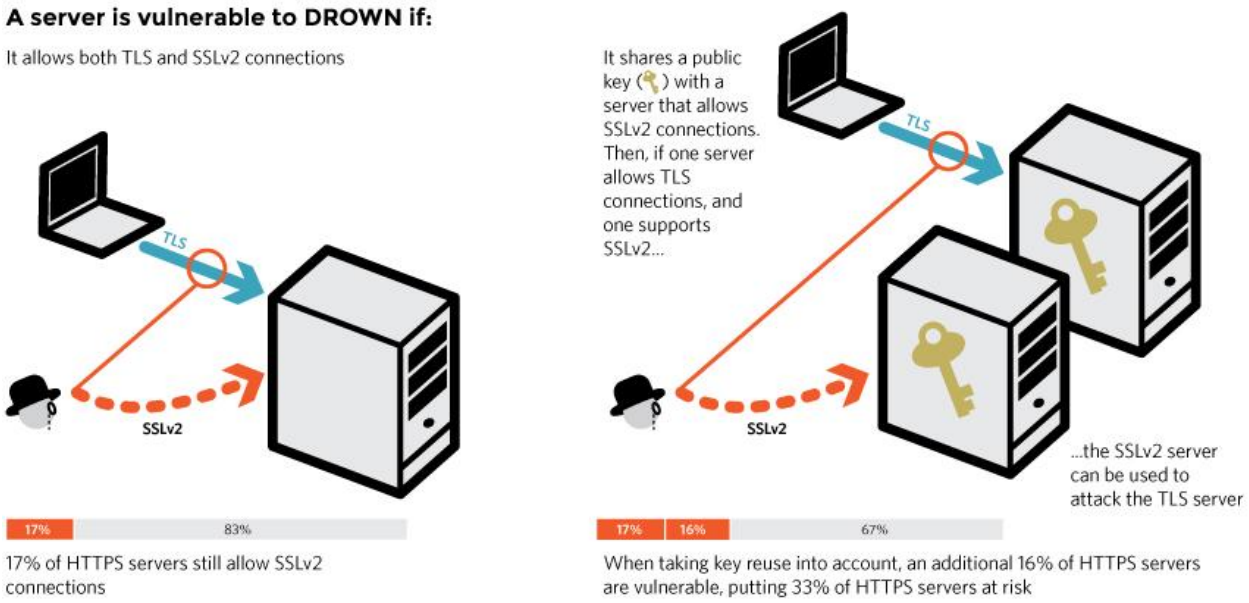
or:

- Its private key is used on *any other server* that allows SSLv2 connections, even for another protocol. Many companies reuse the same certificate and key on their web and email servers, for instance. In this case, if the email server supports SSLv2 and the web server does not, an attacker can take advantage of the email server to break TLS connections to the web server. When taking key reuse into account, *an additional 16%* of HTTPS servers are vulnerable, putting 33% of HTTPS servers at risk.

So, how is OpenSSL affected by DROWN?

OpenSSL: OpenSSL is a cryptographic library used in many server products. For users of OpenSSL, the easiest and recommended solution is to upgrade to a recent OpenSSL version. OpenSSL 1.0.2 users should upgrade to 1.0.2g. OpenSSL 1.0.1 users should upgrade to 1.0.1s. Users of older OpenSSL versions should upgrade to either one of these versions.

The below diagram illustrates how a server becomes vulnerable to a DROWN attack:



DROWN stands for **D**ecrypting **R**SA with **O**bsolete and **W**eakened e**N**cryption.

DROWN is made worse by two additional OpenSSL implementation vulnerabilities. CVE-2015-3197, which affected OpenSSL versions prior to 1.0.2f and 1.0.1r, allows a DROWN attacker to connect to the server with disabled SSLv2 ciphersuites, provided that support for SSLv2 itself is enabled. CVE-2016-0703, which affected OpenSSL versions prior to 1.0.2a, 1.0.1m, 1.0.0r, and 0.9.8zf, greatly reduces the time and cost of carrying out the DROWN attack. I will attempt to cover these two vulnerabilities of OpenSSL next, but caveat as to say there is far less information available on each.

