Fuzzing in C/C++

Testing/Fuzzing and ASAN

ERNW
providing security.

#whoami

- Birk Kauer
  - Security {researcher, analyst, consultant} at ERNW
  - OSCP, OSCE, OSEE
- Interests
  - Application Security
  - Reverse Engineering
  - Penetration Testing

ERNW
providing security.

ERNW GmbH

- Vendor-independent
- Established 2001
- 60 employees, 40 FTE consultants
- Continuous growth in revenue/profits
  - No venture/equity capital, no external financial obligations of any kind
- Customers predominantly large/very large enterprises
  - Industry, telecommunications, finance

ERNW
providing security.

xmas.rip

Agenda

- Testing
- Fuzzer
  - Attack Surface
  - Dump Fuzzer
  - Grammatic Fuzzer
  - Evolutionary Fuzzer
  - Hybrid Fuzzer
- ASAN

Testing

- Testing does always reveal errors in your code
  - Ensures the code keeps doing what you think it should!
  - Supports Continuous Integration (CI)
- Test cases should aim for 100% Code Coverage
- Those test cases are perfect to start fuzzing

ERNW
providing security.

Why Fuzzing?

- A different team usually develops test cases at Software companies
  - They don´t share many insights into the code and develop test cases based on requirements -> This function has to do correct arithmetic multiplications
  - Test team develops cases based on assumptions of the code

- Fuzzing is usually done by security researches, not by code owners

ERNW
providing security.

Why Fuzzing?

- Fuzzers will make no assumptions
- Fuzzers can test 24/7
- Many bugs are not easy to find with code reviews
- Fuzzing results are recyclable

- Code Audits are hard on complex things
  - Binary Parsers
  - Token Parser (Lexer/yacc)
  - Parsers in general
- Fuzzers do often find low-medium complex bugs way faster compared to Code Audits

ERNW
providing security.

What is Fuzzing

- Way to discover bugs in Software by providing ”randomized” input
- Fuzzing can give a quick view on the overall robustness of the application

CWE Top25

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

AFL

Bug-o-rama

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]

Honggfuzz

Bug-o-rama

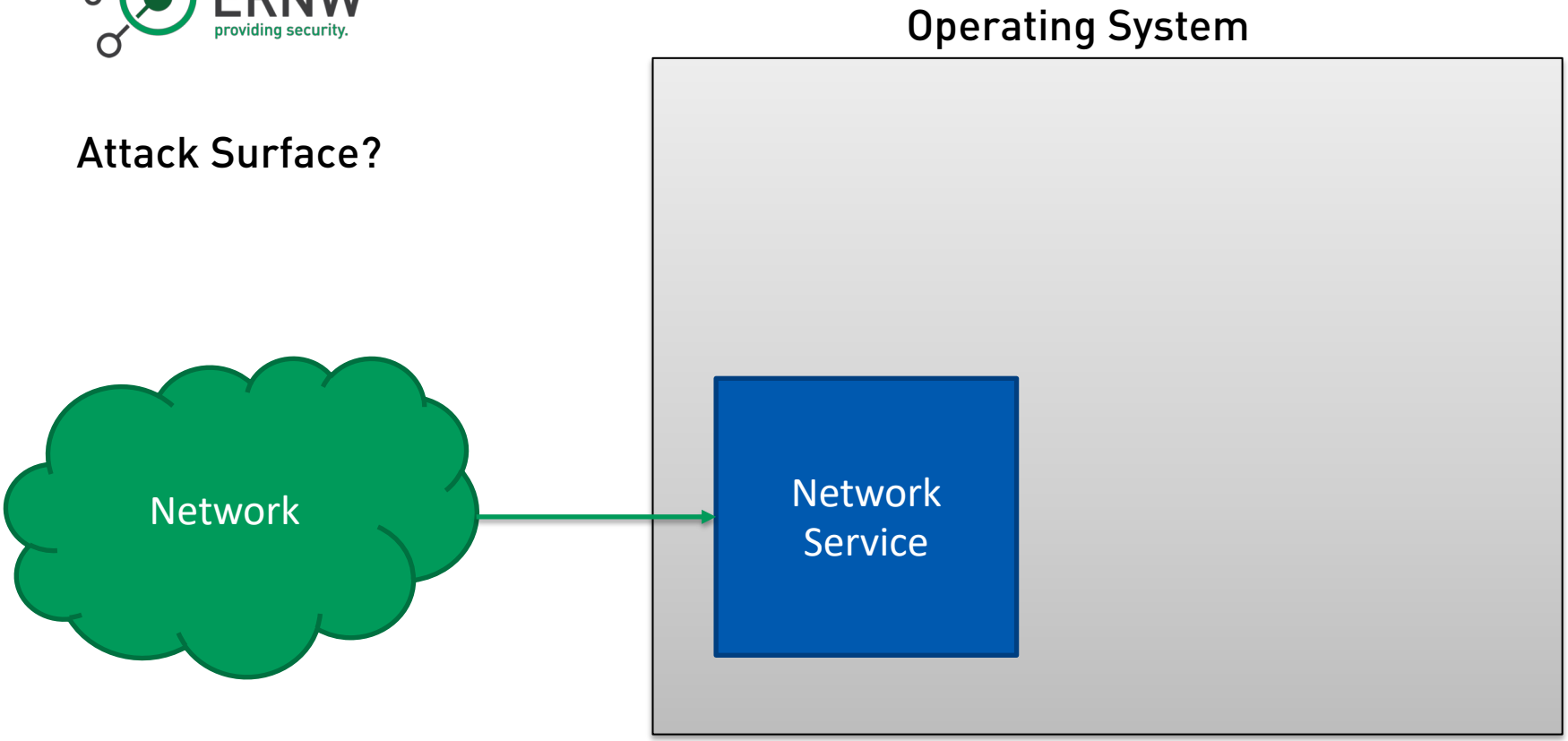
- Pre-auth remote crash in **OpenSSH**
- **Apache HTTPD**
  - Remote crash in **mod_http2** • CVE-2017-7659
  - Use-after-free in **mod_http2** • CVE-2017-9789
  - Memory leak in **mod_auth_digest** • CVE-2017-9788
  - Out of bound access • CVE-2018-1301
  - Write after free in HTTP/2 • CVE-2018-1302
  - Out of bound read • CVE-2018-1303
- Various **SSL** libs
  - Remote OOB read in **OpenSSL** • CVE-2015-1789
  - Remote Use-after-Free (potential RCE, rated as **critical**) in **OpenSSL** • CVE-2016-6309
  - Remote OOB write in **OpenSSL** • CVE-2016-7054
  - Remote OOB read in **OpenSSL** • CVE-2017-3731
  - Uninitialized mem use in **OpenSSL**
  - Crash in **LibreSSL**
  - Invalid free in **LibreSSL**
  - Uninitialized mem use in **BoringSSL**
- Adobe **Flash** memory corruption • CVE-2015-0316
- Multiple bugs in the **libtiff** library
- Multiple bugs in the **librsvg** library
- Multiple bugs in the **poppler** library
- Multiple exploitable bugs in **IDA-Pro**
- Remote DoS in **Crypto++** • CVE-2016-9939
- Programming language interpreters
  - **PHP/Python/Ruby**
  - PHP WDDX
  - PHP
  - Perl
- Double-free in **LibXMP**
- Heap buffer overflow in SAPCAR • CVE-2017-8852
- Crashes in **libbass**
- **FreeType 2**:
  - CVE-2010-2497
  - CVE-2010-2498
  - CVE-2010-2499

