Fuzzing For Worms.

AFL for Linux Network Servers

Dobin Rutishauser

01 SSL/TLS Recommendations
// OWASP Switzerland 2013

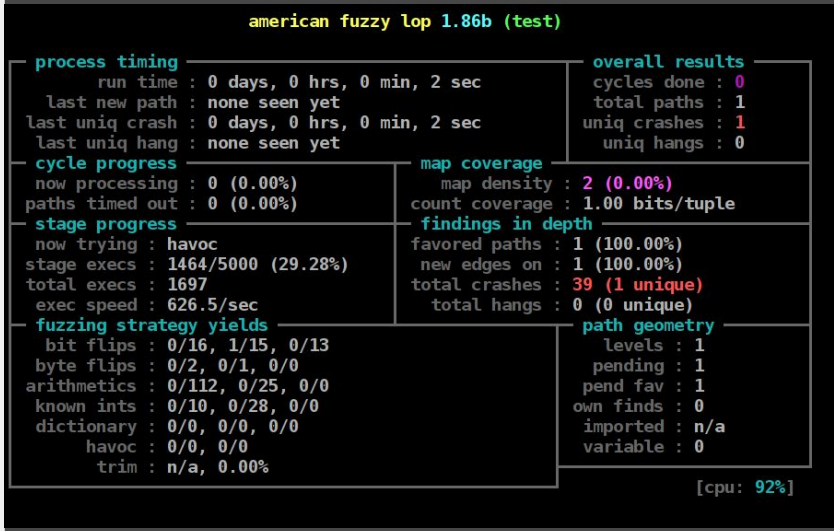
02 BurpSentinel - Semi Automated Web Scanner
// BSides Vienna 2014

03 Automated WAF Testing and XSS Detection
// OWASP Switzerland 2015, Barcamp

04 Lecturer Exploit Development & Mitigations
// Berner Fachhochschule, 2016, 2017, 2018

What we have	01	File Based Fuzzer
What I want	02	Network Fuzzer
Demo	03	Praise the Demo gods!
Feedback Driven Fuzzing	04	Honggfuzz
Results	05	0days?

```
cd ~/binutils-2.25

CC=afl-gcc ./configure

make

mkdir afl_in afl_out

cp /bin/ps afl_in/

afl-fuzz -i afl_in -o afl_out
    ./binutils/readelf -a @@
```