Vulnerability 2: SSLv2 doesn't block disabled ciphers (CVE-2015-3197)

Although the severity rating on this issue is touted as low, it is a key precursor component issue for the above DROWN type of attack. This vulnerability can best be described as:

A malicious client can negotiate SSLv2 ciphers that have been disabled on the server and complete SSLv2 handshakes even if all SSLv2 ciphers have been disabled, provided that the SSLv2 protocol was not also disabled via SSL_OP_NO_SSLv2.

This issue affects OpenSSL versions 1.0.2 and 1.0.1.

OpenSSL 1.0.2 users should upgrade to 1.0.2f
OpenSSL 1.0.1 users should upgrade to 1.0.1r

This issue was reported to OpenSSL on 26th December 2015 by Nimrod Aviram and Sebastian Schinzel. The fix was developed by Nimrod Aviram with further development by Viktor Dukhovni of the OpenSSL development team.

An update on DHE man-in-the-middle protection (Logjam). A previously published vulnerability in the TLS protocol allows a man-in-the-middle attacker to downgrade vulnerable TLS connections using ephemeral Diffie-Hellman key exchange to 512-bit export-grade cryptography. This vulnerability is known as Logjam CVE-2015-4000). OpenSSL added Logjam mitigation for TLS clients by rejecting handshakes with DH parameters shorter than 768 bits in releases 1.0.2b and 1.0.1n.

This limit has been increased to 1024 bits in this release, to offer stronger cryptographic assurance for all TLS connections using ephemeral Diffie-Hellman key exchange.

OpenSSL 1.0.2 users should upgrade to 1.0.2f
OpenSSL 1.0.1 users should upgrade to 1.0.1r

The fix was developed by Kurt Roeckx of the OpenSSL development team.

Vulnerability 3: Divide-and-conquer session key recovery in SSLv2 (CVE-2016-0703)

The divide-and-conquer session key recovery in SSLv2 also referred to as CVE-2016-0703 has a high severity rating and is another precursor exploit that will help enable a DROWN attack. Details of this vulnerability are provided below:

This issue only affected versions of OpenSSL prior to March 19th 2015 at which time the code was refactored to address vulnerability CVE-2015-0293.

s2_srvr.c did not enforce that clear-key-length is 0 for non-export ciphers. If clear-key bytes are present for these ciphers, they *displace* encrypted-key bytes. This leads to an efficient divide-and-conquer key recovery attack: if an eavesdropper has intercepted an SSLv2 handshake, they can use the server as an oracle to determine the SSLv2 master-key, using only 16 connections to the server and negligible computation.

More importantly, this leads to a more efficient version of DROWN that is effective against non-export ciphersuites, and requires no significant computation.

This issue affected OpenSSL versions 1.0.2, 1.0.1l, 1.0.0q, 0.9.8ze and all earlier versions. It was fixed in OpenSSL 1.0.2a, 1.0.1m, 1.0.0r and 0.9.8zf (released March 19th 2015).

This issue was reported to OpenSSL on February 10th 2016 by David Adrian and J. Alex Halderman of the University of Michigan. The underlying defect had by then already been fixed by Emilia Käsper of OpenSSL on March 4th 2015. The fix for this issue can be identified by commits ae50d827 (1.0.2a), cd56a08d (1.0.1m), 1a08063 (1.0.0r) and 65c588c (0.9.8zf).

Vulnerability 4: Heartbleed

While there is a tremendous amount of information and data available on Heartbleed, for purposes of this paper, I have narrowed the explanation down to a streamlined version of facts with a couple of well known graphics as ride along content for illustration and understanding.

Officially referred to as CVE-2014-0160, the Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

How Heartbleed works:

