« Previous: Recognizing another type of… (https://www.synopsys.com/blogs/software-security/another-threat-non-targeted-attacks/)

Next: The Complete Security… (https://www.synopsys.com/blogs/software-security/security-vulnerability-assessment-checklist/) »

Heartbleed bug: How it works and how to avoid similar bugs

Anil Gajawada

Posted by (https://www.synopsys.com/blogs/software-security/author/agajawada/) on Tuesday, September 6, 2016

The Heartbleed bug results from improper input validation in the OpenSSL's implementation of the TLS Heartbeat extension. How can we prevent similar bugs?

The Heartbleed (http://heartbleed.com/) bug is a vulnerability in open source software that was first discovered in 2014. Anyone with an internet connection can exploit this bug to read the memory of vulnerable systems, leaving no evidence of a compromised system.

Heartbleed is an implementation bug (CVE-2014-0160 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160)) in the OpenSSL cryptographic library (https://www.openssl.org/). OpenSSL is the most popular open source cryptographic library (written in C) that provides Secure Socket Layer (SSL) and Transport Layer Security (TLS) implementation to encrypt traffic on the internet. Even though the bug is in the OpenSSL library, it has nothing to do with the SSL/TLS protocols. The vulnerability is in the OpenSSL code that handles the Heartbeat extension (RFC 6520 (https://tools.ietf.org/html/rfc6520)) for TLS/DTLS.

OpenSSL versions 1.0.1 through 1.0.1f are vulnerable unless compiled with the uncommon -DOPENSSL_NO_HEARTBEATS option. The earliest non-vulnerable version is 1.0.1g.

How does the Heartbleed bug work?

The Heartbleed bug got its start from improper input validation in the OpenSSL implementation of the TLS Heartbeat extension. Due to the missing bounds check on the *length* and *payload* fields in Heartbeat requests, coupled with trusting the data received from other machines, the responding machine mistakenly sends back its own memory data.

During a TLS encrypted handshake, two machines send each other Heartbeat messages. According to RFC 6520, a Heartbeat response needs to contain the exact copy of the payload from the Heartbeat request. When a Heartbeat request message is received, the machine writes the payload contents to its memory and copies the contents back in response. The *length* field is meant to be the length of the *payload*. OpenSSL allocates memory for the response based on *length* and then copies the *payload* over into the response using memcpy().

Attackers can send Heartbeat requests with the value of the *length* field greater than the actual length of the *payload*. OpenSSL processes in the machine that are responding to Heartbeat requests don't verify if the *payload* size is same as what is specified in *length* field. Thus, the machine copies extra data residing in memory after the *payload* into the response. This is how (https://xkcd.com/1354/) the Heartbleed vulnerability works. Therefore, the extra bytes are additional data in the remote process's memory.

What is the impact of Heartbleed?

The Heartbleed bug allows anyone on the internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. Sensitive information such as session identifiers, usernames, passwords, tokens, and even the server's private cryptographic keys, in some extreme cases, can be extracted from the memory. To make matters worse, this attack leaves no apparent evidence in the log files. Thus, it is extremely difficult to determine whether the machine has been compromised.

An attacker can read 64 kilobytes of server memory for a single Heartbeat message. However, there is no limit to the amount of memory that can be read from a vulnerable server. Furthermore, an attacker can continue reconnecting and requesting an arbitrary number of 64-kilobyte segments to reveal secrets (passwords, secret keys, credit card numbers, etc.) stored in memory.

Could it have been avoided?

The problem could have been avoided by validating the message length and ignoring Heartbeat request messages asking for more data than their payload needs. A security review of OpenSSL software could have also caught the Heartbleed bug.

What lessons can we learn from the Heartbleed bug?

Heartbleed is a simple bug to remediate. To prevent sensitive data leakage through Heartbleed, upgrade to the latest stable version of OpenSSL. Don't blindly trust the software—it's also best to integrate security programs into software development.

How can we prevent similar bugs in software?