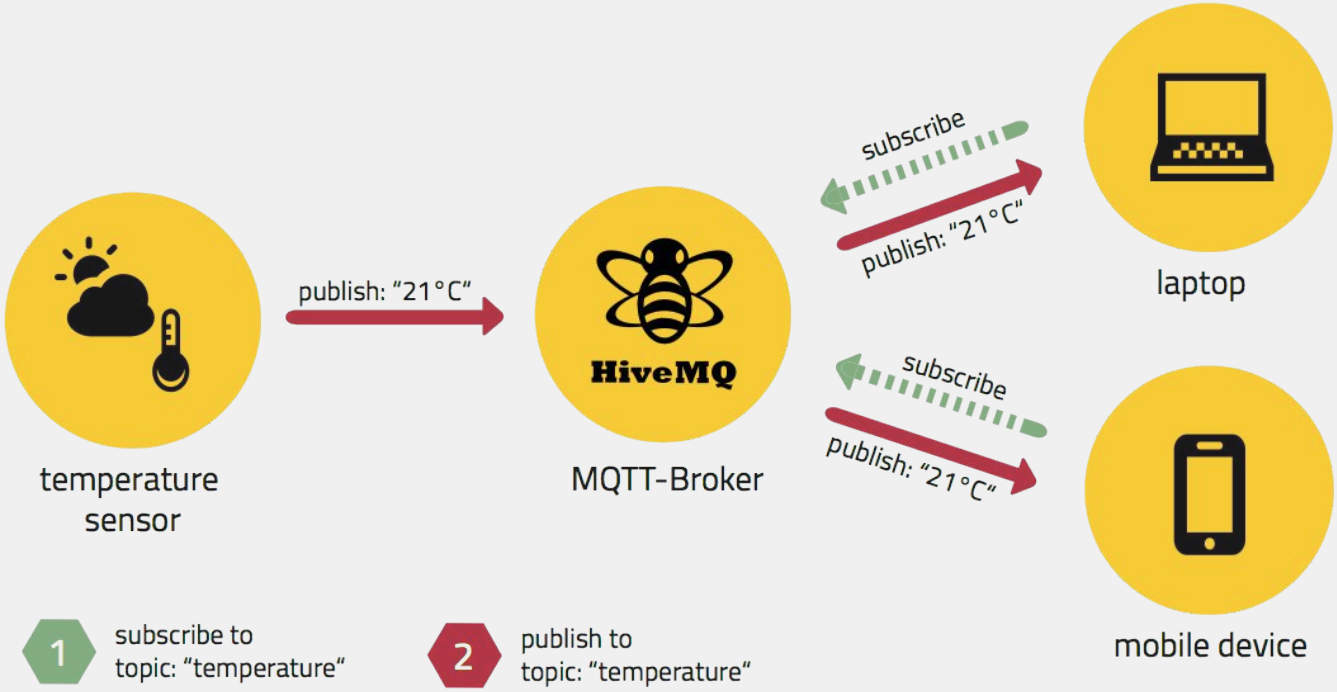


IJG jpeg 1	libjpeg-turbo 1 2	libpng 1
libtiff 1 2 3 4 5	mozjpeg 1	PHP 1 2 3 4 5 6 7 8
Mozilla Firefox 1 2 3 4	Internet Explorer 1 2 3 4	Apple Safari 1
Adobe Flash / PCRE 1 2 3 4 5 6 7	sqlite 1 2 3 4...	OpenSSL 1 2 3 4 5 6 7
LibreOffice 1 2 3 4	poppler 1 2...	freetype 1 2
GnuTLS 1	GnuPG 1 2 3 4	OpenSSH 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 ...

Everything already fuzzed :-(

Why no fuzzing network servers?

OSI Model		
Layer		Protocol data unit (PDU)
Host layers	7. Application	Data
	6. Presentation	
	5. Session	
	4. Transport	Segment (TCP) / Datagram (UDP)
Media layers	3. Network	Packet
	2. Data link	Frame
	1. Physical	Bit



https://www.hivemq.com/blog/how-to-get-started-with-mqtt

01 **CONNECT**

02 **SUBSCRIBE <channel>**

03 **PUBLISH <channel> <msg>**

Using AFL to fuzz a MQTT server?