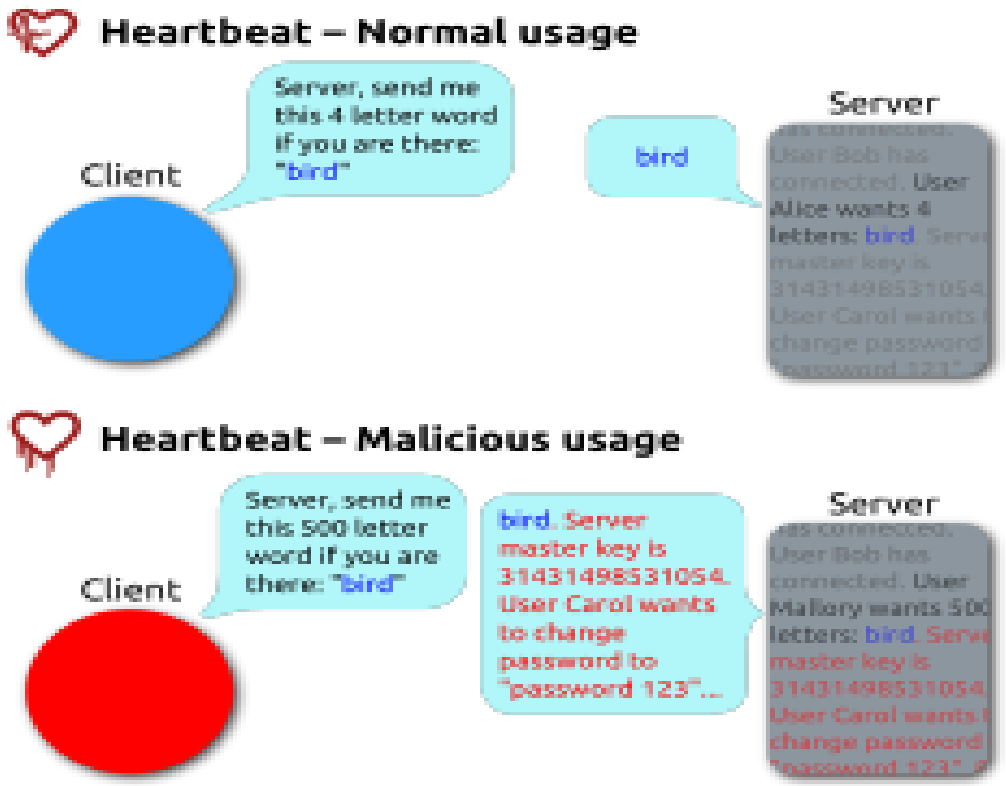
The RFC 6520 Heartbeat Extension tests TLS/DTLS secure communication links by allowing a computer at one end of a connection to send a *Heartbeat Request* message, consisting of a payload, typically a text string, along with the payload's length as a 16-bit integer. The receiving computer then must send exactly the same payload back to the sender.

The affected versions of OpenSSL allocate a memory buffer for the message to be returned based on the length field in the requesting message, without regard to the actual size of that message's payload. Because of this failure to do proper bounds checking, the message returned consists of the payload, possibly followed by whatever else happened to be in the allocated memory buffer.

Heartbleed is therefore exploited by sending a malformed heartbeat request with a small payload and large length field to the vulnerable party (usually a server) in order to elicit the victim's response, permitting attackers to read up to 64 kilobytes of the victim's memory that was likely to have been used previously by OpenSSL. Where a Heartbeat Request might ask a party to "send back the four-letter word 'bird'", resulting in a response of "bird", a "Heartbleed

Request" (a malicious heartbeat request) of "send back the 500-letter word 'bird'" would cause the victim to return "bird" followed by whatever 496 characters the victim happened to have in active memory. Attackers in this way could receive sensitive data, compromising the confidentiality of the victim's communications. Although an attacker has some control over the disclosed memory block's size, it has no control over its location, and therefore cannot choose what content is revealed

A simple diagram below illustrates the Heartbleed vulnerability:



Testing Methodologies & Product Options

Through all of the vulnerabilities listed and detailed above, it has been widely debated that more robust testing and scrutiny of the code base on the front end, may very well have prevented a number of these issues – or at least exposed them earlier in the pre-release phase. In this course we spent a significant amount of time reviewing and articulating test methodologies. The trick, as with most things, is when and where to use such testing methods and modalities and how to apply them at the lowest overhead in term of both finance and personnel resources.