What to Fuzz?

- Attackers have different views on the system
- Don´t waste time Fuzzing “Trusted“ data

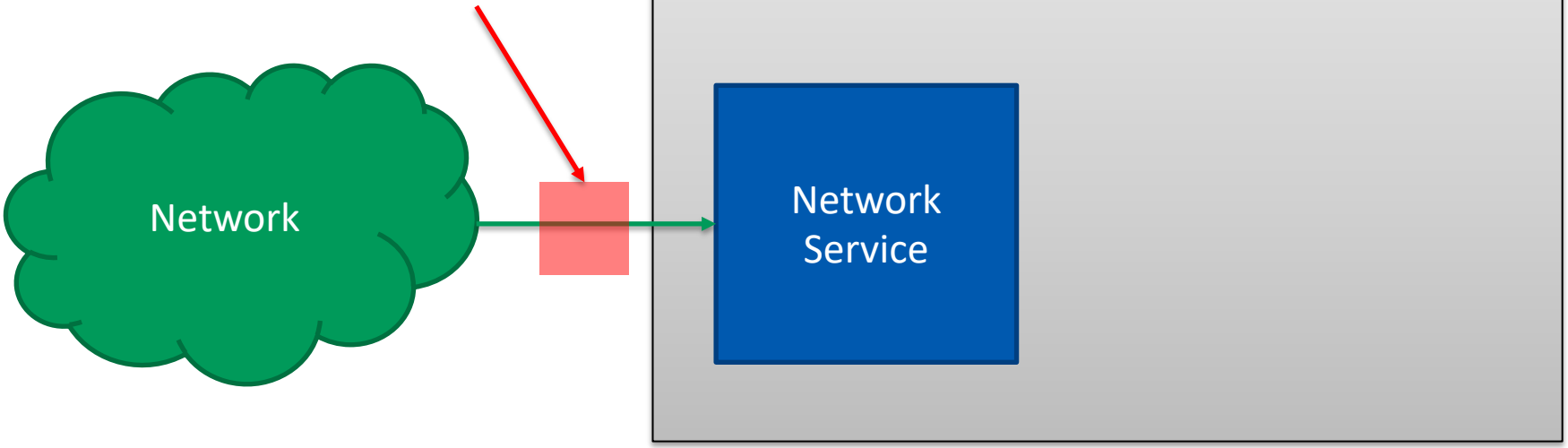
ERNW
providing security.

Attack Surface?



ERNW
providing security.

