Global Information Assurance Certification Paper

The Role of Static Analysis in Heartbleed

GIAC (GSEC) Gold Certification

Author: Jeff Sass, jsass@adobe.com
Advisor: Stephen Northcutt

Accepted: February 12, 2015

Abstract

The Heartbleed bug was one of the largest security vulnerabilities of 2014, not only because of the media attention it garnered but also because it affected over half a million web sites on the Internet. Because the bug was in OpenSSL, it affected web sites, VPN concentrators, client applications and mobile devices. This paper details what the Heartbleed bug is, how the details were disclosed, and how vendors responded to it. The role of static analysis in software quality is then discussed. How static analysis, specifically Coverity's TAINTED_SCALAR heuristic, was improved to detect this bug will also be presented. Finally, how end users can protect themselves from similar vulnerabilities will be discussed.

1. Introduction

Numbered security vulnerabilities known as Common Vulnerabilities and Exposures (CVEs), have been on the rise since the United States Computer Emergency Readiness Team (US-CERT) began tracking them in 1999. In 1999 there were 1,597 CVEs and in 2014 there were 9,526. On April 7, 2014, CVE-2014-0160 ("Vulnerability Summary for CVE-2014-0160", 2014) was disclosed. It *"affected over half a million widely trusted web servers used on the Internet"* ("Half a million widely trusted websites vulnerable to Heartbleed bug", 2014). This vulnerability is commonly referred to as the Heartbleed bug ("The Heartbleed Bug", 2014). In order to understand why the Heartbleed bug had such an impact on the Internet, we must first look at what Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols are and how they work.

TLS is used to secure communications between two endpoints. TLS is the successor to SSL although many references still use the SSL/TLS terminology. For this discussion of Heartbleed, the term TLS will be used throughout. TLS encrypts and decrypts packets of data as they flow between two endpoints. It does this so that eavesdroppers cannot spy on the data while it is in transit. There are many computing applications that require this type of message confidentiality. The most common example is when a web browser connects to a web site using https:// instead of http://. The reasonable assumption is, for that session, there are no eavesdroppers who are decrypting the traffic and stealing private information (i.e. credit cards, passwords or digital certificates). This assumption was proven to be incorrect in applications that used certain versions of OpenSSL containing the Heartbleed bug.

1.1. Incorporating Open Source Libraries

When designing secure applications one of the first questions that must be answered is: should developers write their own cryptography library or should they use an open source one? In the earlier days of the Internet, there wasn't a choice. Developers had to write their own. Modern software development however, should follow the advice from the SANS Institute top 25 most dangerous software errors. CWE-327 states to not use a broken or risky cryptographic algorithm. By only reading the title, a developer might think that the given advice means that developers should write their own algorithm.

Jeff Sass, jsass@adobe.com

On the contrary, the advice from MITRE and SANS ("CWE/SANS TOP 25 Most Dangerous Software Errors", 2010) states:

"Cryptography is just plain hard. If brilliant mathematicians and computer scientists worldwide can't get it right (and they're always breaking their own stuff), then neither can you."

Per this advice, when looking at cryptography libraries, developers should generally choose to use an existing algorithm and more specifically an implementation of that library that has been peer reviewed.

Other factors to consider involve whether or not the library is actively developed, has a proven track record of quality, has an acceptable licensing agreement, and whether or not the library is open source. Some cryptography libraries are commercially available and some are open source. OpenSSL is open source and has a proven track record of fixing security vulnerabilities ("OpenSSL vulnerabilities", 2014). Looking at the number of previous CVEs in OpenSSL's vulnerability list might lead a developer to choose to write their own implementation of a cryptographic library. However the opposite is generally true. Because the OpenSSL team is actively fixing vulnerabilities, and has been for over a decade, they have established a level of trust that cannot easily be gained by a newly written library. Examining their CVE list before April 2014 would have led most developers to choose to incorporate the library instead of writing their own.

Another aspect to consider is if the library is a commercial one or open source. Open source libraries provide the source code as well as the compiled binary. This gives developers the ability to inspect the source code as one way to help determine the level of quality. It is common for open source libraries to also contain unit tests to help prove the stated quality level. While open source libraries can never guarantee quality, nor can commercial libraries, being able to read the source code is additional data to help guide the decision. An experienced developer can determine within a few hours what the level of quality an open source library possesses.

