Heartbleed Keeps Flowing - Open Source Security

Melissa Iori (miori01)

Abstract

Heartbleed is a vulnerability discovered in OpenSSL, in 2012. The bug allows attackers to receive confidential data from servers running affected versions of OpenSSL by specifying a requested response length that is much larger than the requested payload [1]. This catastrophic vulnerability in a highly ubiquitous software project had a negative impact on the reputation of open source projects. Are open source projects more secure because they are more highly scrutinized, or less so? How can future vulnerabilities of this nature be prevented? What was learned from Heartbleed? These are the questions my paper seeks to address. I have also created a small, lightweight demo, in JavaScript, to visually demonstrate how such a small bug was able to cause widespread data leakage.

Introduction

OpenSSL is an implementation of TLS that runs on a significant percentage of Web servers today [2]. In 2014, a major vulnerability was discovered in OpenSSL. This vulnerability had a simple cause - a missing bounds check on data being read out of memory [1]. However, the implications of this little bug were serious. The Heartbleed vulnerability in OpenSSL (an open source implementation of SSL/TLS) was disclosed in 2014. The exploitation of Heartbleed leaves behind no obvious trace in any server log files, so owners of vulnerable servers will likely not know that they have been attacked until their users are negatively affected by the information leakage caused by Heartbleed [1]. One benefit of open source software is its availability to any and all who wish to audit it. Since the code can be analyzed by any number of security professionals, experts, and testers, some assume that it is, in fact, being audited. This particular fallacy contributes to a large security gap in the open source software community. Attribution can sometimes relate to code quality, and instead of whats being done, it‘s what isn‘t being done that damages the quality of open source software.

To the Community

Open source software is more crucial than ever to todays businesses and developers. This paper details Heartbleed, a critical vulnerability in the open source OpenSSL, but every open source project, indeed every software project, currently in existence deals with both design flaws and implementation bugs. Developers should be aware of the importance of code reviews, and enterprises should acknowledge the risks of open source and consider ways that their own developers and security analysts can contribute to open source projects.

Heartbleed

OpenSSL is described as a "Cryptography and SSL/TLS Toolkit" [2]. It is a widely used implementation of SSL/TLS written in C and hosted on Github. The project is actively maintained by a community of committers. The team for OpenSSL is not especially large for such a critical project. As commentator John Walsh pondered, "Think about it, OpenSSL only has two people to write, maintain, test, and review 500,000 lines of business critical code" [3].

On April 1, 2014, Heartbleed was secretly reported by Neel Mehta, a member of Google's security team. CVE-2014-0160 was reserved for the Heartbleed vulnerability [1].

Heartbleed is described as a "buffer over-read" vulnerability. In TLS, the "heartbeat" of a connection allows a secure connection to stay "alive" by ensuring that the server and client are still successfully communicating at certain intervals. The heartbeat request sends a string of data and the length of that string. The server then returns the same string to the client. To exploit Heartbleed, an attacker can provide a request with a short string and specify a string length that is much longer than the actual string. When the request reaches the server (it can also be sent to a TLS client, but a server is usually the preferred target), the server will allocate a memory buffer according to the size of the request length - which, when Heartbleed is being exploited, will typically be a much larger buffer than is actually needed. Thus, when the string message is sent back to the client, it also sends back whatever data happened to be contained in that extra allocated memory - which could be any number of things including user passwords, session keys, social security numbers and lots of other critical data stored on the server that is in memory at the time of the attack. While the attacker can control the amount of data to expose, they cannot

control what memory location to start reading from, so they may end up reading through a lot of arbitrary data before finding something valuable [1] [4].

The implications of the Heartbleed vulnerability are far-reaching. After disclosure, several attacks utilizing Heartbleed were reported. It is still unknown how many X.509 private keys from websites could have been leaked due to Heartbleed, which could compromise secure communications with those sites if used [1]. All servers running OpenSSL version 1.0.1 through 1.0.1f need to be patched to version 1.0.1g or higher [1]. Educating users on the severity of the vulnerability and the need to patch affected servers was a long process that has yet to be fully realized. A website, heartbleed.com, was created to educate the public about this vulnerability and what can be done to protect against it [1].

According to the Pew Research Centers report on public response to HeartBleed, only 39% of Internet users claimed to take action to protect their data by changing passwords or deleting accounts [5].

Open Source Security and Vulnerabilities

When an open source vulnerability is discovered and disclosed, open source developers act quickly to release a patched version of their software. However, with small teams and limited funding, getting the resources to respond quickly and effectively is at risk.

All open source projects need a plan when it comes to vulnerabilities. Many vulnerabilities were reported in 2017, including in popular open source libraries such as CPython, Moment.js and Spring-LDAP [6]. The problem extends to projects both open source and outside of open source, such as highly used frameworks like Spring, which reported both CVE-2016-9879 and CVE-2016-5007. Both vulnerabilities involve security control for certain paths or resources, which could result in directory traversal exploits. Angular.js, a very popular JavaScript framework, suffered from a lack of user input sanitization of a certain parameter which could allow cross-site scripting attacks. In Apache Tomcat, CVE-2017-12617 was discovered, which leaves systems open to remote command execution attacks [7].

