Fuzzing Project report June 2015

Hanno Bck

June 2015

1 Introduction

In November 2014 I started the Fuzzing Project as an attempt to increase the effort in fuzzing free software applications. In May 2015 the Linux Foundation's Core Infrastructure Initiative decided to fund my work. In this report I'll give an overview of my work in the past months. As I already worked on this before the funding this report will in parts include work that has been done before.

I am usually using the tool american fuzzy lop, developed by Michal Zalewski from the Google security team [1]. I use this in combination with Address Sanitizer, which is a compiler feature in gcc and clang that allows to catch many memory access errors that would otherwise not be noticed.

American fuzzy lop is currently only able to fuzz file-based inputs, therefore network and USB fuzzing are out of scope for this tool.

1.1 Dedicated Server

Up until recently all my fuzzing work was done on my private laptop. The CII funding allowed me to rent a dedicated server for my work. I try to make sure that this server is always running fuzzing jobs. The dedicated server runs Ubuntu, however most of the fuzzing work is done on a chroot'ed Gentoo system.

Currently I feel that this server is sufficient to support my work and I don't need further hardware resources. Based on my experience it rarely makes sense to run fuzzing jobs extremely long, I usually stop them after one day.

1.2 OpenSSL

Starting in March I have systematically tested all possible file input vectors in OpenSSL. This led to the discovery of a parsing issue on PSS parameters while fuzzing OCSP responses. This issue was co-discovered by Brian Carpenter and fixed in OpenSSL 1.0.2a.

I discovered one further security issues that was disclosed to the OpenSSL-developers, it was also co-discovered by a Google developer [2]. It is fixed in

[1] `http://lcamtuf.coredump.cx/afl/`

[2] `https://blog.fuzzing-project.org/15-Out-of-bounds-read-in-OpenSSL-function-X509_cmp_time-CVE-2015-1789-and-other-minor-issues.html`

the June releases of OpenSSL (1.0.2b/c). I further reported a number of non-security issues in the ASN1 definition parser [3]. I also submitted some smaller patches to fix undefined behaviour [4], uninitialized memory [5], memory leak issues [6] and a typo [7].

Besides the file fuzzing I also wanted to be able to fuzz the TLS handshake of OpenSSL. Fortunately the OpenSSL API allows to create a file handshake without doing any real networking by passing buffers. I therefore created a small test application that took raw TLS packets from the command line and was able to inject them into the handshake. The code is published on Github [8].

I didn't find any new issues with this method, however I was able to show that fuzzing the TLS handshake on an old version of OpenSSL (1.0.1f) with Address Sanitizer and american fuzzy lop would've been sufficient to detect the Heartbleed bug [9].

1.3 Network Fuzzing, OpenSSH

While american fuzzy lop is a very powerful tool the lack of network fuzzing is a significant downside. I therefore tried to find ways how afl could be applied to network inputs. The method used for OpenSSL only works in rare cases when the API of a library allows to pass network input through memory buffers. Also this method requires manual work for every application and is therefore not easily scalable.

I tried to create a library that could be preloaded and would intercept libc's networking functions, simulate a network connection and pass data read from a file. However this turned out to be difficult, as the libc socket API is quite diverse and overlaps with the API for file access. My current implementation only works in rare cases, however I was still able to use it to find an out of bounds access in OpenSSH [10].

I intend to release this library soon under a free software license, however right now it is not really in a usable state. I learned later that there is a tool called preeny [11] which follows a similar approach. However based on my experience preeny suffers from similar problems and only works in rare cases. One user on the american fuzzy lop users mailing list mentioned he is also working on a similar tool.

[3] https://github.com/openssl/openssl/commit/c4137b5e828d8fab0b244defb79257619dad8fc7 https://github.com/openssl/openssl/commit/111b60bea01d234b5873488c19ff2b9c5d4d58e9 https://rt.openssl.org/Ticket/Display.html?id=3800&user=guest&pass=guest

[4] https://rt.openssl.org/Ticket/Display.html?id=3816&user=guest&pass=guest

[5] https://github.com/openssl/openssl/commit/539ed89f686866b82a9ec9a4c3b112878d29cd73

[6] https://rt.openssl.org/Ticket/Display.html?id=3861&user=guest&pass=guest

[7] https://rt.openssl.org/Ticket/Display.html?id=3796&user=guest&pass=guest

[8] https://github.com/hannob/selftls

[9] https://blog.hboeck.de/archives/868-How-Heartbleed-couldve-been-found.html

[10] http://www.openwall.com/lists/oss-security/2015/05/16/3

[11] https://github.com/zardus/preeny

1.4 Full system with Address Sanitizer

Based on Gentoo Linux I've tried to create a full Linux system that is using Address Sanitizer. This could be used as a basis for people needing a very strong security protection for a system or alternatively as a debugging tool to find memory access violation bugs in software.

