Lecture 11 : Fuzzing

Alexandre Bartel

2019

Previously...

Previously... in Lecture 1 (Introduction)

- Software Development Life-cycle
- Vulnerability Life-cycle
- Vulnerability Disclosure

Previously... in Lecture 2 (Buffer Overflow)

- A buffer on the stack
- Return address on the stack
- Overwrite return address
- Jump to shellcode on the stack

Previously... in Lecture 3 (ROP)

- NX bit (stack non-executable)
- Gadgets in already loaded code
- Chain gadgets (addresses of gadgets and data on the stack)
- Only data on the stack

Previously... in Lecture 4 (ASLR)

- Randomize code segment at program start
- Breaks gadget chains
- Bypass with information leak (e.g, vulnerability)

Previously... in Lecture 6 (CFI)

- Mecanism to allow only "intended" paths
- Binary instrumentation to add IDs
- Indirect jumps, call, returns check if ID of "destination" is correct
- Pure software implementation have 20% overhead

Previously... in Lecture 7 (Heap-Overflow)

- How a heap-overflow can be attacked depends on the heap management implementation
- The "unlink" attack present in early versions of glibc provides a "write anywhere" primitive to the attacker
- Recent implementations performs more check to prevent "unlink" based attacks

Previously... in Lecture 8 (Type Confusion)

- What is it? Manipulation of an object through another object.
- Consequences? Undefined behavior, hijack control flow.
- Why it works? No verification at runtime (otherwise runtime and/or memory overhead).

Previously... in Lecture 9 (SQL Injection)

- Code injection attacks enables bypass of authorization checks and/or execution of arbitrary code on the server
- Consequences: attacker gets access to privileged environment and/or can dump databases
- Protection include sanitization of the input and/or well defining what is code and what is data

Previously... in Lecture 10 (Confused Deputy)

- Confused Deputy Attack are a type of privilege escalation vulnerability
- Attacker exploits the associated vulnerability (misconfiguration, logic error) to have more privilege
- Protection include sanitization of the input and/or changing the configuration and/or patching the code logic

Fuzzing

Fuzzing

- Oxford English Dictionary: ”Perhaps imitative of the action of blowing away light particles.”
- Technique to test software by using random data as input

Software Input

- File
- Network connection
- USB device (keyboard, mouse)
- Environement variable
- ...

Random Data

- Input could be totally random (in size and values)
- But, this technique has limitations
- Often, code has checks on the input bytes (random inputs often will be rejected at this stage)
- The generated input should take these checks into account to test the code at a deeper level

Behavior

- Any software should "normally" quit on an invalid input
- Most do not and crash (C/C++)
- This crash means the input has be handled incorrectly
- From a security perspective, the analyst has to manually analyze the crash to see if the bug is a security vulnerability
- In case of security vulnerability, what can be exploited (return address on the stack, ...) ?

Target Applications

- Any software could be fuzzed
- We limit our presentation to the C/C++ languages
- But, there are fuzzers for Java and other languages

1990: first publication on the topic

- Miller, Barton P., Louis Fredriksen, and Bryan So. "An empirical study of the reliability of UNIX utilities." Communications of the ACM 33.12 (1990): 32-44.
- Available here: ftp://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf
- Input = random characters
- Target = UNIX utilities (awk, cat, diff, ...) note: binaries are untouched
- Result = 25 - 33 % of utilities on any version of UNIX crashed
- Cause = no valid bounds for arry referenc; input fields should be bounded; check system call return values; pointers should be checked before usage; ...