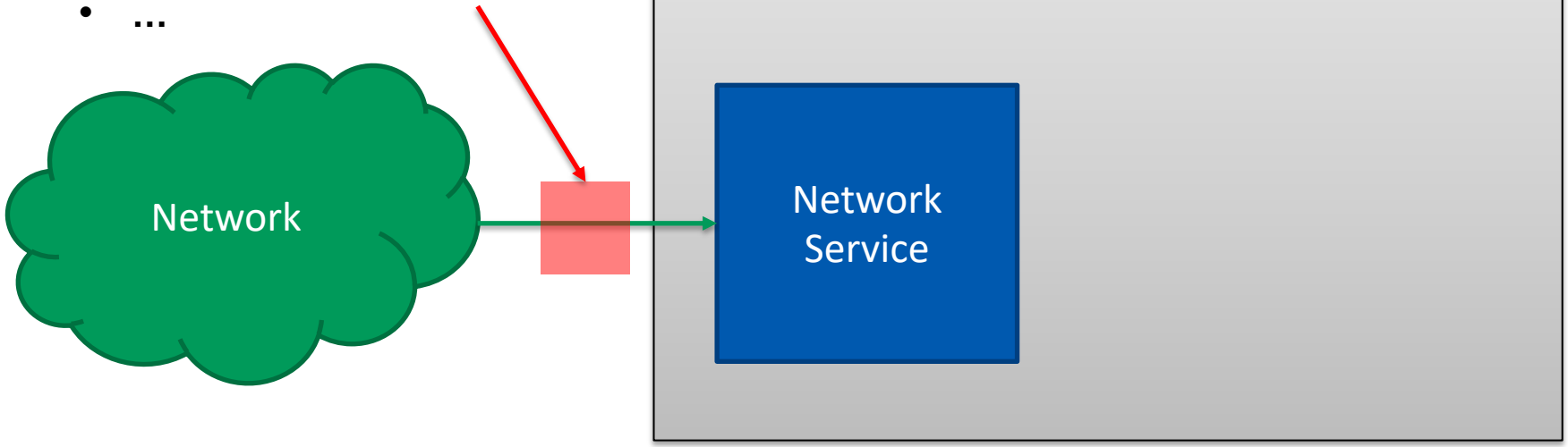
Problems?



ERNW
providing security.

