How the Heartbleed bug reveals a flaw in online security

April 11 2014, by Robert Merkel

Does Heartbleed expose flaws in the way some security-critical software is developed? Credit: Flickr/Kaleenxian, CC BY-NC-ND

The Heartbleed bug that's potentially exposed the personal and financial data of millions of people stored online has also exposed a hole in the way some security software is developed and used.

The bug is in an extremely widespread piece of software called OpenSSL. OpenSSL allows programmers to write systems that send sensitive data such as financial or medical information over the internet, with confidence that anybody "listening in" will only get indecipherable gibberish.

It also provides a way to prove that a message came from a particular organisation's computer, so that you can be confident you're sending your credit card details to Amazon or Apple rather than a criminal.

How was OpenSSL developed?

OpenSSL is not the only tool that provides these facilities, but it is by far the most common, due to its free availability and long history.

OpenSSL dates from the late 1990s, and like many other crucial pieces of internet software, is developed by a loosely-organised global bunch of hobbyists, students and volunteers.

It is made available as open source software for anyone to use for free on very liberal terms. Most of the world's internet servers – and every Android smartphone – use a great deal of software developed in this manner, though many such developer teams include paid professionals from companies who use the software.

The Heartbleed bug

On New Year's Eve 2011, German researcher and OpenSSL contributor Robin Seggelmann added code implementing a new feature called "heartbeats".

The idea was straightforward: if a connection between two computers stays silent for too long, it is disconnected, so periodic "heartbeat" messages can keep the connection going.

As well as a simple "I'm here", messages contain a arbitrary "payload" which is sent back and forth, a little like this:

Computer 1: "Hi, I'm still here, the payload is 5 characters long and is '12345'."

Computer 2: "Hi, great, you're still there, and your payload was 5 characters long and was '12345'."

Unfortunately, Seggelmann's code didn't check that the payload was of the indicated length, so a malicious request could request more data than was in the payload:

Computer 1: "Hi, I'm still here, the payload is 50,000 characters long and is '12345'."

Computer 2 would then send back a message with a payload of the requested length, the first characters of which would be the 12345 sent. The rest would be whatever happened to be in the computer's memory next to the payload.

The exact contents sent back varied between systems and over time. But as well as information such as user passwords or private data, it could contain something called the private master key.

With access to this key, an "attacker" can electronically impersonate the organisation who rightfully owns the key, and unscramble all the private messages sent to that organisation – including old ones, if they've kept the previously unreadable scrambled versions.

Criminals could, for instance, steal the key of a major bank and then electronically impersonate it. It's a potential field day for spies, too.

Discovery and consequences

The buggy code was incorporated into a June 2012 release of OpenSSL that was widely adopted, and there it stayed until discovered virtually simultaneously by Google's security team, and Codenomicon, an internet security company.

Before informing the public, they informed the OpenSSL developers, who fixed the bug by adding the missing checks.

At this moment, there is no evidence that anybody has maliciously exploited the bug but system administrators have acted both to prevent exploitation, and reduce the consequences if it has already been.

The fix is simple. The task of getting it deployed to the millions of systems using OpenSSL is not.

System administrators across the world have been furiously installing the fix on millions of computers. They're also scrambling to generate new master keys.

For most end users, the biggest nuisance will come when administrators request password changes.

Most users have multiple internet accounts; many of these will be affected by the Heartbleed bug and their administrators will request their users to change passwords in case they have been stolen.

In addition, many embedded computers in devices such as home network routers may be vulnerable, and updating these is a time-consuming manual task.

Even if there hasn't been any malicious exploitation of the bug, the costs of people's time will likely run into the hundreds of millions of dollars.

A tiny mistake but a major headache

Contrary to a variety of conspiracy theories, the simplest and most likely explanation for the bug is an accidental mistake. Seggelmann denies doing anything deliberately wrong.

Mistakes of the type that caused Heartbleed are have led to security problems since the 1970s. OpenSSL is written in a programming language called C, which also dates from the early 1970s. C is renowned for its speed and flexibility, but the trade-off is that it places all responsibility on programmers to avoid making precisely this kind of mistake.

There are currently two broad streams of thought in the technical community about how to reduce the likelihood of such mistakes:

1. use technical measures, such as alternative programming languages, that make this type of error less likely
2. tighten up the process for making changes to OpenSSL, so that they are subject to much more extensive expert scrutiny before incorporation.

Dealing with risk

My view is that while both of these points have merit, underlying both is that the Heartbleed bug represents a massive failure of risk analysis.

It's hard to be too critical of those of who volunteer to build such a useful tool but OpenSSL's design prioritises performance over security, which probably no longer makes sense.

But the bigger failure in risk analysis lies with the organisations who use OpenSSL and other software like it. The development team, language choices and development process of the OpenSSL project are laid bare, in public, for anyone who cares to find out.

The consequences of a serious security flaw in the project are equally obvious. But a huge array of businesses, including very large IT businesses depending on OpenSSL with the resources to act, did not take any steps in advance to mitigate the losses.

They could have chosen to fund a replacement using more secure technologies, and they could have chosen to fund better auditing and testing of OpenSSL so that bugs such as this are caught before deployment.

They didn't do either, so they – and now we – wear the consequences, which likely far exceed the costs of mitigation.

And while you shake your head at the IT geeks, I leave you with a question – how are you identifying and managing the risks that your own organisation faces?

This story is published courtesy of The Conversation *(under Creative Commons-Attribution/No derivatives).*

Provided by The Conversation

Citation: How the Heartbleed bug reveals a flaw in online security (2014, April 11) retrieved 2 May 2024 from https://phys.org/news/2014-04-heartbleed-bug-reveals-flaw-online.html