In short, the good reputation and large userbase of certain projects is not enough to ensure that they will be secure. Especially within large frameworks and platforms, new vulnerabilities are discovered and disclosed every day, making users question whether their private information is safe.

All open source projects must come up with a plan for patching vulnerabilities and educating users about the impacts. The Open Source Vulnerability Database cataloged known vulnerabilities in open source software, but has shut down [8].

Action Items

Open source project leaders and the enterprises who sponsor open source projects may sometimes worry about the reputation and good name of their software when dangerous vulnerabilities are disclosed. Vulnerabilities are an inevitable part of building software, whether they originate from design flaws or code bugs. Teams should have plans in place for damage control after the fact, but the more important piece that some teams overlook is the possibility of mitigation and prevention by acting proactively. Not every vulnerability can be foreseen and prevented, but many can with increased communication.

The Sans Institute recommends, when using open source software at an enterprise, a proactive approach with certain software vendors sponsoring open source projects, in order to perform "rigorous security evaluation and certification". It is also recommended to consider in-house expertise and support options for an open source product before use [9]. To catch bugs and flaws early on, code should be reviewed before merging into the master branch of any project. More specifically, an explicit code security review should be routine to check for missing bounds checks, missing sanitization of input, and other potential exploitation points. The review should be done by someone with experience in security and penetration testing. Overall, an increase in funding, attention, and sponsorship would be beneficial to improving the situation of open source security, such that it is suitable for business-critical applications [9].

Conclusion

In many cases, increased communication and coordination throughout the open source community can mitigate incidents before they happen. Careful monitoring of critical open source projects like OpenSSL (which depends largely on donations) is important, and code reviews should be regular events, especially before major changes are merged into the main production branch. Communication between developers, as well as communication between developers and enterprises, as well as between developers and users, is all critical to achieving security. Security is not achieved by one

person working alone, but rather by the entire community working together to educate, learn and apply security practices on a regular basis.

When vulnerabilities are found in open source software, there should be a coordinated effort to educate all users and those affected. This effort should be inclusive and take into account the varying levels of security knowledge and experience of its audience. When enterprises and independent open source developers work together, the open source community becomes stronger and more cohesive. More vulnerabilities are identified sooner rather than later, resulting in safer software and greater user security, lessening the effects of vulnerabilities like Heartbleed and granting software users the security and assurance they need in order to utilize open source software with confidence.

Supporting Material:

Source code on Github: `https://github.com/m-iori/HeartbleedVisualDemo`
Live version: `https://www.eecs.tufts.edu/~miori01/heartbleedVisualDemo.html`

This visual "demo" is meant to serve as a simulation of how the Heartbleed vulnerability works for an attacker. The demo allows a visitor to this site to compare a vulnerable "server" to a patched "server" (the HTTPS requests and responses are simulated in JavaScript). Decision makers without technical knowledge tend to be focused on the consequences, not always on preventing incidents before they happen, and vulnerabilities only receive focus after the attack has already occurred. What this demo demonstrates is a clear cause and effect communicated to the viewer ("if I dont update my OpenSSL instance, then sensitive data on my server could be open for any attacker to see"). In comparison, the patched "server" is safe from harm, even if the same exploitation is used against it. All of this should convince viewers that updating software is worth the time spent, without using too much technical jargon and allowing viewers to "roleplay" as an attacker and see how simple a successful attack can be.

Bibliography

[1] Synopsys, "Heartbleed bug." http://heartbleed.com. Accessed: 2018-04-30.

[2] "Openssl." https://www.openssl.org. Accessed: 2018-04-30.

[3] J. Walsh, "Free can make you bleed." https://blog.ssh.com/free-can-make-you-bleed. Accessed: 2018-04-30.

[4] "Rfc 6520 - transport layer security (tls) and datagram transport layer security (dtls) heartbeat extension." https://tools.ietf.org/html/rfc6520. Accessed: 2018-04-30.

[5] PewResearchCenter, "Heartbleed's impact." https://assets.pewresearch.org/wp-content/uploads/sites/14/2014/04/PIP_Heartbleed-impact_043014.pdf. Accessed: 2018-04-30.

[6] A. Goldstein, "Top 10 open source security vulnerabilities in december." https://resources.whitesourcesoftware.com/blog-whitesource/top-5-open-source-security-vulnerabilities-in-december. Accessed: 2018-04-30.

[7] G. Avner, "Top 10 security vulnerabilities of 2017." https://resources.whitesourcesoftware.com/blog-whitesource/top-10-security-vulnerabilities-of-2017. Accessed: 2018-04-30.

[8] "Osvdb — everything is vulnerable." https://blog.osvdb.org. Accessed: 2018-04-30.

[9] S. R. Vadalasetty, "Security concerns in using open source software for enterprise requirements." https://www.sans.org/reading-room/whitepapers/awareness/security-concerns-open-source-software-enterprise-requirements-1305. Accessed: 2018-04-30.