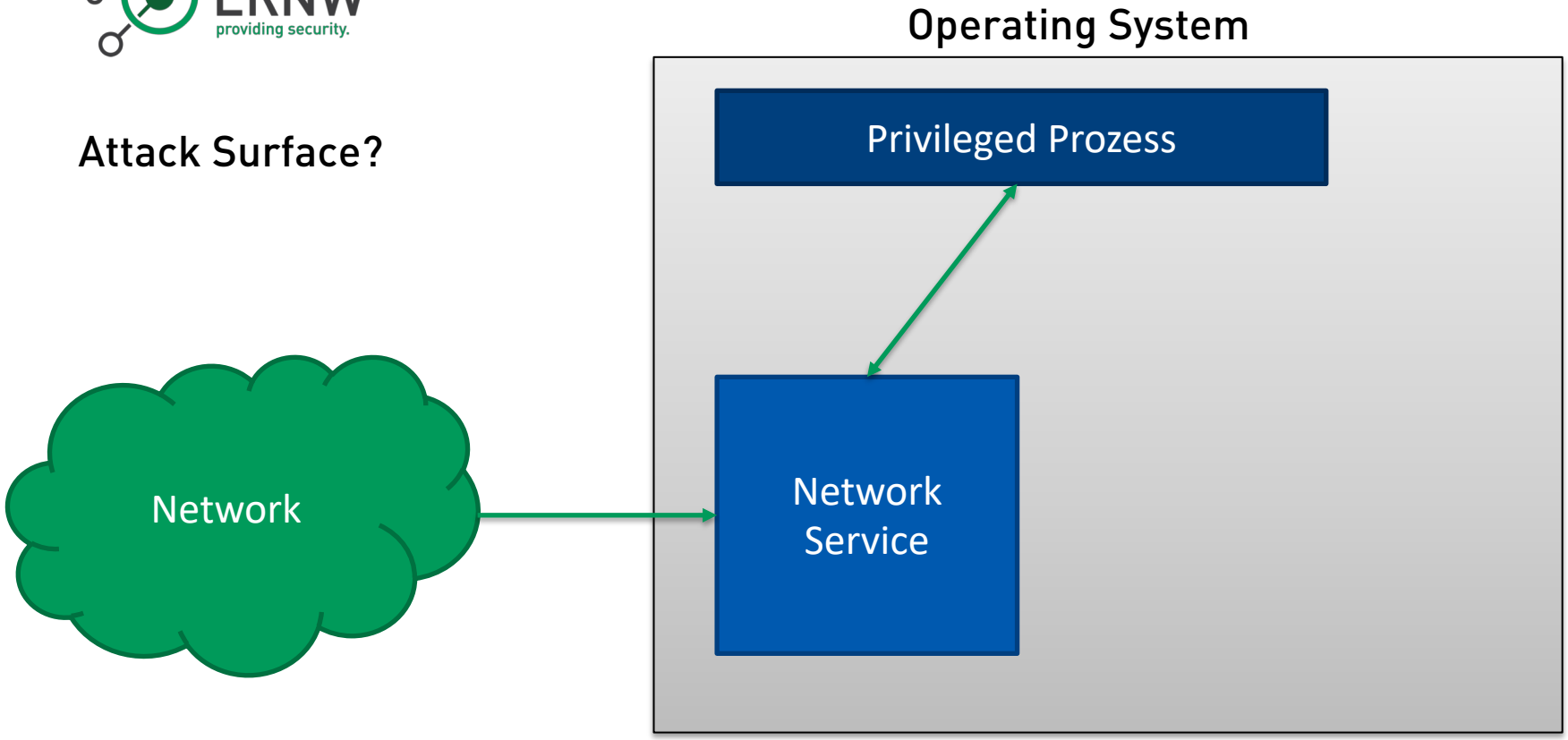
Parsing Problems

- Memory Corruptions
- Logic Bugs
- ...



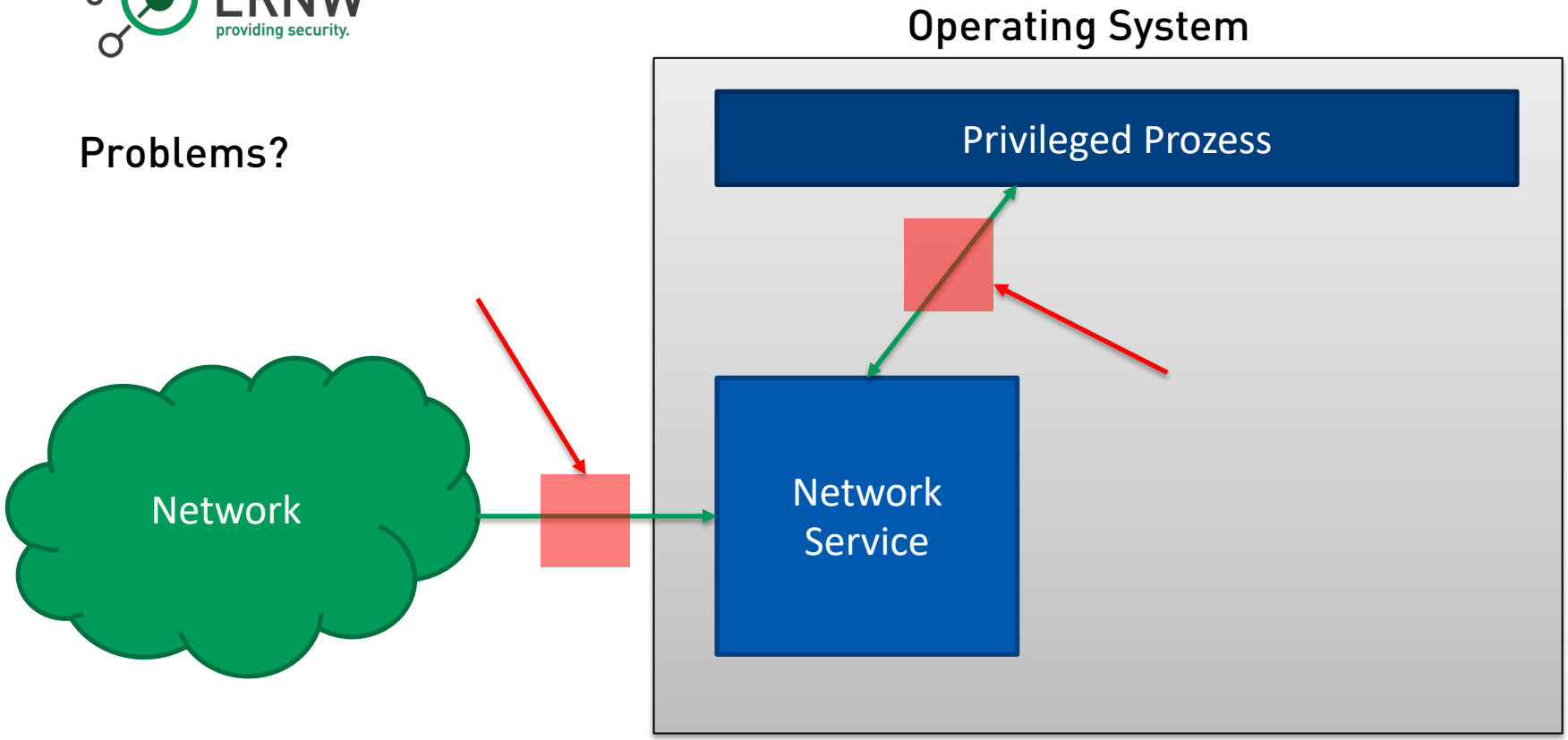
ERNW
providing security.

Attack Surface?