As we look at testing mechanisms and tools, for purposes of this paper, I am going to reflect on some of the articles and reading materials to attempt to add context and specifics on a few testing methods and products:

Bootstrapping Trust in Commodity Computers

One example of an attestation model/approach is the Trusted boot vs. Secure boot via the use of TPM. In short some of the benefits of this model are outlined below:

The two scenarios can be summarized as such:

Trusted Boot Scenario

- Measure system during boot for remote verification
- Operating system is booted based on a measured system (*integrity verifiable*)
- Enable verification of boot:
  - Base layer is immutable
  - The integrity of the base layer is measured
  - Transition to higher layer only occurs after valid measurement
- Remote party can verify measurements to determine integrity

Key Takeaways

- Computer is not stopped if secure boot guarantee is violated
- Provable to remote systems
- Requires root of trust

Secure Boot Scenario

- Ensure only a secure system is booted
- Operating system that is bootstrapped is based on a untampered foundation (*integrity guarantee*)
- Initially not a problem, but nowadays field upgradable FLASH memory is used
- Integrity of a layer can only be guaranteed if-and-only-if:
  - Base layer is immutable
  - The integrity of the base layer is verified
  - Transition to higher layer only occurs after valid verification

Key Takeaways

- Computer is stopped if secure boot guarantee is violated
- Not provable to remote systems

Access controls of a TPM solution such as: Storage Access Controls, Protected and TPM Sealed storage and remote access elements like TPM based attestation. This process can be defined as: During the protocol, the verifier supplies the attestor with a nonce to ensure freshness (i.e., to prevent replay of old attestations). The attestor then asks the TPM to generate a *Quote*. The Quote is a digital signature covering the verifier's nonce and the current measurement aggregates stored in the TPM's Platform Configuration Registers (PCRs). The

attestor then sends both the quote and an accumulated measurement list to the verifier. Some things practitioners should be aware of are the concerns of the platform state, Privacy, the roots of the Trust model, validation and day-to-day application and future implications of the model.

Using TrustVisor to achieve efficient TCB Reduction and Attestation

The implementation of TrustVisor on an AMD platform in an effort to achieve robust attestation. TrustVisor has three basic operating modes. *Host mode* refers to execution of TrustVisor code at the system's highest privilege level. TrustVisor in turn supports two guest modes: legacy and secure. In *legacy guest mode*, a commodity x86 OS and its applications can execute without requiring any awareness of the presence of TrustVisor. The legacy OS manages all peripheral devices on the system (network, disk, display, USB, etc.), with the TPM as the only device shared between TrustVisor and the untrusted legacy OS.

The fact that TrustVisor also incorporates two basic Trust computing mechanisms – 1. The sealed storage mechanism and 2. Remote Attestation mechanism. The sealed storage mechanism can best be described as a mechanism in which a particular PAL can encrypt data along with a policy such that the resulting ciphertext can only be decrypted by the PAL specified in the policy. The remote attestation mechanism is described as a mechanism by which a remote party can be convinced that a particular PAL indeed ran on a particular platform (optionally with particular inputs and producing particular outputs) protected by TrustVisor. Both of these mechanisms are enabled by an *integrity measurement* process that maintains a set of measurements (cryptographic hashes) of all code in the TCB for a PAL of interest.

The Implementation and evaluation of the TrustAdvisor hypervisor solution can greatly enhance code execution integrity. This system enforces code and execution integrity, and data secrecy and integrity for PALs. TrustVisor enables fine-grained attestations to the PAL's execution. TrustVisor supports unmodified legacy OSes and their applications, so that only new applications developed with enhanced security properties require any awareness of TrustVisor. The significant security benefits of TrustVisor outweigh the performance costs, which will mostly vanish with improved hardware virtualization support.

Design and Implementation of a TCG Based Integrity Measurement Architecture

A trust measurement architecture to a web server application to illustrate how a TPM system can detect such items as rootkits and malware with limited impact to performance. This model in which a remote system (in this case the challenger system) has to prove to another system (in this case the attesting system) that it is of sufficient integrity to use. In this model the integrity measurement architecture is based on the following measurement systems: Verification Scope, Executable Content, Structured Data and Unstructured Data.

Specific to the Design Architecture are 3 core components: They are as follows:

- The **Measurement Mechanism** on the attested system determines what parts of the run-time environment to measure, when to measure, and how to securely maintain the measurements.
- An **Integrity Challenge Mechanism** that allows authorized challengers to retrieve measurement lists of a computing platform and verify their freshness and completeness.

- An **Integrity Validation Mechanism**, validating that the measurement list is complete, non-tampered, and fresh as well as validating that all individual measurement entries of runtime components describe trustworthy code or configuration files.

Flicker as a security validation tool

The use of Flicker as a matter of ensuring a trusted security platform and secure execution of code in the CPU's most privileged mode is yet another way to validate the security posture of a platform. Flicker is an architecture for isolating sensitive code execution using a minimal TCB (Trusted Computing Base).