Method 1: Wrap

Wrap packet-parsing function

Method 2: Rewire

Re-wire read(), write() to read from files

Method 1: Wrap

Isolate main functionality

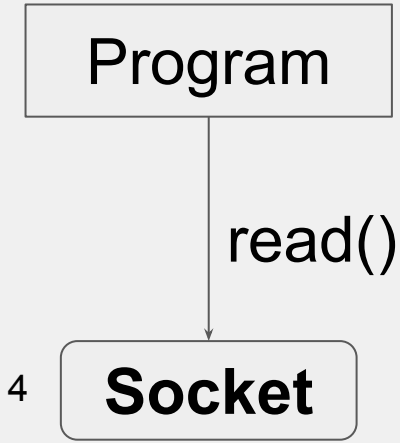
Write a wrapper

// Libfuzzer

// AFL Persistent Mode

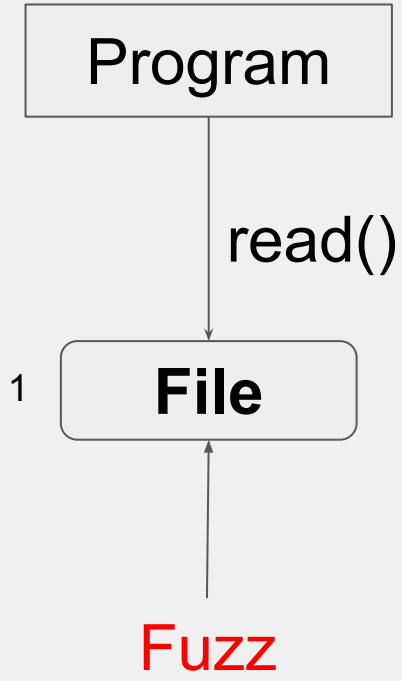
// WinAFL

```
extern "C" int LLVMFuzzerTestOneInput(
  const uint8_t *data,
  size_t size)
{
  parse_mqtt_packet(data);
}
```



Method 2: Re-wire

Re-wire read(), write() to read from files (not from sockets)



Method 2: Re-wire

Re-wire read(), write() to read from files (not from sockets)