A number of key packages fail during compilation, I already submitted a fix for Python 2 to allow compilation with address sanitizer. It was incorporated in the latest 2.7.10 Python release [12].

1.5 Dovecot denial of service

During the OpenSSL handshake test I discovered that in certain situations my test application would crash. However this was not due to a bug in OpenSSL, it was a missing check for errors in my application. In certain states OpenSSL would crash if one tries to continue a handshake after a connection failure.

I later found out that a similar bug was present in the Dovecot pop3/imap server. If configured to reject SSLv3 connections Dovecot would crash on connection attempts with that version. After a chat conversation with Aaron Zauner, who was scanning the TLS configuration of mail hosts as part of a research project, I learned that he had problems that their scans would crash Dovecot installations. I therefore investigated the issue further and reported the issue to the Dovecot developers [13]. It was fixed in Dovecot 2.2.17.

1.6 Further fuzzing

Appart from these efforts I mainly tried to scale my fuzzing work and fuzz more applications. Notable findings where memory access bugs in Wireshark [14], Curl [15], Libtasn1 [16], SQLite [17], GnuTLS and Berkeley DB.

I also reported memory access bugs in libarchive, libcdio, sed, arc, ctags, make, chmlib, exiv2, gnupg, nghttp2, desktop-file-utils, radare2, dc/bc, webalizer, netpbm, dcraw, splint, patch, indent, podofo, cabextract, sharutils/uudecode, fontforge, libidn.

[12] https://bugs.python.org/issue24061

[13] https://blog.fuzzing-project.org/12-Denial-of-Service-in-Dovecot-and-unexpected-crashes-in-OpenSSL-TFPA-
html

[14] hhttps://blog.fuzzing-project.org/11-Read-heap-overflow-invalid-memory-access-in-Wireshark-TFPA-0072015.
html

[15] https://blog.fuzzing-project.org/8-Why-it-can-make-sense-to-fuzz-config-files-two-out-of-bounds-vulnerab
html

[16] https://blog.fuzzing-project.org/9-Heap-overflow-invalid-read-in-Libtasn1-before-4.
5-TFPA-0052015.html

[17] https://blog.fuzzing-project.org/10-Two-invalid-read-errors-heap-overflows-in-SQLite-TFPA-0062015.
html

1.7 Vulnerability terminology

When I made the issue in the OpenSSH handshake public this caused probably a bit more attention than it deserved. I was criticised by members of the IT security community for publicly calling an out of bounds read access a "Heap overflow" [18].

This highlights an issue with an inconsistent terminology. The term "Buffer overflow" (and, in accordance to that the more specific words "Heap overflow" and "Stack overflow") is sometimes used to describe only out of bounds write access issues, while sometimes it's also used to describe out of bounds reads. MITRE defines buffer overflows in CWE-120 and the description is clear that it is only intended to cover out of bounds writes [19]. Address Sanitizer will report all out of bounds access issues - whether read or write - as overflows.

The vast majority of the bugs I report due to fuzzing are heap out of bounds reads. In result of the criticism I received I decided to always name these issues "out of bounds read" in the future to avoid confusions.

I'd also like to point out that with these issues it is often hard to qualify the severity. While the vast majority of them are probably not exploitable in any meaningful way there are situations where a heap out of bounds read can result in catastrophic bugs (e. g. Heartbleed). The assignment of CVE identifiers to these issues is inconsistent. However I think there is no controversy that an out of bounds read should always be considered a bug and should be fixed.

1.8 Notes on reporting bugs

The majority of my work is not the technical process of fuzzing applications, it's the reporting of bugs. Free software projects have very different ways of handling bug reports. Some have bug trackers, some accept reports on mailing lists, some can only be directly contacted via E-Mail.

The results to my bug reports where mixed, but most of the time I had the feeling that my reports were appreciated by the respective upstream developers and fixed quickly. However what this work sometimes painfully shows is that there is a large number of free software applications that are widely used, but they are unmaintained. Sometimes it's impossible to contact an upstream developer, sometimes I learn that they are no longer interested in developing their application and sometimes I don't get any reaction.

In cases of abandoned projects if possible I still try to submit reports in a way that they can be found by others. In case there is a public bug tracker, forum or mailing list I post the reports there, with the intent that people interested in reviving development or forking a project can find them.

[18] `https://twitter.com/i0n1c/status/599575653361577984`
[19] `https://cwe.mitre.org/data/definitions/120.html`