1.2. Heartbeat Extension

In order to understand the Heartbleed bug, we must first look at how TLS handshakes work. When a browser connects to a web server there is a handshake that

occurs to establish the connection. Then depending on whether it is HTTP or HTTPS, the security of that connection is established. HTTP/HTTPS traffic is stateless by default. This means that once the data is sent, the connection ends. When there is more traffic to send, the connection must be reestablished and that requires that both computers go through another handshake process. Heartbeats were added to TLS as specified in RFC 6520 to *"keep the connection alive without continuous data transfer"* (R. Seggelmann, 2012). When the OpenSSL team added the heartbeat extension feature, they turned it on by default. Developers have the option to compile with the

-DOPENSSL_NO_HEARTBEATS compiler flag to disable it; however, the OpenSSL developers made the design decision to turn on the Heartbeat extension by default.

This Heartbeat extension is what keeps the connection open. Heartbeat protocol messages have four parts: *HeartbeatMessageType*, *payload_length*, *payload*, and *padding*. As long as the client continues to receive a *HeartbeatResponse* to match the *HeartbeatRequest* the connection is maintained. When the server receives the HeartbeatRequest it responds with a *HeartbeatResponse* with an *"exact copy of the payload of the received HeartbeatRequest"* ("OpenSSL Heartbeat Vulnerability", 2014). Normally the confidentiality of the payload is not compromised.

1.3. Heartbeat vulnerability

When the Heartbleed bug is exploited, the attacker will create a specially crafted *HeartbeatRequest*. This request will shrink the *payload* to a smaller value, possibly as small as 1 byte, and set the *payload_length* to something larger up to 65,535 bytes. According to page 5 of RFC 6520 (Seggelmann, 2012), the implementation of the specification should have discarded the *HeartbeatMessage*.

"If the payload_length of a received HeartbeatMessage is too large, the received HeartbeatMessage MUST be discarded silently."

Instead of discarding the *HeartbeatMessage*, the OpenSSL implementation placed the message into memory at the size specified by the specially crafted request. This is where the bug occurred.

When a software library takes input from end users and puts it into memory without checking the parameters, this input is called tainted input. The OpenSSL implementation trusted the *payload_length* parameter from the client without checking the actual size of the payload. This is a violation of CWE-807: Reliance on Untrusted Inputs in a Security Decision (Christey, "2011 CWE/SANS Top 25 Most Dangerous Software Errors").

The OpenSSL library looked in the server's memory at the address of the payload and then copied the section of memory up to the attacker's specified *payload_length.* This could be up to 65535 bytes. That memory was then returned in the *HeartbeatResponse*. The attacker now has access to whatever was stored in the web server's memory at that specific time. This could include user names, passwords, or possibly the encryption keys that were used to establish the secure connection.

There is nothing preventing the attacker from repeating this attack in an attempt to continue to steal confidential data from the server's memory. Since the exchange of data happens during the initial handshake part of the protocol, "exploitation of this bug does not leave any trace" in the webserver logs ("The Heartbleed Bug", 2014). Dr. Bagley referred to this process as "a bit like panning for gold" (Bagley, 2014).

Researchers reviewed the logs of passive Internet taps and did not find any large-scale evidence of Heartbleed exploits up to April 7. On April 8, they did discover "subsequent exploit attempts from almost 700 sources" (Durumeric, 2014). This shows that Heartbleed was extremely easy to exploit, and attackers used tools like Metaspolit's pen-testing modules to attack servers immediately after the disclosure.

2. Heartbleed discovery

Two independent security research teams discovered the Heartbleed bug. Neel Mehta of Google Security discovered Heartbleed on March 21st, 2014 while conducting a source code review of OpenSSL (Grubb, 2014). On April 2nd, 2014, a team of Finnish Codenomicon engineers named Antti Karjalainen, Riku Hietamaki, and Matti Kamunen discovered the bug while performing testing on Codenomicon's Defensics SafeGuard feature. ("The Heartbleed Story", 2014).