// AFL Style

// Preeny desock

Case Study:
Mongoose 6.8

CSNC-2017-023 //

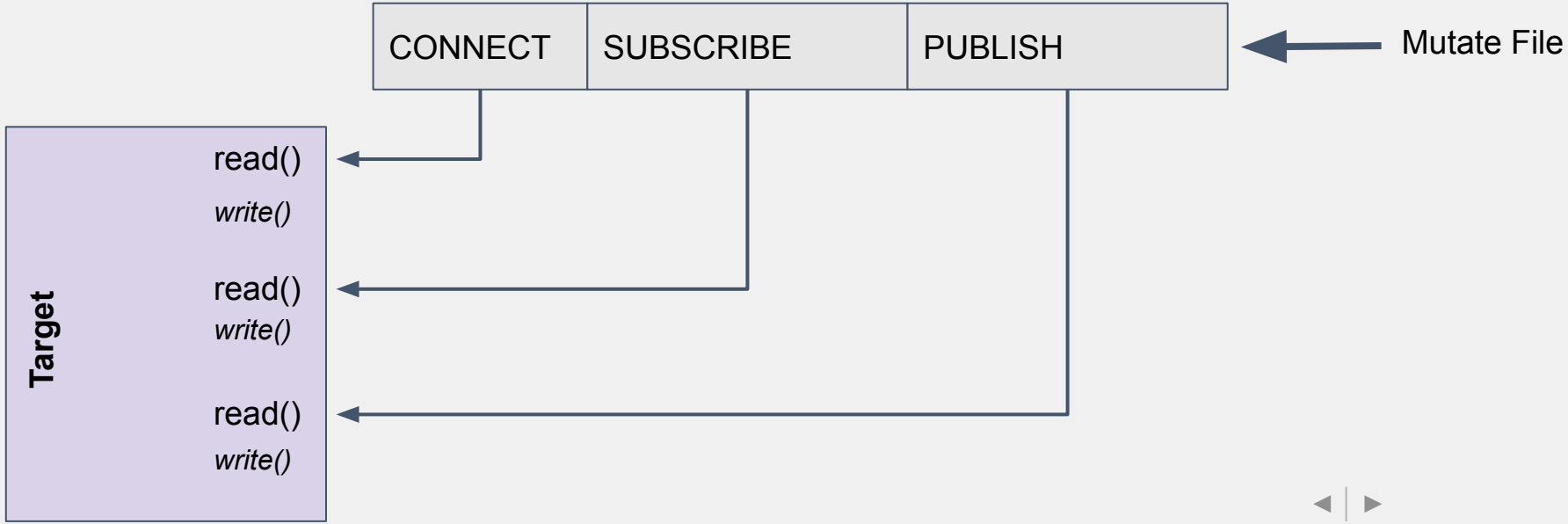
Manual Re-Wire for de-socking

Fuzz with AFL

Stack BOF RCE Exploit

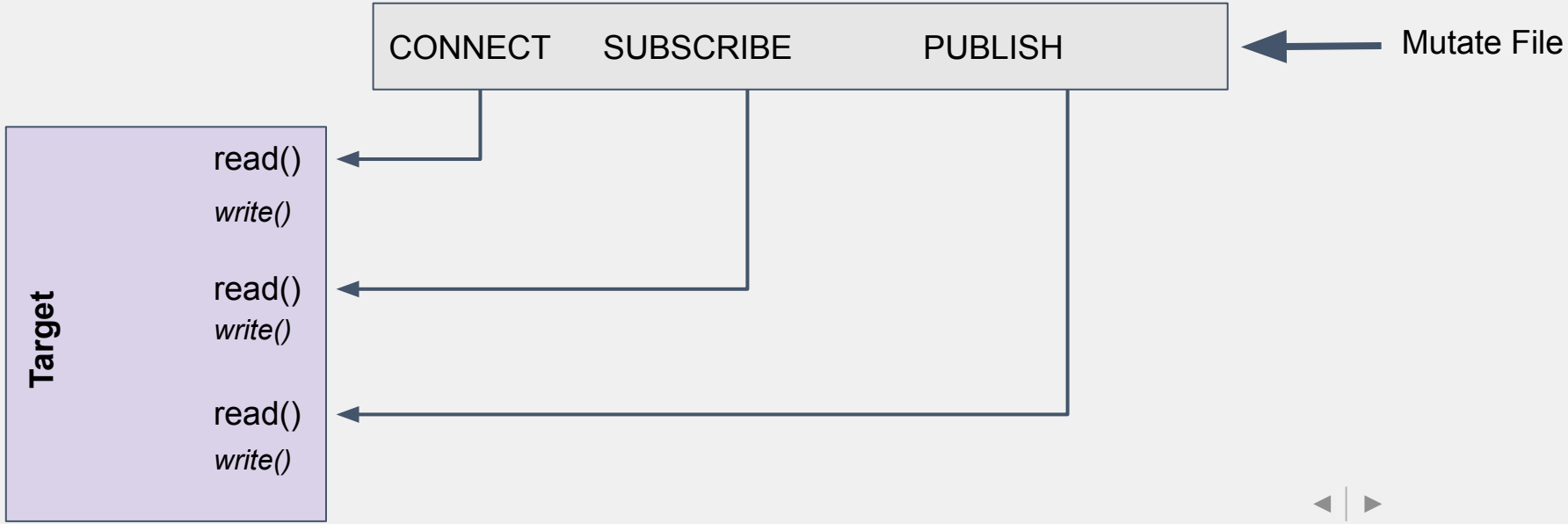
File Based Fuzzer Problems

What are packets? //



File Based Fuzzer Problems

What are packets? //



A better way? Generational Fuzzers

Protocol Specifications //

Reproduce protocol in code as XML

Specify every field, datatype, etc.

```
<DataModel name="DataPASV">
    <String value="PASV "/>
    <String value=""/>
    <String value="\r\n"/>
  </DataModel>

  <!-- TYPE [type_first] [type_second]:
  set the type of file to be transfered -->
  <DataModel name="DataTYPE1">
    <String value="TYPE "/>
    <String length="1" value="A"/>
    <Block minOccurs="0" maxOccurs="1">
      <String length="1" mutable="true">
        <Hint name="ValidValues" value="N;T;C"
      </String>
    </Block>
    <String value="\r\n"/>
  </DataModel>
```

Generators:

- Peachfuzz (XML “Pits”)
- Spike -> Sulley -> Boofuzz (python specs)
- Blab
- Dharma

Protocol Learning:

- Netzob (UI)
- Pulsar (AI)