The Flicker implementation leverages TPM based attestation, TPM Sealed Storage concepts and TPM vs. Dynamic PCR's which reinforces reading's from the prior articles above. Late launch concepts are critical to establishing the need/relevance of Flicker as a solution. Flicker achieves its properties using the Late Launch capabilities of the base system. Instead of launching a VMM, Flicker pauses the current execution environment, executes a small snippet of code and then resumes operation of the previous environment. Flicker achieves this in isolated execution mode. After an extensive overview of how the SKINIT snippet of code works, the paper continues with the Developer's perspective on how to build and execute a PAL.

Specific application scenarios for use of Flicker such as SSH Password authentication – the primary goal in this scenario being, to prevent malicious code on the server from learning the user's password, even if the server's OS has been compromised.

Ultimately, Flicker significantly improves the security and reliability of the code it executes and suggest that the implementation and use of Flicker-based applications could become a reality, which would greatly enhance onboard security of commodity computers.

An Alternative to OpenSSL: LibreSSL

LibreSSL is an open-source implementation of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. It was forked from the OpenSSL cryptographic software library in April 2014 as a response by OpenBSD developers to the Heartbleed security vulnerability in OpenSSL, with the aim of refactoring the OpenSSL code so as to provide a more secure implementation.

LibreSSL was forked from the OpenSSL library starting with the 1.0.1g branch and will follow the security guidelines used elsewhere in the OpenBSD project.

After the Heartbleed bug in OpenSSL, the OpenBSD team audited the code afresh, and quickly realised they would need to maintain a fork themselves.The libressl.org domain was registered on 11 April 2014; the project announced the name on 22 April 2014.

In the first week of code pruning, more than 90,000 lines of C code were removed. Older or unused code has been removed, and support for some older or now-rare operating systems removed. LibreSSL was initially being developed as an intended replacement for OpenSSL in OpenBSD 5.6, and was then ported back to other platforms once a stripped-down version of the

library was stable. As of April 2014, the project was seeking a "stable commitment" of external funding.

Many folks in secure coding circles have called into question the quality of the OpenSSL codebase and many believe that LibreSSL is a much more robust and secure solution because of some of the proactive development and testing methodologies and code practices that have been employed in its development. Although it is a fork of the original OpenSSL set, numerous changes have been incorporated and the code base is believed to be stronger.

Vulnerabilities in LibreSSL as compared to OpenSSL are significantly reduced as illustrated by the tables below:

Total vulnerabilities between the release of LibreSSL and the release of OpenSSL 1.0.2:

Severity	**LibreSSL**	**OpenSSL**
Critical	0	1
High	3	6
Moderate	9	14
Low	6	21
Total	18	42

Since the release of OpenSSL 1.0.2

Severity	**LibreSSL**	**OpenSSL**		
		1.0.1	**1.0.2**	**1.1.0**
Critical	0	0	1	1
High	0	2	7	3
Medium	12	15	22	2
Low	7	10	26	15
Unclassified	2	0	0	
Total	21	27	35	

CONCLUSION

In conclusion, it is clear that robust code development and strong testing methodologies are clear paths to better code and better platforms. As Illustrated above in the charts that compare LibreSSL and OpenSSL, when proactive measures are put in place on the front end and proper testing and validation principles and techniques are layered into the process, the result is a far stronger offering. Some might contend that the adoption of LibreSSL as compared to the adoption of OpenSSL maybe one of the reasons for a discovery of less vulnerabilities, but it is clear to me from my research on the topic that alternative solutions like LibreSSL that have been developed with a security-by-design approach, are likely the longer term winners in the new era of heightened awareness of Cybersecurity.

It is clear that OpenSSL may have been rushed to market to meet a minimum demand, but obviously the core developers of LibreSSL have given much thought on the front end of the design and architecture to ensure better outcomes for industry and adoption of such methods are on the rise and at the forefront of the discussion these days – which is a good thing for all of us in the fight (The good side of course).

References

https://en.wikipedia.org/wiki/LibreSSL

https://en.wikipedia.org/wiki/Heartbleed

http://heartbleed.com/

https://drownattack.com/

CVE-2016-0800

CVE-2015-3197

CVE-2016-0703