Jeff Sass, jsass@adobe.com

The Google team notified the OpenSSL team initially while the Codenomicon team notified the National Cyber Security Centre Finland (NCSC-FI) who then asked the CERT Coordination Centre for a CVE number. Codenomicon then registered the heartbleed.com domain, created the Heartbleed logo, and reported the bug to a member of the OpenSSL team (ironically a Google engineer named Ben Laurie). The member then forwarded the information to the entire OpenSSL team. Because two independent sources disclosed the bug within the same period of time, a patch was released later that day on April 7, 2014 instead of trying to perform a more coordinated rollout. A more detailed timeline is available from Ben Grubb of the Sydney Morning Herald (Grubb, 2014).

The Heartbleed logo

Photograph: /Codenomicon

3. Heartbleed Response

Hundreds of news reports and articles were posted on the Heartbleed bug. These quickly spread through the security community and media outlets.

3.1. Security Community Response

Dr. Johannes Ullrich of SANS posted details of Heartbleed to the Internet Storm Center (Ullrich, 2014). Jake Williams, while speaking at the SANS 2014 conference, recorded a webcast to the SANS webcast archive (Williams, 2014). He also referenced that a Heartbleed testing module for Metasploit, a pen testing tool used to test and exploit vulnerabilities, was posted to GitHub (n.d.).

The lynda.com training site also posted two training videos, “Heartbleed Tactics for Small IT Shops” (Gassner, 2014), and “Protecting Yourself from the Heartbleed Bug” (Seeley, 2014). The security community understands how to respond to security vulnerabilities and ensure the information is accurate and sent to the necessary professionals.

3.2. Media Response

When examining the mass media response, two examples stand out: the nightly comedy show, The Colbert Report, and the web comic xkcd. On the April 8 episode, Stephen Colbert starts his show by taping a laptop with duct tape and twine and placing a mousetrap on top of the box in order to "secure his data". He goes on to say:

"The Internet was supposed to be a lawless frontier where all of humanities desires and vices merged into a royally connected id held in check by a barely regulated rats-nest of technical abstractions I don't understand. How did that get out of control?"

A graphical description of the Heartbleed bug was from the xkcd web comic. It was used because of its simple explanation showing how the vulnerability works ("xkcd: Heartbleed Explanation", 2014). The comic medium was used effectively to quickly show the vulnerability. The combination of these and other media responses highlights a recent need to explaining security concepts to the masses in an accurate, approachable way.

3.3. OpenSSL Response

The OpenSSL team responded by fixing the bug and posting the details on their vulnerability page at http://openssl.org/news/vulnerabilities.html. The patch was released in version 1.0.1g and 1.0.2-beta2 of their library. Performing a diff on the tlsl_process_heartbeat() function in the t1_lib.c files from OpenSSL 1.0.1f and 1.0.1g shows how the bounds check of this memory over-read was added. 1.0.1g was released to the public on April 7, 2014 ("OpenSSL vulnerabilities", 2014).

3.4. Vendor Response

Vendors began reviewing their client and server machines to determine if they contain the vulnerable version of OpenSSL. If the vulnerable version was found the vendor issued a patch via their normal software update process. For running webservers an additional best practice was needed to "*reissue a new private key and expire all active user sessions*" (Williams, 2014). Client applications should update their applications with a non-vulnerable OpenSSL library.

3.5. End-User Response

The recommendation from "The Heartbleed Bug" (2014) was to change your password, but only after you have verified that the server you are connecting to has already applied the patch and re-issued their digital certificates.

3.5.1. Changing passwords

End-users can check https://lastpass.com/heartbleed/ or similar sites linked from security researcher Brian Krebs' blog post (Krebs, 2014), to determine if a web site is still vulnerable. If end-users changed their password before the site was patched, then the act of visiting the site and changing your password would incur a higher likelihood of putting the user's password in the web server memory. This act of changing the password could then be exploited compared with a user who didn't change their password during this initial rush of disclosure and patching. Brian Krebs gave the following advice, "It certainly can't hurt to change your password now and then again next week." (Wood, 2014).

3.5.2. Password Managers

For a modern Internet user, asking them to reset their password on all sites that were listed as vulnerable can be very time consuming. To follow Brain Krebs' advice and reset them two times could easily result in hundreds of password changes for an average Internet user. The advice of many security professionals, including L. Newman (2014) is to setup a password manager. A password manager helps you in two ways. First, they keep track of all of the passwords you use so you are not tempted to write them down in a non-secure location. Second, the generate feature will automatically assign a long, unique, and random password for each site you visit. This will ensure that the password you change to is strong. Because of the wide media coverage of Heartbleed, many users started using password managers to help organize their digital lives and make the process of changing passwords much easier if another vulnerability occurs.

