WHITEPAPER

SYNOPSYS®

Diary of a Heartbleed

TABLE OF CONTENTS

Page 3: Discovering Heartbleed

Page 3: Test Suites

Page 4: The Discovery

Page 5: What is the Heartbeat protocol?

Page 6: Branding Vulnerabilities

Page 7: Trouble in the Supply Chain

Page 8: Removing Heartbleed

Page 8: Software Testing Tools

Discovering Heartbleed

Heartbleed is a SSL/TLS vulnerability found in older versions of OpenSSL. It was independently co-discovered in April 2014 by the Synopsys research team in Finland (formerly Codenomicon) and by Neel Mehta of Google's security team. According to Mark Cox at OpenSSL, "the coincidence of the two finds of the same issue at the same time increases the risk while this issue remained unpatched. OpenSSL therefore released updated packages [later] that day."[1] Officially, the world first learned about the Heartbleed vulnerability on April 7, 2014, when the open source organization OpenSSL issued a fix.

The official Common Vulnerabilities and Exposures (CVE) reference to Heartbleed, as issued by Standard for Information Security Vulnerability Names maintained by MITRE, is CVE-2014-0160.[2] However a common name was chosen to help identify it.

The Heartbleed vulnerability affects how OpenSSL implements the heartbeat protocol in TLS. In computing, a heartbeat, or a simple data message, typically determines the persistence of another machine in a given transaction; in this case, a heartbeat determines the persistence of the encryption between a client and a server. Heartbleed allows an attacker to request data more than a simple response; in other words, it could allow for the leakage of passphrases and encryption keys.

So how did the researchers find Heartbleed?

Test suites

Criminal hackers think outside the box. They often do not use software the way it was intended; they come at it in ways that the developer may not have considered. Using this technique, criminal hackers often find previously unknown vulnerabilities or zero-days. Sometimes the vulnerabilities they find are exploitable and significant.

Fuzz testing is one way to simulate this random way of thinking. Fuzz testing pumps random bits of data at a port. The input chosen assumes no protocol knowledge (in other words, the bits don't necessarily follow formatting and other protocol rules). However, generating random bits provides shallow code coverage, requires a lot of time, and is only occasionally effective. It is like waiting for a group of monkeys to pound out all the works of William Shakespeare on a keyboard.

Instead, one might produce a template which would provide capture and playback capabilities. The problem is that you lose the randomness and the templates are biased toward preconceived conventions. It is the opposite problem of random bits; in this case, the data might is too narrow.

The best method is to use a test suite. Ideally one that provides a complete model of a protocol according its specifications (RFC's, 3GPP, etc). Test suites systematically fuzz messages and fields looking to test boundary conditions, bad checksums and lengths, and troublesome strings specific to a protocol. Comprehensive suites provide fully state aware testing for all protocols and leverages the response of the system under test (SUT) to provide smart behavior adaptation capabilities. This allows tests to extend to rare and often vulnerable protocol elements and not just data on the wire.

While developing a test suite for a boundary condition in the heartbeat sub-protocol of TLS in OpenSSL, Heartbleed was first discovered.

1 https://plus.google.com/+MarkJCox

2 https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160

The discovery

Heartbleed was discovered while developing new tests, collectively called SafeGuard, for the TLS suite of the Synopsys fuzzing tool (Defensics). In these tests, the SafeGuard test suite attempts to expose failed cryptographic certificate checks, privacy leaks, or authentication bypass weaknesses that allow man-in-the-middle (MitM) attacks and eavesdropping. To test the new features in our fuzzing tool, researchers used the latest versions of OpenSSL (1.0.1f) and GnuTLS (3.2.12) as test targets.

"Testing open source implementation comes as a 'by-product,' " explained Sami Petajasoja, Defensics Product Manager at Synopsys. The process uses negative testing techniques for feeding invalid, unexpected, or even random data to the SUT. "The goal is to find security vulnerabilities and other defects in SUT file format and protocol implementations," he said. "This type of fuzz testing does not require access to source code and, therefore, can be used for testing proprietary software."