State of the art: AFL [1]

- AFL: American Fuzzy Loop
- Instrument binaries to check code coverage
- Input gives better coverage $\rightarrow$ kept for future mutation
- Mutation operators:
  - truncate input
  - switch bits
  - change byte value
  - arithmetic operation
  - add bloc of bytes
  - shift bytes

[1] Available here: http://lcamtuf.coredump.cx/afl/

AFL: Evolutionary Fuzzing

- AFL receives feedback from the program under test about the property to maximize (code coverage)
- Feedback gives information about the coverage (did the new input did better code coverage?)
- Input improving code coverage is changed (is "evolved" through "mutation" for instance)
- Using thi approach AFL learns little by little the input structure

AFL: Some vulnerabilities it helped discovered

IJG jpeg 1 libjpeg-turbo 1 2 libpng 1 libtiff 1 2 3 4 5 mozjpeg 1 PHP 1 2 3 4 5 6 7 8 Mozilla Firefox 1 2 3 4 Internet Explorer 1 2 3 4 Apple Safari 1 Adobe Flash / PCRE 1 2 3 4 5 6 7 sqlite 1 2 3 4... OpenSSL 1 2 3 4 5 6 7 LibreOffice 1 2 3 4 poppler 1 2... freetype 1 2 GnuTLS 1 GnuPG 1 2 3 4 OpenSSH 1 2 3 4 5 PuTTY 1 2 ntpd 1 2 nginx 1 2 3 bash (post-Shellshock) 1 2 tcpdump 1 2 3 4 5 6 7 8 9 JavaScriptCore 1 2 3 4 pdfium 1 2 ffmpeg 1 2 3 4 5 libmatroska 1 libarchive 1 2 3 4 5 6 ... wireshark 1 2 3 ImageMagick 1 2 3 4 5 6 7 8 9 ... BIND 1 2 3 ... QEMU 1 2 lcms 1 Oracle BerkeleyDB 1 2 Android / libstagefright 1 2 iOS / ImageIO 1 FLAC audio library 1 2 libsndfile 1 2 3 4 less / lesspipe 1 2 3 strings (+ related tools) 1 2 3 4 5 6 7 file 1 2 3 4 dpkg 1 2 rcs 1 systemd-resolved 1 2 libyaml 1 Info-Zip unzip 1 2 libtasn1 1 2 ... OpenBSD pfctl 1 NetBSD bpf 1 man, mandoc 1 2 3 4 5 ... IDA Pro reported by authors clamav 1 2 3 4 5 6 libxml2 1 2 4 5 6 7 8 9 ... glibc 1 clang / llvm 1 2 3 4 5 6 7 8 ... nasm 1 2 ctags 1 mutt 1 procmail 1 fontconfig 1 pdksh 1 2 Qt 1 2... wavpack 1 2 3 4 redis / lua-cmsgpack 1 taglib 1 2 3 privoxy 1 2 3 perl 1 2 3 4 5 6 7... libxmp radare2 1 2 SleuthKit 1 fwknop reported by author X.Org 1 2 exifprobe 1 jhead ? capnproto 1 Xerces-C 1 2 3 metacam 1 djvulibre 1 exiv 1 2 Linux btrfs 1 2 3 4 6 7 8 Knot DNS 1 curl 1 2 3 wpa_supplicant 1 libde265 reported by author dnsmasq 1 libbpg (1) lame 1 2 3 4 5 6 libwmf 1 uudecode 1 MuPDF 1 2 3 4 imlib2 1 2 3 4 libraw 1 libbson 1 libsass 1 yara 1 2 3 4 W3C tidy-html5 1 VLC 1 2 FreeBSD syscons 1 2 3 John the Ripper 1 2 screen 1 2 3 tmux 1 2 mosh 1 UPX 1 indent 1 openjpeg 1 2 MMIX 1 OpenMPT 1 2 rxvt 1 2 dhcpcd 1 Mozilla NSS 1 Nettle 1 mbed TLS 1 Linux netlink 1 Linux ext4 1 Linux xfs 1 botan 1 expat 1 2 Adobe Reader 1 libav 1 libical 1 OpenBSD kernel 1 collectd 1 libidn 1 2 MatrixSSL 1 jasper 1 2 3 4 5 6 7 ... MaraDNS 1 w3m 1 2 3 4 Xen 1 OpenH232 1... irssi 1 2 3 cmark 1 OpenCV 1 Malheur 1 gstreamer 1... Tor 1 gdk-pixbuf 1 audiofile 1 2 3 4 5 6 ... zstd 1 lz4 1 stb 1 cJSON 1 libpcre 1 2 3 MySQL 1 gnulib 1 openexr 1 libmad 1 2 ettercap 1 lrzip 1 2 3 freetds 1... Asterisk 1 ytnef 1 2 3 4 ... raptor 1 mpg123 1 Apache httpd 1 exempi 1 2 libgmime 1 2 3 pev 1 2 3 4 Linux mem mgmt 1 sleuthkit 1 Mongoose OS 1 iOS kernel 1gg

