Security of OpenSSL Tool

John Phillips *johphill@mines.edu*
2nd Abhaya Shrestha *ashrestha@mines.edu*
3rd Joe Granmoe *jgranmoe@mines.edu*
4th Patrick Curran *pcurran@mines.edu*
5th Collin McDade *collinmcdade@mines.edu*
6th Noor Malik *nmalik@mines.edu*

***Abstract*—There have been small but catastrophic modifications to openssl in the past [1] [2] that drastically reduced the potential number of keys that could be generated during key generation in the debian distribution. We'd like to explore one such instance, and see if a similar vulnerability that is easy to exploit could still be introduced into the codebase with a small modification to the base source of OpenSSL.**

***Index Terms*—OpenSSL, security vulnerability, debian**

I. Introduction

A. Background

In 2008 a debian developer made a patch to openssl to fix valgrind warnings [2] [3]. Parts of openssl were deliberately using uninitialized memory to add entropy for key generation in a nonessential way, and this caused client programs that linked to openssl to generate error/warnings when developers tried to validate programs with valgrind. But there was also another function call that looked very similar elsewhere in the code which was critical to key generation. The debian developer that worked on the bug (as seen in [2]) mistakenly thought that *both* function calls should be removed when only one should have. With both gone, the function `ssleay_rand_bytes`, which was used to return random bytes to client code, suddenly only relied on the current process PID to return randomness. This meant that now there were only 2^{15} possible values returned into the random buffer, so that programs like `ssh-keygen` that used that function to generate random bytes for keys, were suddenly extremely weak.

B. Motivation

We'd like to dig into the current and past source code of openssl, understand the first exploit referenced in [1] in the old code, and see if a similar or identical exploit is still easy to introduce and reproduce. We'd also like to get a sense of how hard it would be for a malicious actor to introduce another similar exploit into the code – for a similar mistake to happen in 2022. This would involve researching the past and current workflows for making potentially security-critical patches to both debian and the OpenSSL project at a minimum, and potentially other popular linux distributions as well.

Open questions are, for example,

- What does it take to be a developer on one of these projects?
- How are patches reviewed?
- How formal is the patch review process, and have potential bugs already been successfully caught and removed before introduction to distributed code in the past?

C. Goal

We'd like to reproduce the 2008 bug from the git commit of the version referenced in [2]. Then, we'd like to see if we can make a similar modification to the current code and demonstrate a similar working exploit (i.e. we'd like to derive a key and decrypt a file for example), and show that it could be very simple for a disgruntled employee or malicious actor to introduce a small patch to OpenSSL in a downstream distribution (or potentially to the project itself) that could make it very vulnerable, or alternatively demonstrate that the codebase has improved and is much harder to accidentally exploit this way. We will also investigate the current process in the debian distribution and the OpenSSL project for adding patches, and report on what we find.

II. Idea and Approach

First, we'd like to compile and demonstrate the vulnerability as it existed in 2008. Most likely this would involve creating a VM image with an old debian version, compiling an OpenSSL version with the patch specified in the debian bug report [3], linking `ssh-keygen` to the library (which will be a re-creation of what was distributed in 2008), and write a simple program to guess the generated key, since there are only 2^{15} possibilities.

Then, we'd like to fork a current openssl version, and see if we can make a small modification to the code that reduces the possible number of keys in a way that recreates the 2008 vulnerability. This modification should be similar to the exploit in [1] in that it is a small number of lines of code that decreases entropy for key generation, making it easier to guess the private key generated by openssl.

Then we would create another small program that can guess the key generated by the new hacked version, also in a reasonable amount of time since it knows that the keyspace is now much smaller. We could then for example forge a signature or decrypt a file encrypted with the public key.

Next, we will investigate their upstream process on how they detect vulnerability on their code (do they have an automated, rigorous code review process etc.). This will show us that despite the possbility of developer error, how likely are we to see the vulnerability exposed to production environments.

III. Expected results

First, we'd have a runnable 'exploit' that works on the old debian-modified openssl. This should demonstrate that it only takes a small change to openssl that could be made in any number of places to decrease the potential keyspace for private keys.

We also would either demonstrate that that kind of vulnerability is still easy to introduce, potentially accidentally, or that the codebase and processes around introducing fixes have been fixed in debian and/or the openssl project.

As far as the upstream process is concerned, if we find their process is rigorous, then OpenSSL is fairly secure for its most recent version (with the given computing power at the moment of the experiment). Otherwise, we will recommend some other alternaitve to OpenSSL.

IV. Conclusion

We'd like to create an easy-to-exploit vulnerability in the current openssl source that is similar to the one inadvertently introduced into the debian source code in 2008 as documented in [1]. If it's still possible to make such a change, this raises concerns about the security of an open source security-critical system like openssl. If we are not able to find easy-to-exploit vulnerability and our inverstigation of their contribution process shows rigorous code reviews and automation in place, we will conclude that current version of OpenSSL (with the given computing power) is a secure library for encryption. Otherwise, we will recommend other libraries to use as an alternative to openssl.

References

[1] https://isotoma.com/blog/2008/05/14/debians-openssl-disaster/
[2] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=363516
[3] https://research.swtch.com/openssl