Fuzzing works by sending specially formed input to a server.

```
GET AaAaAaAaAaAaAaAaAaAa HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

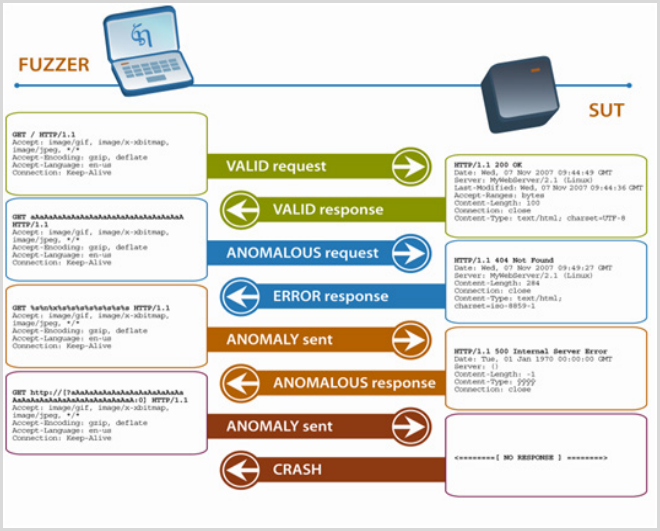
In response, the server might respond with:

```
HTTP/1.1 404 Not Found
Date: Wed, 07 Nov 2007 09:49:27 GMT
Server: MyWebServer/2.1 (Linux)
Content-Length: 284
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

However, when malformed input is sent …

```
GET http://[?aAaAaAaAaAa::0] HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive
```

… a variety of consequences might occur such as crashes, denial of service, security exposures, degradation of service, thrashing, or anomalous behavior.

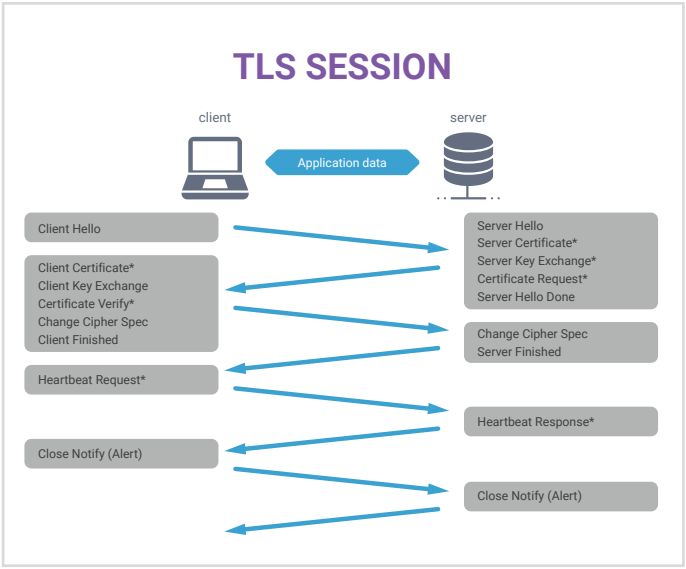


Our fuzzing team has used this technique before and have several other open source vulnerabilities. These include:

- Numerous flaws in ASN.1/SNMP in 2001/2002
- Apache IPv6-URI flaw in 2004
- Numerous flaws in image formats in 2005
- Numerous flaws in XML libraries in 2009
- Several flaws in Linux Kernel IPv4 and SCTP in 2010
- RSA signature verification vulnerability in strongSwan in 2012
- Several OpenSSL and GnuTLS vulnerabilities in 2004, 2008, 2012, and 2014

What is the Heartbeat protocol?

In a typical SSL connection, the client and server establish a secure (meaning encrypted) line of communication. The peer sends a heartbeat request and the other peer responds by sending a copy of the request's payload. The use of the Heartbeat extension is negotiated during the TLS handshake. During this process, the client may send a Datagram Transport Layer Security (DTLS) message to make sure the other peer is still alive.