4. Static Analysis

Now that the mechanics of Heartbleed have been presented, we turn to the role of static analysis in software quality. Static analysis is the process of determining code

quality without executing the program. Static analysis tools trace all of the possible branches of the code without executing it. In contrast, dynamic analysis works by analyzing executing code. Dynamic analysis techniques used to find Heartbleed are discussed further in Dr. Wheeler's paper "How to Prevent the Next Heartbleed" (Wheeler, 2014).

Janet Gregory and Lisa Crispin have written two books on agile testing (Crispin, L., & Gregory, J., 2009) and (Gregory, J., & Crispin, L., 2014). In chapter 8 of their later book, "More Agile Testing", an updated version of the agile testing quadrants is presented and shown below in Figure 1.

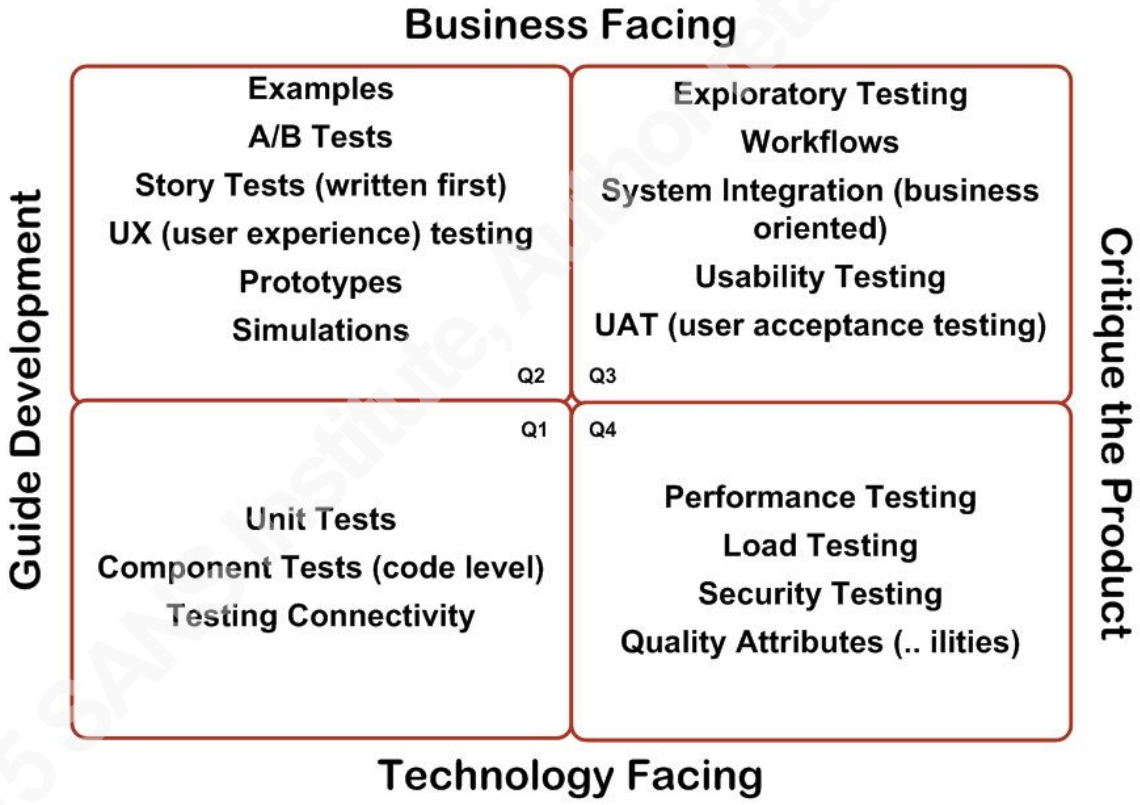

Figure 1 – Agile testing quadrants - (Gregory, J., & Crispin, L., 2014), used with permission

Security testing falls into Q4 in the bottom right and is influenced by technology-facing tests that critique the product. Security tests are in the same quadrant as performance and load tests and a category called "ilities" (e.g. reliability, interoperability, scalability). In their first book "Agile Testing", the authors state:

"However, there are many tasks that need specialized knowledge. A good example is security testing… We're talking about probing for external security flaws and knowing the types of vulnerabilities in systems that hackers exploit. That is a specialized skill set." (Crispin, L., & Gregory, J., 2009)

Software development teams have the challenge of balancing the amount of effort they spend on each of the testing quadrants. Focusing on the customer experience should guide all of these trade-off decisions on product development teams. Teams also need to evaluate which of the quadrants will give the most "bang for the buck". If there is a dedicated quality engineering team then the responsibility for testing items in the higher quadrants generally falls to them. Because security testing is in Q4 and requires specialized skills, historically this work is either outsourced to a third party vendor or not done at all. Luckily, in recent years, there has been a sharp increase in the number of security jobs at technology companies and many are now choosing to grow this specialized set of skills in-house.

Adobe uses a mix of all three types of security resources. First, ASSET (Adobe's Secure Software Engineering Team) focuses on the larger Adobe security landscape and helps coordinate security testing with third parties. Second, Adobe utilizes these third party vendors for pen-testing projects to liaison with developers writing the code. Finally, individual product teams have security champions that are responsible for ensuring the overall security profile of the entire product.

There are many security tools that developers can use, some are open source and some are commercial. Adobe uses many different tools, but the most popular commercial ones are Coverity and Checkmarx.

4.1. Coverity

Coverity, which was recently purchased by Synopsys, is a "leading provider of software quality and security testing solutions" (Coverity, 2014). It is considered more than a security testing tool. The static analysis engine not only detects security defects but also more general coding defects that are unknowingly made by developers. Adobe uses the tool for general quality purposes but for this discussion we will focus on the specifically on the static analysis security checkers. Coverity can be used with the Coverity Scan service or deployed on-premise.

Jeff Sass, jsass@adobe.com

4.1.1. Coverity Scan

Coverity offers Coverity Scan free of charge to the open source community. This service scans over 2,500 open source projects and provides the results directly to the development team. Over 100,000 defects have been fixed that were identified by this system. Since February 23, 2006, OpenSSL has been one of the projects scanned by Coverity Scan. In the "Coverity Scan Security Spotlight" report ("Coverity Releases Security Spotlight Report on Critical Security Defects in Open Source Projects", 2014), the Coverity team discusses how tainted data can be exploited in Heartbleed and how other similar vulnerabilities can be detected and fixed in the open source community.

4.1.2. Coverity on-premise

Companies who purchase Coverity for use on their internal source code traditionally deploy it on-premise and include the tool as part of the build process. At Adobe, we do this via a continuous build system as well as using an IDE plugin that developers use while writing the code.

4.1.3. Coverity challenges

The main challenge that static analysis tools face is how to balance the amount of false positives that a tool generates with the effectiveness of the tool finding actual bugs in the code. One of the reason Coverity's static analysis tools are so popular is that they have a low false positive rate. If developers and quality engineers are spending time investigating false positives they are not spending time finding and fixing real bugs. There isn't one set rule for how many false positives are allowed before a developer stops trusting the tool. A good benchmark is less than 20%. Coverity's desire to use fast algorithms and keep their false positive rate low prevented the 7.0.3 version of their static analysis engine from detecting Heartbleed.

4.1.4. Detecting Heartbleed with Coverity

When Coverity's CTO and co-founder Andy Chou heard about Heartbleed, he blogged about his team's investigations (Chou, 2014). Heartbleed was a buffer over-read on the memcpy() function call. More specifically it was a buffer over-read involving tainted data. It is difficult to determine that the tainted data came from an untrusted source like a network socket or from an attacker. Coverity wasn't designed to test all of

the memcpy() statements in a program and assume they have tainted data in them. This would have invariably led to a larger than acceptable false positive rate. A way to make the problem more tractable is to examine if the data was part of a byte-swap operation. There are many use cases for byte-swap operations but one common one is sending data across a network. When performing that operation, you cannot assume that the server has the same endianness as the client and vice-versa. Programmers use network byte order and then either do a byte-swapping operation or not depending on if that matches your computers endianness.

In the case of Heartbleed, the *n2s* macro was designed to do the byte-swapping. By examining the code preceding the memcpy(), and determining that a byte swapping operation was occurring, Coverity could now make a reasonable assumption that the data was from an untrusted source and could be tainted.