Are fuzzers used in practice?

- They should be used more [1]
- Some companies run fuzzers on open-source projects [2]

[1] https://www.computerworld.com/article/2516829/pwn2own-winner-tells-apple–microsoft-to-find-their-own-bugs.html
(2010)

[2] https://www.eweek.com/cloud/google-fuzzing-service-uncovers-1k-bugs-in-open-source-projects (2017)

Fuzzing versus Manual Code analysis

- F: Automated process
- F: Generates input and detect crashes
- F: Tries to cover maximum code but not all code can be covered
- F: Thus, will not find all bugs
- F: Crashes might be hard to analyze
- F: Might take to long to generate correct input for the parser
- M: Can analyze any code
- M: Can look for patterns
- M: Can focus on code behind parsing

What to do after a Crash?

- Isolate the input which triggered a crash
- Launch the original program under gdb and give it the input
- Manually find out what is the problem

Timeout

- Some input can make the target program loop indefinitely
- Solution: give a timeout after which the fuzzer will switch to the next input

Practical Considerations: The file system

- A fuzzer typically writes several thousands of files in seconds
- This is not something hard-drives or SSD were designed for
- Solution: use a filesystem in memory

Practical Considerations: Reaching the code

- It might happen that you want to fuzz a particular piece of code (ex: file parser)
- However, this code might run after a slow initialization code (ex: several seconds for graphic engine initialization)
- In that case, you need to manually change the code to bypass the graphic initialization part (ex: removing it)
- You also want the program to stop after the file parser
- In that case, you update the code to quit the program after the parser

Conclusion

- A fuzzer tries to find bug in a target program by generating random input
- The generating of input can be totally random or guided by a property (ex: code coverage)
- A bug/problem is found when the program crashes or hangs
- A fuzzing technique is easy to setup, is automated but in most cases cannot anlyze the whole program

Question?

Projets

- Groups of 2
- Choosen topics:
  1. Patch for CVE-2018-20343 (Ricardo, Alex)
  2. Study and PoC for CVE-2013-0912 (Type confusion) (Adriano, Yurii, Ervin)
  3. Exploitation of a PoC type confusion in C++ (Ihor, Artem)
  4. Heart Bleed (Kendal, Romain)
- Deliverables: Presentation + Code (PoC)
  - 25 minutes per presentation (ex: 15 minutes presentation 10 demo)
  - 5 minutes for questions

Projet: Patch for CVE-2018-20343

- Understand CVE-2018-20343, a buffer overflow vulnerability
- You have to identify all instances of buffer overflow in the code (the code is not very big)
- You have to patch the vulnerable code

Projet: Study and PoC for CVE-2013-0912 (Chrome type confusion)

- Reproduce the SVG code for the exploit based on information you find on the internet
- You should create a VM with a distribution from 2013 and have the vulnerable version of Chrome
- Exploin all the steps of the attack from type confusion to arbitrary code execution.
- Demo: show effect of type confusion

Projet: Exploitation of a PoC type confusion in C++

- Write a proof-of-concept showing how to exploit a type confusion in C++ in a x86_64 architecture (latest debian)
- Demo: present exploitation and how to prevent it.

Projet: Heart Bleed

- Explain the vulnerability and how to exploit it.
- Demo: show that an attacker can extract private information.