Heartbleed is not a design flaw within the TLS/DTLS (transport layer security protocols) heartbeat extension (RFC6520), but, rather, it is an implementation problem in OpenSSL. The implementation change in OpenSSL

was introduced in version 1.0.1 which was released in December 2011 and has been public since March 2012. So Heartbleed was present, but unknown, for two years.

In OpenSSL (especially in the vulnerable versions 1.0.1-through-1.0.1f) the heartbeat protocol support is compiled and installed by default, and does not require the negotiation of the Heartbeat extension. The vulnerable versions of OpenSSl also do not require that the TLS handshake is completed before a Heartbeat request is sent. Since this vulnerability can be exploited before authentication, anonymous attacks can occur.

"Heartbleed is an example of an elusive vulnerability," said Petajasoja. "At first glance, the only indication was the suspiciously large size of the server replies. It would be very hard for a human to notice this from hundreds of thousands of lines from test logs. Our tools are automated, and our fuzzing tool caught it immediately."

According to the protocol, a single heartbeat allows for a 64kbs payload. However, without a bounds check, the vulnerable versions of OpenSSL allows an attacker to repeatedly request 64kb chunks of data. For example, a simplified heartbeat message might be a client request to send back the word "bird" (4 characters), and so the server would respond with "bird" (4 characters). In a Heartbleed attack, the malicious client might say instead send me the word "bird" (500 characters). In order to provide those 500 characters, a vulnerable version of OpenSSL might copy arbitrary memory content in order to fill that request. As it appeared, the memory content can contain all kinds of sensitive data including the confidential keys from the OpenSSL server.

For a proof of concept, the researchers attacked themselves. "We have tested some of our own services from an attacker's perspective. We attacked ourselves from outside, without leaving a trace," they wrote on Heartbleed.com. "Without using any privileged information or credentials, we were able steal the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication."

The Defensics team reported Heartbleed to the Finnish Computer Emergency Response Team (CERT-FI) for verification. CERT-FI also began reaching out to the authors of OpenSSL, software, operating systems, and appliance vendors. This activity is supported by a timeline later published by the Sydney Morning Herald which showed that some of the contacted vendors mitigated the vulnerability before the public disclosure.[3]

Branding vulnerabilities

Jerimiah Grossman, founder of WhiteHat security, has seriously suggested common names be used for severe vulnerabilities much the way the National Weather Service names hurricanes.[4] Indeed the national weather service states, "experience shows that the use of short, distinctive names in written as well as spoken communications is quicker and less subject to error than the older, more cumbersome latitude-longitude identification methods."[5]

One of the first examples of this was BEAST. Short for Browser Exploit Against SSL/TLS. SSL Beast, revealed in late September 2011, is an exploit, that leverages weaknesses in cipher block chaining (CBC) to exploit the Secure Sockets Layer (SSL) protocol. The CBC vulnerability enables man-in-the-middle (MITM) attacks against SSL, providing hackers with access to the data passed between a Web server and a Web browser.

With the Heartbleed vulnerability discovery, came the need to inform as many system administrators as

3 http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140414-zqurk

4 https://twitter.com/jeremiahg/status/712456571515592706

5 http://www.nhc.noaa.gov/aboutnames_history.shtml

possible. The CVE designation, CVE-2014-0160, although technically accurate, is not an effective shorthand for communicating a vulnerability of this magnitude. Especially for one that crosses a variety of different constituencies. Thus was given a nickname, Heartbleed, because the vulnerability leaked data using the heartbeat extension. The Heartbleed nickname proved to be more effective when communicating new details around this vulnerability.