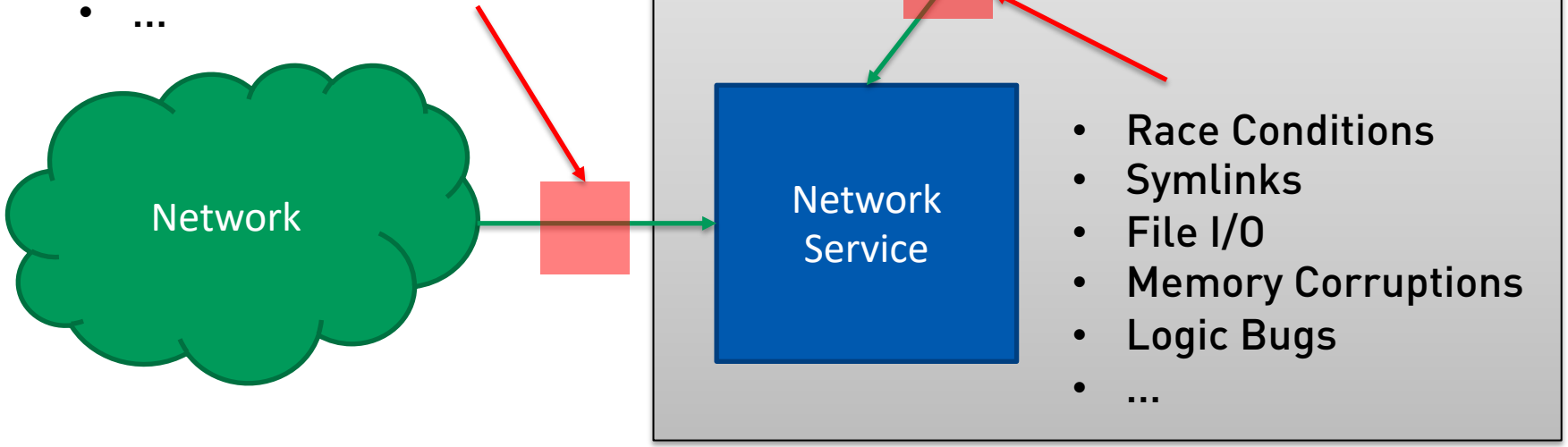
ERNW
providing security.

Problems?



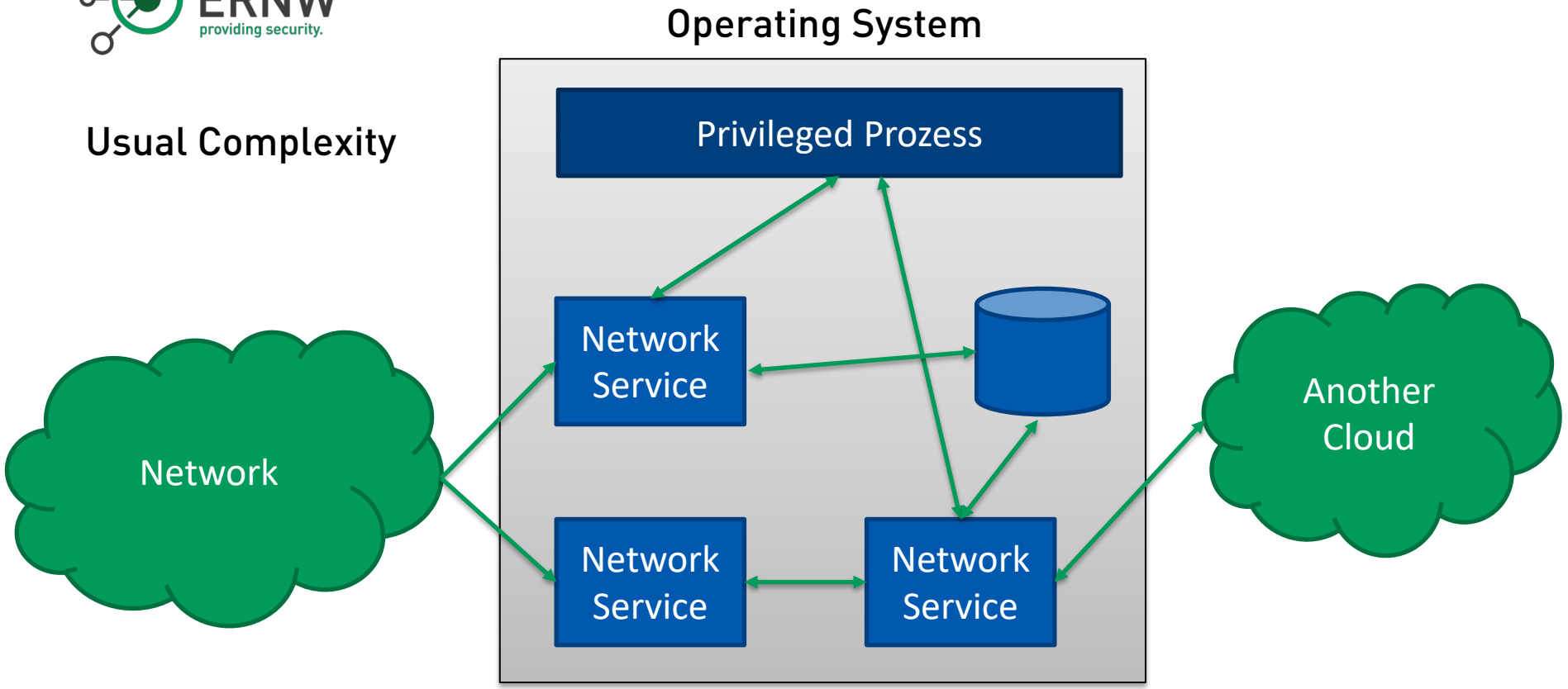
ERNW
providing security.