Stateless

DNS, DHCP, HTTP, COAP

Ignore reply

Packets are independant

Use AFL

Stateful

MQTT, VNC, Teamspeak

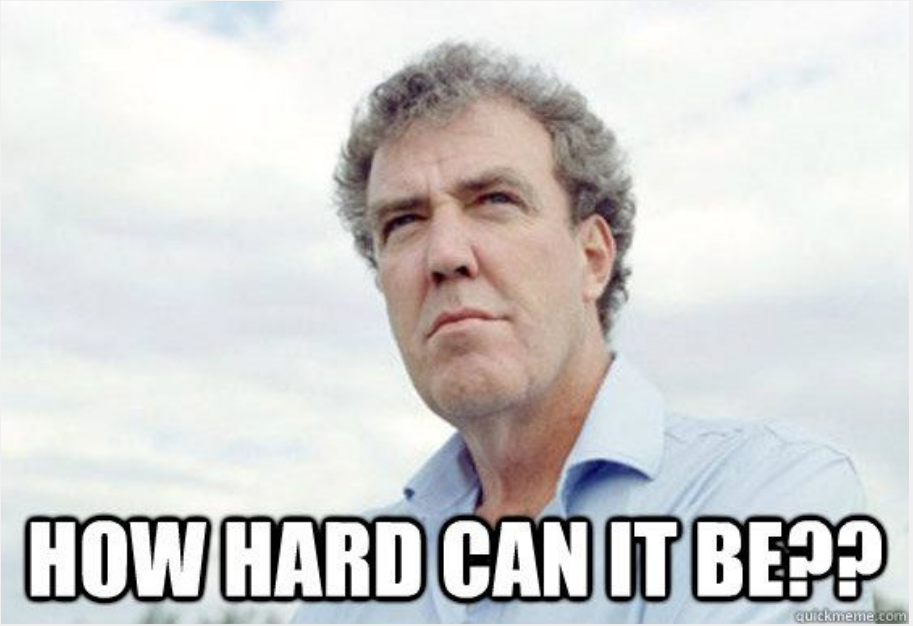
Responses important

Packets dependant

Use Peach

I dont want to **patch**
I dont want to **XML** protocols

i want to fuzz!



01

Intercept

Capture Network Data

02

Replay Data

Fuzz some data and send

03

Detect Crash

Check if server is stil alive

04

???

05

Profit!