Coverity 7.0.0.3 was released on April 23, 2014, which detected Heartbleed in the default configuration by adding a TAINTED_SCALAR checker (Coverity Support, 2014).

4.1.5. Modeling

Another approach Coverity could have used was to model the Heartbleed bug. Coverity provides a modeling feature in order for developers to teach the static analysis engine about specific programming constructs in the code. Because Heartbleed was a specific bug, the easier approach would have been to simply model the exact behavior.

Modeling has its place and Adobe has used it successfully in the past. Coverity's design decision to detect Heartbleed with a new TAINTED_SCALAR checker solves not only this particular vulnerability, but also future vulnerabilities that use the same byte swapping design. This "clever" approach has already been improved upon by GrammaTech's CodeSonar (Anderson, 2014).

5. Conclusions

Heartbleed was a serious vulnerability and there are other serious vulnerabilities that have yet to be disclosed. The Stuxnet worm used four distinct zero-day

vulnerabilities in its search for computers used to control Iran's nuclear facilities (Zetter, 2014). It takes increased skill to find and exploit one vulnerability, making it more impressive that Stuxnet contained four. The national debate continues if vulnerabilities should be disclosed when they are discovered or kept for exploitation later by the NSA (Zetter, 2014). As that discussion continues, development teams should employ all of the available tools to remove vulnerabilities in the first place. By improving our static analysis detection algorithms, development teams have a better chance at catching these bugs before they are exploited.

Jeff Sass, jsass@adobe.com

6. References

(2014, December 11). Retrieved January 10, 2015, from http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

April 2014 Web Server Survey (2014). Retrieved from http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html

Anderson, P. (2014, May 1). Finding Heartbleed with CodeSonar. Retrieved January 29, 2015, from http://www.grammatech.com/blog/finding-heartbleed-with-codesonar

Bagley, S. (2014, April 18). Heartbleed, Running the Code. Retrieved January 26, 2015, from https://www.youtube.com/watch?v=1dOCHwf8zVQ

Chou, A. (2014, April 13). On Detecting Heartbleed with Static Analysis. Retrieved January 27, 2015, from http://security.coverity.com/blog/2014/Apr/on-detecting-heartbleed-with-static-analysis.html

Christey, S. (2011, September 13). 2011 CWE/SANS Top 25 Most Dangerous Software Errors. Retrieved January 21, 2015, from http://cwe.mitre.org/top25/#CWE-807

Cipriani, J. (2014, April 9). Heartbleed bug: Check which sites have been patched - CNET. Retrieved January 20, 2015, from http://www.cnet.com/how-to/which-sites-have-patched-the-heartbleed-bug/

Colbert, S. (2014). [Television series episode]. In *The Colbert Report*. New York: Stephen Colbert.

Coverity Support. (2014, April 23). [Email] "Request access to heartbleed hotfix"

Coverity. (n.d.). Retrieved January 26, 2015, from http://www.coverity.com/company/

Coverity Releases Security Spotlight Report on Critical Security Defects in Open Source Projects. (2014, October 15). Retrieved January 25, 2015, from http://www.coverity.com/press-releases/coverity-releases-security-spotlight-report-on-critical-security-defects-in-open-source-projects/

Crispin, L., & Gregory, J. (2009). Critiquing The Product Using Technology-Facing Tests. In *Agile Testing: A Practical Guide for Testers and Agile Teams*. Upper Saddle River, NJ: Addison-Wesley.

CWE/SANS TOP 25 Most Dangerous Software Errors. (2010, February 16). Retrieved January 10, 2015, from http://www.sans.org/top25-software-errors/2010/

Jeff Sass, jsass@adobe.com

Durumeric, Z., Kasten, J., Adrian, D., Halderman, J., Bailey, M., … (2014). The Matter of Heartbleed. *Proceedings of the 2014 Conference on Internet Measurement Conference,* 475-488. Retrieved January 20, 2015, from https://jhalderm.com/pub/papers/heartbleed-imc14.pdf

Gassner, D. (n.d.). Lynda.com Training | Heartbleed Tactics for Small IT Shops. Retrieved January 9, 2015, from http://www.lynda.com/Developer-Servers-tutorials/Heartbleed-Tactics-Small-Shops/169720-2.html

Gregory, J., & Crispin, L. (2014). Using Models To Help Plan. In *More Agile Testing: Learning Journeys for the Whole Team*. Upper Saddle River, NJ: Addison-Wesley.