Looking ahead, we offer four key pieces of advice to integrate into your software development life cycle and the maintenance of existing software:

- Integrate security-related activities, which are required for creating secure software.
- Give all open source software and components in use a thorough security review to avoid using components with known vulnerabilities (https://www.owasp.org/index.php/Top_10_2013-A9-Using_Components_with_Known_Vulnerabilities).
- Never trust the input received from an external system.
- Always perform server-side input validation.

This post is filed under Open source and software supply chain risks (https://www.synopsys.com/blogs/software-security/category/open-source-and-software-supply-chain-risks/).

Anil Gajawada

Posted by
Anil Gajawada

Anil Gajawada is a security consultant at Synopsys. He has over 5 years of security experience in a variety …

SEE AUTHOR ARCHIVE (https://www.synopsys.com/blogs/software-security/author/agajawada)

More from Open source and software supply chain risks

Building a software Bill of Materials with Black Duck (https://www.synopsys.com/blogs/software-security/building-sbom-with-black-duck/)

Posted by Mike McGuire (https://www.synopsys.com/blogs/software-security/author/mmcguire/) on August 3, 2022

Software compliance, quality, and standards (https://www.synopsys.com/blogs/software-security/tag/software-quality-compliance/)

Software composition analysis (https://www.synopsys.com/blogs/software-security/tag/software-composition-analysis/)

CyRC Vulnerability Analysis: Repo jacking in the software supply chain (https://www.synopsys.com/blogs/software-security/cyrc-vulnerability-analysis-repo-jacking/)

Theo Burton
Posted by (https://www.synopsys.com/blogs/software-security/author/burtont/) on August 2, 2022

Cybersecurity Research Center (https://www.synopsys.com/blogs/software-security/tag/cybersecurity-research-center/)

Five types of software licenses you need to understand (https://www.synopsys.com/blogs/software-security/5-types-of-software-licenses-you-need-to-understand/)

Phil Odence
Posted by (https://www.synopsys.com/blogs/software-security/author/podence/) on July 27, 2022

Open source license compliance (https://www.synopsys.com/blogs/software-security/tag/open-source-license-compliance/)

Software compliance, quality, and standards (https://www.synopsys.com/blogs/software-security/tag/software-quality-compliance/)

AppSec Decoded: Get the most out of your open source software (https://www.synopsys.com/blogs/software-security/appsec-decoded-open-source-software-ossra/)

Posted by Synopsys Editorial Team (https://www.synopsys.com/blogs/software-security/author/synedt/) on July 14, 2022

Mergers and acquisitions due diligence (https://www.synopsys.com/blogs/software-security/tag/mergers-acquisitions/)

Get Newsletter

GET THE OSSRA REPORT

(https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html?intcmp=sig-blog-os-rightrail)

RELATED TAGS

Web application security (https://www.synopsys.com/blogs/software-security/tag/web-application-security/)

SEE ALL TAGS

SYNOPSYS® (https://www.synopsys.com/)

PRODUCTS

RESOURCES

Application Security (/software-integrity.html)

Semiconductor IP (/designware-ip.html)

Verification (/verification.html)

Design (/implementation-and-signoff.html)

Silicon Engineering (/silicon.html)

Solutions (/solutions.html)

Services (/services.html)

Support (/support.html)

Community (/community.html)

Manage Subscriptions (https://online.synopsys.com/contact-form-subscription-center.html)

LEGAL

Privacy (/company/legal/privacy-policy.html)

Trademarks & Brands (/company/legal/trademarks-brands.html)

Software Integrity Agreements (/company/legal/software-integrity.html)

CORPORATE

About Us (/company.html)

Careers (/careers.html)

CSR Report (/company/corporate-social-responsibility.html)

Inclusion & Diversity (/careers/inclusion-diversity.html#present)

Investor Relations (/company/investor-relations.html)

Contact Us (/company/contact-synopsys.html)

FOLLOW

(https://twitter.com/Synopsys)