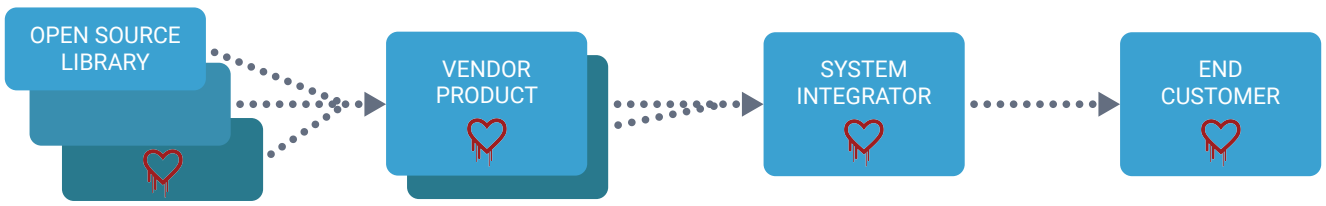
The Defensics team registered the "heatbleed.org" website on April 5, 2014. However, details of Heartbleed leaked before all the affected vendors could were properly informed. An icon was designed quickly and a page was made public shortly after the team at OpenSSL issued a fix on April 7, 2014. The intent of the page, the icon, and the use of a nickname was to get as much awareness as possible. The site continues to support a wealth of technical details necessary to mitigate the vulnerability.[6]

After Heartbleed, other vulnerabilities have been named—DROWN, POODLE, and Ghost. Whether each of these vulnerabilities are as significant as Heartbleed and/or warrant a such convenient naming scheme is debatable.

Trouble in the supply chain

Within the first month, roughly half of the vulnerable IP systems on the Internet were either patched or otherwise mitigated. These were obvious uses of the vulnerable versions of OpenSSL such as ecommerce and banking sites. However, there remain hundreds of thousands of less obvious uses of OpenSSL software—even today.

OpenSSL remains one of the more widely used software libraries. It's possible that many derivative software products contain this vulnerability because there's an underlying assumption that open source projects are "okay" to use without further testing. In many cases, derivative software products also do not have the mechanisms to support updates should vulnerabilities be found after general availability. So a wide variety of software development projects might contain and even spread the Heartbleed vulnerability to other products in



the supply chain. The problem, therefore, becomes much harder to mitigate.

Getting software right—and secure—is compounded by the fact that today's software is no longer written, but mostly assembled. Up to 90 percent of software might be third-party code adopted within the early stages of the software development lifecycle (SDLC). First-party code is often the glue that not only stitches the assembled components together, but also adds unique IP and product differentiation. However, without a Bill of Materials, without software composition analysis, a developer cannot say for sure whether the third party code added to a project is, in fact, secure.

Additionally, developers should use gating techniques throughout the various stages in the SDLC. At Synopsys we call this process Software Signoff. Software Signoff introduces formal testing gates at a number of places

6 http://heartbleed.com/

within the software development lifecycle (SDLC). For example, the introduction of static analysis testing each time the software is checked in during development can eliminate the high costs of patches and mitigations later in the cycle. And periodic testing of the software against known CVEs and Common Weakness Enumerations (CWEs) can further eliminate known vulnerabilities such as Heartbleed from existing software products.

Removing Heartbleed

Heartbleed has gotten more publicity than any previous vulnerability. The upside is the community reacted quickly and started mitigating the problem almost immediately. The mainstream reporting of Heartbleed helped raise awareness on the importance of mitigating security vulnerabilities in general.

> The best way to mitigate a Heartbleed vulnerability is to upgrade OpenSSL to a fixed version.

The best way to mitigate a Heartbleed vulnerability is to upgrade OpenSSL to a fixed version (version 1.0.1g or later). If that is not possible, the next best method is to disable the Heartbeat protocol functionality by recompiling OpenSSL with the Heartbeat flag off. However, doing either of these is not enough. Not only do you need to generate new secret and public key-pairs, but you must also create new certificates and revoke old ones. Of course, as these tasks are performed, clients should change their passwords.