Parsing Problems
- Memory Corruptions
- Logic Bugs
- ...



ERNW
providing security.

Usual Complexity



ERNW
providing security.

Finding what to Fuzz (Attack Surface)

- Distinguish between every single component and connections
- Identify potential attack vectors
  - Identify which types vulnerabilities apply to this attack vector
  - Fuzz them!

- Trust can only exist in a 1:1 relationship

- Fuzz on your debug vuilds with and without sanitizers enabled!

Fuzzer History & Fuzzer Types

General Fuzzer Types

- Mutation Based
  - Changes already valid/known input formats
  - Flips / Changes / Adds / Subtracts the input
- Generation Based
  - Creates input from scratch
  - Require some level of intel of the input format
    - Technically random data would also count

ERNW
providing security.

Dumb Fuzzer – Generation Based

- Generating random input

```
for i in {1..100}; do dd if=/dev/urandom bs=100 count=1 |
nc myservice.dns 443; done
```

Dumb Fuzzer – Mutation Based

```python
#!/usr/bin/env python3
import random


seed_input = "This is a test"

def flip_random_character(s):
    """Returns s with a random bit flipped in a random position"""
    if s == "":
        return s

    pos = random.randint(0, len(s) - 1)
    c = s[pos]
    bit = 1 << random.randint(0, 6)
    new_c = chr(ord(c) ^ bit)
    # print("Flipping", bit, "in", repr(c) + ", giving", repr(new_c))
    return s[:pos] + new_c + s[pos + 1:]


for i in range(10):
    print(repr(flip_random_character(seed_input)))
```

```
'This i3 a test'
'This is a(test'
'This is a te3t'
'This is a$test'
'this is a test'
'This is a tesd'
'This is a0test'
'This(is a test'
'This!is a test'
'This is a tewt'
```

ERNW
providing security.

Smart “Dumb” Fuzzer – Mutation Based

- Can grasp some meanings of the input
  - Numbers
  - Strings
  - Data

- Example of Smart “Dumb” Fuzzer
  - Radamsa -> https://gitlab.com/akihe/radamsa
  - Dizzy -> https://github.com/ernw/dizzy
  - Etc…

Demo

ERNW
providing security.

https://blog.ret2.io/2018/07/25/pwn2own-2018-safari-sandbox/

Dumb Fuzzers

+

- Very fast to implement
- Just mutates and flips bits
- Usually finds quite easy bugs very fast
- No need of source code
- ...

-

- Fails on language-based inputs (XML/SQL/Code)
- Usually hits same paths
- Does not learn anything about the input
- ...

ERNW
providing security.

Grammatic based Fuzzer

- Grammatic -> formalisms to formally specify input languages

```
<start> ::= <digit><digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

ERNW
providing security.

Grammatic based Fuzzer

- Grammatic -> formalisms to formally specify input languages

```
<start> ::= <integer>
<integer> ::= <digit> | <digit><integer>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

ERNW
providing security.