Grubb, B. (2014, April 15). Heartbleed disclosure timeline: Who knew what and when. Retrieved January 20, 2015, from http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html

Half a million widely trusted websites vulnerable to Heartbleed bug. (2014, April 8). Retrieved February 8, 2015, from http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html

Heartbleed Update. (2014, April 17). Retrieved January 26, 2015, from http://blogs.adobe.com/psirt/?p=1085

Hern, A. (2014, April 9). Retrieved January 20, 2015, from http://www.theguardian.com/technology/2014/apr/08/heartbleed-bug-puts-encryption-at-risk-for-hundreds-of-thousands-of-servers

Internet security flaw may affect YOU. (2014, April 9). Retrieved January 20, 2015, from http://www.cnn.com/videos/tech/2014/04/09/mxp-heartbleed-internet-security-flaw.hln

Krebs, B. (2014, April 14). Heartbleed Bug: What Can You Do? Retrieved January 27, 2015, from http://krebsonsecurity.com/2014/04/heartbleed-bug-what-can-you-do/

Newman, L. (2014, April 10). Heartbleed Should Motivate You to Get a Password Manager. Retrieved January 26, 2015, from

Jeff Sass, jsass@adobe.com

http://www.slate.com/blogs/future_tense/2014/04/10/password_managers_can_protect_you_from_vulnerabilities_like_heartbleed.html

OpenSSL vulnerabilities. (2014, April 8). Retrieved January 26, 2015, from http://openssl.org/news/vulnerabilities.html

Ramzan, Z. (2014, April 8). OpenSSL Heartbeat (Heartbleed) Vulnerability (CVE-2014-0160) and its High-Level Mechanics. Retrieved January 27, 2015, from http://vimeo.com/91425662

Seeley, J. (2014, April 1). Lynda.com Training | Protecting Yourself from the Heartbleed Bug. Retrieved January 9, 2015, from http://www.lynda.com/Business-Computer-Skills-Mac-tutorials/Protecting-Yourself-from-Heartbleed-Bug/169873-2.html

Seggelmann, R. (2012, February 1). Retrieved January 26, 2014, from http://tools.ietf.org/pdf/rfc6520.pdf

The Heartbleed Bug. (2014, April 8). Retrieved January 10, 2015, from http://www.heartbleed.com

The Heartbleed Story. (2014). Retrieved January 26, 2015, from http://www.codenomicon.com/resources/brochure/pdf/Heartbleed-Story.pdf

Ullrich, J. (2014, April 9). How to talk to your kids (or manager) about "Heartbleed" Retrieved January 21, 2015, from https://isc.sans.edu/forums/diary/How+to+talk+to+your+kids+or+manager+about+Heartbleed/17943

Vulnerability Summary for CVE-2014-0160. (2014, December 11). Retrieved January 10, 2015, from http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

Wheeler, D. (2014, October 20). How to Prevent the next Heartbleed. Retrieved January 29, 2015, from http://www.dwheeler.com/essays/heartbleed.html

Williams, J. (2014, April 9). OpenSSL "Heartbleed" Vulnerability. Retrieved April 9, 2014, from https://www.sans.org/webcasts/openssl-heartbleed-vulnerability-98105

Williams, J., & Goodman, A. (2014, May 22). How Defense-In-Depth Helps Protect You From Unexpected Vulnerabilities Like Heartbleed. Retrieved January 9, 2015, from https://www.sans.org/webcasts/defense-in-depth-helps-protect-unexpected-vulnerabilities-heartbleed-98155

Jeff Sass, jsass@adobe.com

Wood, M. (2014, April 9). Flaw Calls for Altering Passwords, Experts Say. Retrieved January 20, 2015, from http://www.nytimes.com/2014/04/10/technology/flaw-calls-for-altering-passwords-experts-say.html

xkcd: Heartbleed Explanation. (2014, April 9). Retrieved January 20, 2015, from http://xkcd.com/1354/

Zetter, K. (2014). Digital Pandora. In Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon. New York: Crown.

(n.d.). Retrieved January 20, 2015, from https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/scanner/ssl/openssl_heartbleed.rb

Jeff Sass, jsass@adobe.com