At the time of discovery, there were estimated to be roughly 600 thousand servers in the world vulnerable to Heartbleed. By June 2014, two months after disclosure, that number was estimated to be 300K by security researcher Robert Graham.[7]

The downside of all this attention was that the rush to mitigate also led to mistakes. True mitigation for Heartbleed requires multiple steps. Completing only a few steps creates a false sense of security.

For example, the University of Maryland analyzed the Heartbleed patch status of over a million popular websites in the US (November 2014)[8]. Among their findings, "…approximately 93 percent of the websites analyzed had patched their software correctly within three weeks of Heartbleed being announced, only 13 percent followed up with other security measures needed to make the systems completely secure…"

There is an additional requirement that certificates used by vulnerable websites must be revoked and reissued after Heartbleed has been mitigated, but many sites did not perform this additional task or did so incorrectly. The University of Maryland authors concluded "…Many people seem to think that if they reissue a certificate, it fixes the problem, but, actually, the attack remains possible just as it did before. So, you need to both reissue and revoke the certificates."

Software testing tools

Many feel that since open source software is widely used, and people have reviewed it many times to make sure it is secure and of high quality. As it appears, OpenSSL contains a lot of different functionality, even beyond the basics of what is required and it is fairly complex. It has accumulated a lot of functionality over long periods of time, not all of which has been properly tested. So, with Heartbleed, people are starting to realize that if something is open source it doesn't necessarily mean it has been reviewed by people.

Fortunately, there are a number of tools available to address open software testing needs. By utilizing multiple tools, such as static code analysis, fuzz testing, software composition analysis, and testing and validation suites, users can benefit from a diverse testing portfolio and leverage a strategy of defense in depth.

- **Static Analysis:** Static analysis identifies logical inconsistencies and other indications that the developer likely didn't implement a feature correctly. When source code is available, static analysis identifies specific types of bugs and vulnerabilities—even while the product is still under development. Unlike dynamic testing tools, such as validation suites and fuzzing, static analysis does not actually run the product during testing. Instead, it analyzes the program structure and logic and then looks for indications of undesirable behavior.
- **Fuzz Testing:** Fuzzing is a dynamic testing tool that finds a wide variety of problems affecting both security and reliability. To verify that products behave correctly in the presence of invalid or unexpected inputs, fuzz testing tools supply carefully-designed inputs over the external interfaces. Simple fuzzers might send random data to the device, while sophisticated fuzzing tools will intelligently structure the data to test both random and malicious scenarios.
- **Software Composition Analysis:** Software composition analysis (SCA) produces an accurate Bill of Materials (BoM) for a particular product, verifying that licensing obligations are met, that all appropriate components have been tested, and that security vulnerabilities—and the need for future updates—are tracked after products are released into the wild. SCA works by scanning binary or source code to identify known components from a database of open source projects and releases, or from proprietary components and releases added by users. SCA guides development decisions about which components to use, when to release updates and patches, and when to upgrade the components they are using in their own product.
- **Testing and Validation Suites:** To verify that protocols are implemented correctly and products behave as intended in the presence of valid inputs, validation suites demonstrate conformance with the standards defining the protocol. Validation often consists of tests, such as unit and integration tests, utilized during development as well as later stage functional and exploratory testing. With testing and validation suites, users are dynamically testing the product—meaning they need to have and to run a test case for each bit of functionality to be tested.

THE SYNOPSYS DIFFERENCE

Synopsys offers the most comprehensive solution for integrating security and quality into your SDLC and supply chain. Whether you're well-versed in software security or just starting out, we provide the tools you need to ensure the integrity of the applications that power your business. Our holistic approach to software security combines best-in-breed products, industry-leading experts, and a broad portfolio of managed and professional services that work together to improve the accuracy of findings, speed up the delivery of results, and provide solutions for addressing unique application security challenges. We don't stop when the test is over. Our experts also provide remediation guidance, program design services, and training that empower you to build and maintain secure software.

For more information go to www.synopsys.com/software

SYNOPSYS®
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: (800) 873-8193
International Sales: +1 (415) 321-5237
Email: software-integrity-sales@synopsys.com