Grammatic based Fuzzer

- Grammatic -> formalisms to formally specify input languages

```
<start> ::= <number>
<number> ::= <integer> | +<integer> | -<integer>
<integer> ::= <digit> | <digit><integer>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

ERNW
providing security.

Grammatic based Fuzzer

- The fuzzer will generate “valid” input based on the given grammar

- Example:
  - Domato -> https://github.com/googleprojectzero/domato
  - GBF


ERNW
providing security.
Demo

ERNW
providing security.

Grammatic based Fuzzer

+

- Generates Input based on a given grammatic
- Can mutates on it
- Can find complex bugs in programs
- No need of source code
- ...

-

- Hard to implement
- Bugs to find are implementation specific
- Slow
- ...

ERNW
providing security.

Evolutionary/Genetic Fuzzer

- Evolutionary Input Generation
  - Incremental
- Uses Control Flow Feedback
  - Trace while fuzzing
- Code Coverage Driven

- Example:
  - American Fuzzy Lop (AFL)
  - Honggfuzz
  - LibFuzzer

ERNW
providing security.

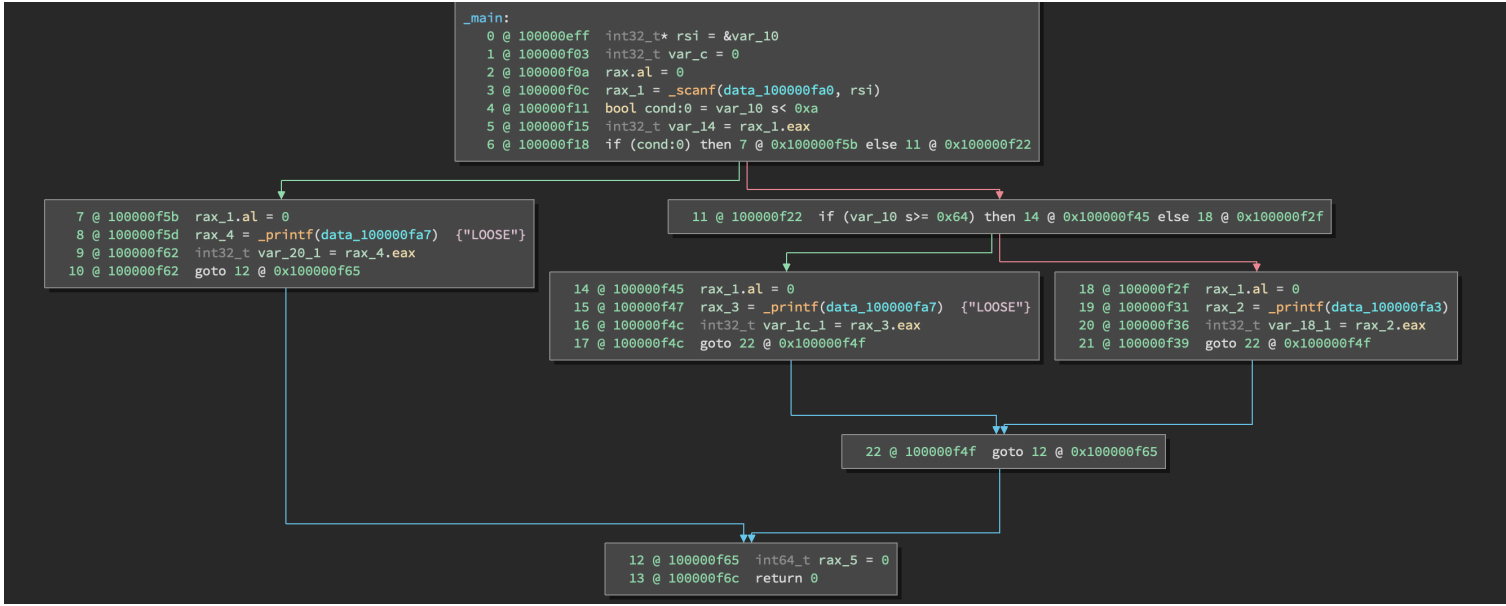
History of Evolutionary/Genetic Fuzzers

- 2006 – Sidewinder – Sparks & Cunningham
- 2007 – Evolutionary Fuzzing System – Jared Demott
- 2007 – Bunny the Fuzzer – Michal Zalewski (lcamtuf)
- 2013 – American Fuzzy Lop – Michal Zalewski (lcamtuf)
- 2014 – Nightmare/BCCF – Joxean Koret
- 2015 – Honggfuzz – Robert Swiecki
- 2015 – covFuzz – Atte Kettunen
- 2016 – Choronzon – Zisis Sialveras /Nikos Naziridis

Coverage Example

```c
#include <stdio.h>

int main(){
    int n;
    scanf ("%d",&n);
    if(n>=10){
        if(n<100){
            printf("WIN");
        }else{
            printf("LOOSE");
        }
    }else{
        printf("LOOSE");
    }
    return 0;
}
```



ERNW
providing security.

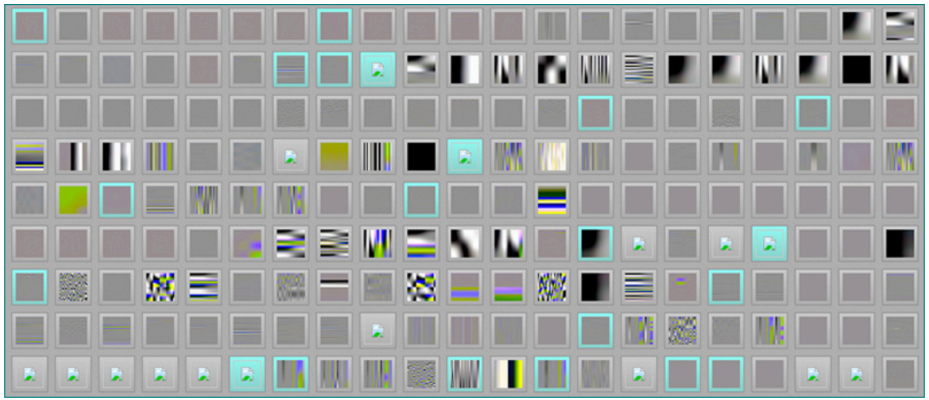
After AFL Instrumentation


ERNW
providing security.
Demo

ERNW
providing security.

Pulling JPEGs out of thin air

```
$ mkdir in_dir
$ echo 'hello' >in_dir/hello
$ ./afl-fuzz -i in_dir -o out_dir
./jpeg-9a/djpeg
```



ERNW
providing security.

Optimizing AFL

- Time consuming initializations
  - __AFL_INIT();
- Persistent Mode
  - while (__AFL_LOOP(1000)) { <Fuzz> }

Things to consider

- Most complete input corpus you can get (unit tests?)
- Minimal Corpus
- It is all about speed here
  - Parallelize
  - Optimize

- Use generated Input cases as Unit Test cases for later development

ERNW
providing security.

Recommendation for Evolutionary Fuzzer

- Throwing it at something that takes in a complex language (JavaScript) might not be the very best idea
- Try to limit threads to a minimum (Coverage can be corrupted)
- Patch out sanity checks like Checksums (CRC)

ERNW
providing security.

Evolutionary Fuzzer

+

- Learns about the input
- Traverses slowly through the program
- Finds deeply hidden bugs
- Easy to parallelize
- ...

-

- Fails on language-based inputs (XML/SQL/Code)
- Harder to implement
- Required to write a wrapper
- Needs ”sort of” source code
- Hard times with strcmp()
- ...

ERNW
providing security.

Hybrid Fuzzer

- Evolutionary /Genetic Fuzzers do work great on byte formats
- Grammatic Fuzzers work great on languages (JavaScript/SQL)

- Combining them requires some sort interpreted language
  - Inject bytecode to the interpreter
  - Mutate on the bytecode
  - Bytes -> Language
    - LLVM
    - Java Bytecode
    - Python Bytecode

ERNW
providing security.

Hybrid Fuzzer

+

- Mixes Grammatic and Genetic Fuzzer
- Based on bytecode and intermediate language
- Fast way to find bugs in very complex environments

-

- Very hard to implement
- Usually not applicable to companies
- Partially loosing advantages from evolutionary fuzzer

ERNW
providing security.

Recommendations

- Always try to implement a dumb fuzzer
- If the software needs a higher security level
  - Invest in a active developed and matured fuzzer
- Green Field Fuzzing
  - If someone else touched it! Try to implement your own!
  - Especially on Generation Based Fuzzer
- When your Fuzzer stop finding bugs it is time to advance it
  - Not to stop it!

Considerations

- Fuzzing will find you bugs - But not all bugs
- Hard to find Bugs
  - Race Conditions
  - Double Fetches

AddressSanitizer

AddressSanitizer Overview

- Introduced in LLVM/Clang(>v3.1) and GCC(>v4.8)
- Finds
  - Buffer Overflows (stack, heap, globals)
  - Heap-use-after-free, stack-use-after-return
  - Leaks, double-frees, etc.
- Compiler Module (LLVM, GCC)
  - Instruments all loads/stores
  - Inserts redzones/guards around stack/globals
- Run-time library
  - malloc replacements (redzones/guards)

Demo

Compiler Sanitizer

- AddressSanitizer (detects addressability issues)
- LeakSanitizer (detects memory leaks)
- ThreadSanitizer (detects data races and deadlocks)
- MemorySanitizer (detects use of uninitialized memory)
- UndefinedBehaviourSanitizer (detects conversion and overflow problems)

ERNW
providing security.

AddressSanitizer

- Should be used in every toolchain and testing/fuzzing phase to rule out errors

```
warning: trailing garbage in marker segment (3 bytes)
warning: trailing garbage in marker segment (15 bytes)
=================================================================
==10951==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xb5301a80 at pc 0xb7195911 bp 0xbf941cb8 sp 0xbf941ca8
READ of size 4 at 0xb5301a80 thread T0
    #0 0xb7195910 in jpc_pi_nextpcrl /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_t2cod.c:354
    #1 0xb719249e in jpc_pi_next /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_t2cod.c:120
    #2 0xb719c56c in jpc_dec_decodepkts /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_t2dec.c:441
    #3 0xb711efcf in jpc_dec_process_sod /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_dec.c:628
    #4 0xb711da24 in jpc_dec_decode /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_dec.c:425
    #5 0xb711ce62 in jpc_decode /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jpc/jpc_dec.c:262
    #6 0xb7108b53 in jp2_decode /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/jp2/jp2_dec.c:218
    #7 0xb70e0260 in jas_image_decode /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/base/jas_image.c:444
    #8 0x8049ad3 in main /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/appl/jasper.c:236
    #9 0xb6f19636 in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x18636)
    #10 0x80491d0  (/home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/build-asan-debug/src/appl/jasper+0x80491d0)

0xb5301a80 is located 0 bytes to the right of 48-byte region [0xb5301a50,0xb5301a80)
allocated by thread T0 here:
    #0 0xb728ddee in malloc (/usr/lib/i386-linux-gnu/libasan.so.2+0x96dee)
    #1 0xb70eb2d8 in jas_malloc /home/fire/bing/afl/libraries/jasper/jasper-version-2.0.5/src/libjasper/base/jas_malloc.c:242
```

ERNW
providing security.

Combining booth worlds

- Fuzzer + AddressSanitizer

AFL + AddressSanitizer

- Will be significant slower but it is usually good to have an instance running
- Regularly run all generated input through all needed sanitizer

ERNW
providing security.

Rule of Thumb

- Fuzzers don´t have to be perfect
  - start with Alpha version and fuzz
  - improve over time
  - Not fuzzing is wasting time/CPU cycles
- As code owner
  - Start fuzzing now. You will have a time and source code advantage!
- Fuzzer writing (50-70%)
- Making it stable (30-50%)
- Monitor Coverage and extend corpus
- Real work starts when triaging all the crashes to find the valuable bugs / or fixing them ;-)

ERNW
providing security.

Thank you for your Attention!

Questions?

bkauer@ernw.de

@lod108

www.ernw.de

www.insinuator.net