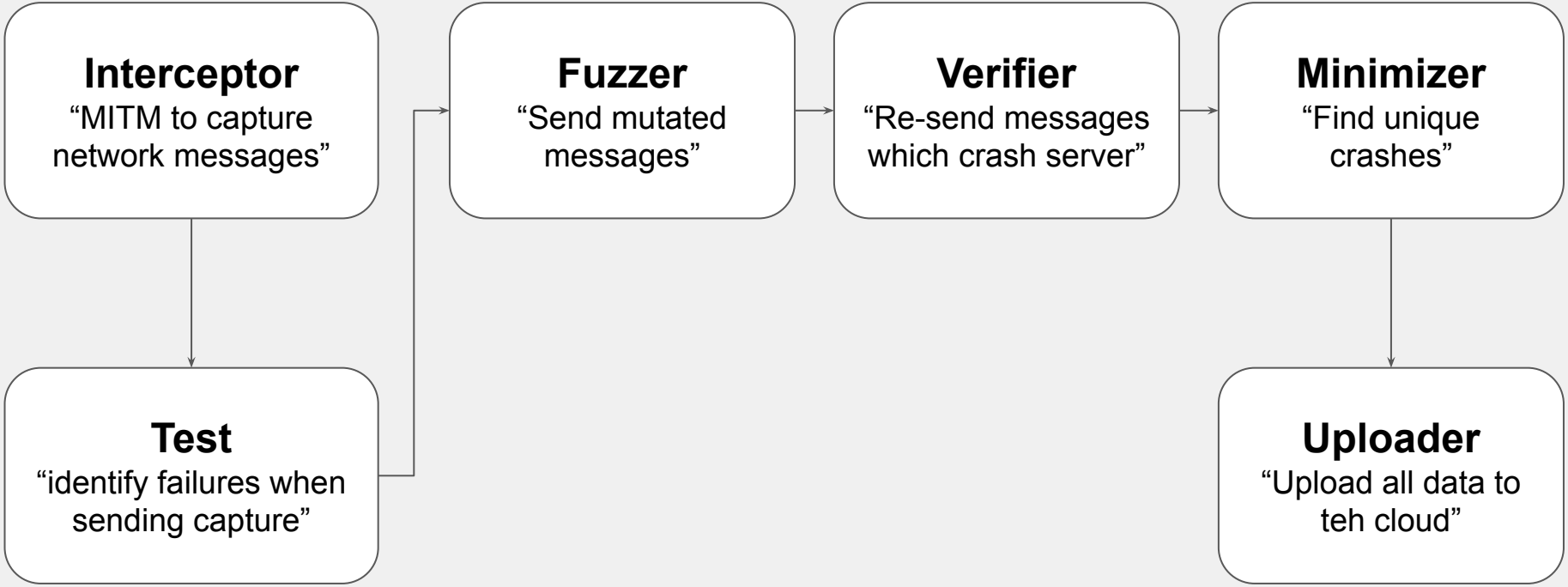
FFW Basic

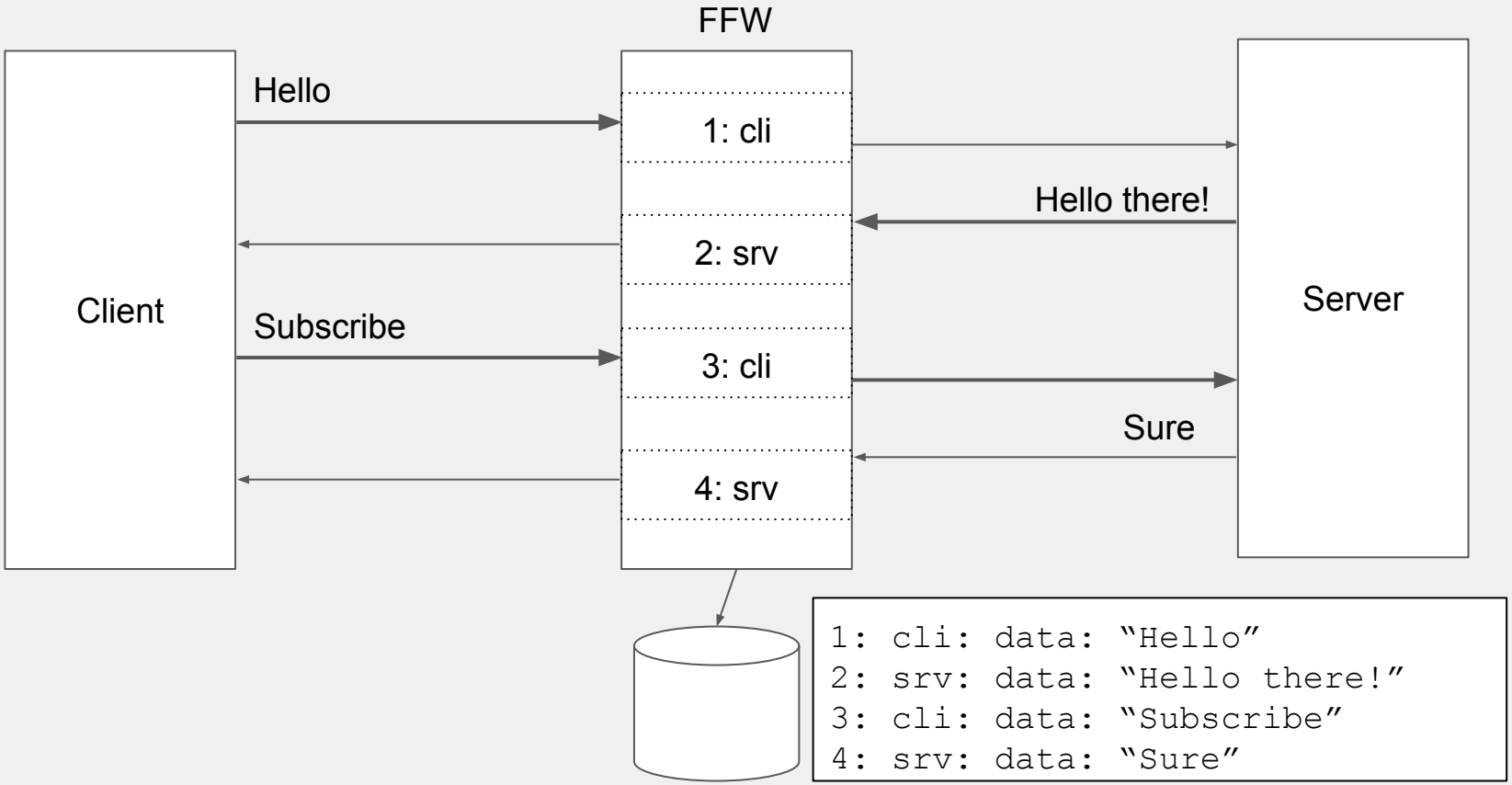
Mongoose 6.8

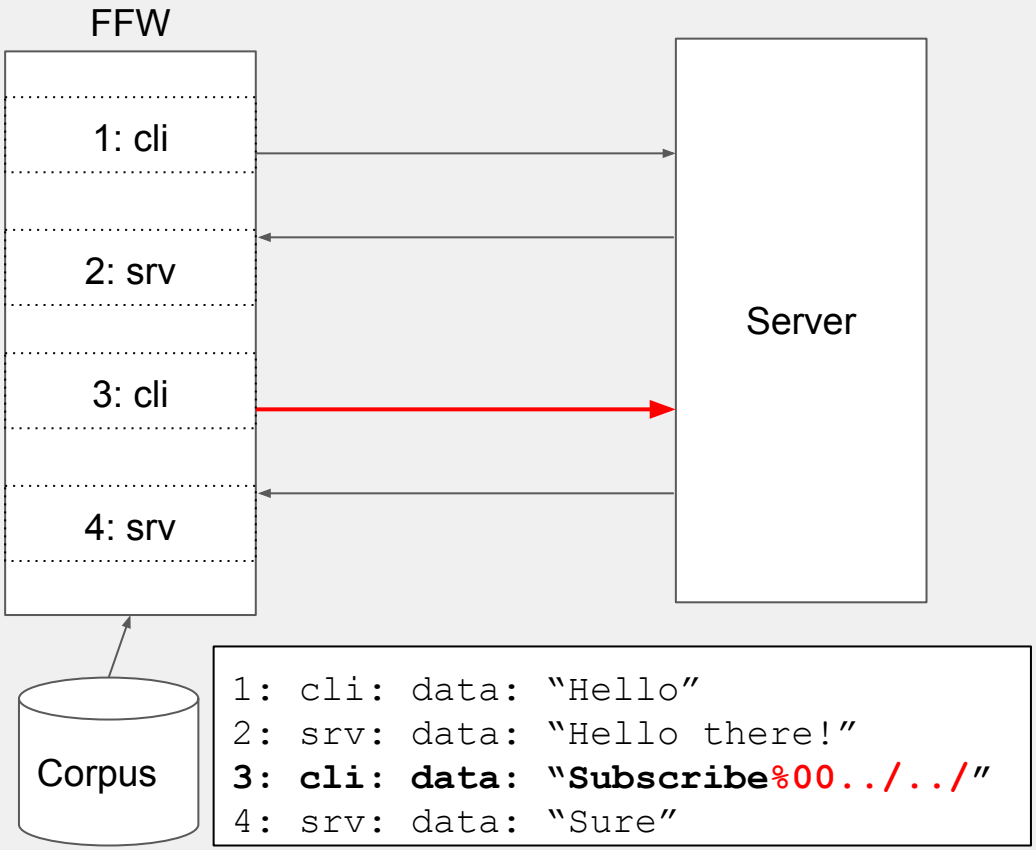
Demo

Praise the Demo Gods

FFW Internals


Interceptor
“MITM to capture network messages”
Fuzzer
“Send mutated messages”
Verifier
“Re-send messages which crash server”
Minimizer
“Find unique crashes”
Test
“identify failures when sending capture”
Uploader
“Upload all data to teh cloud”

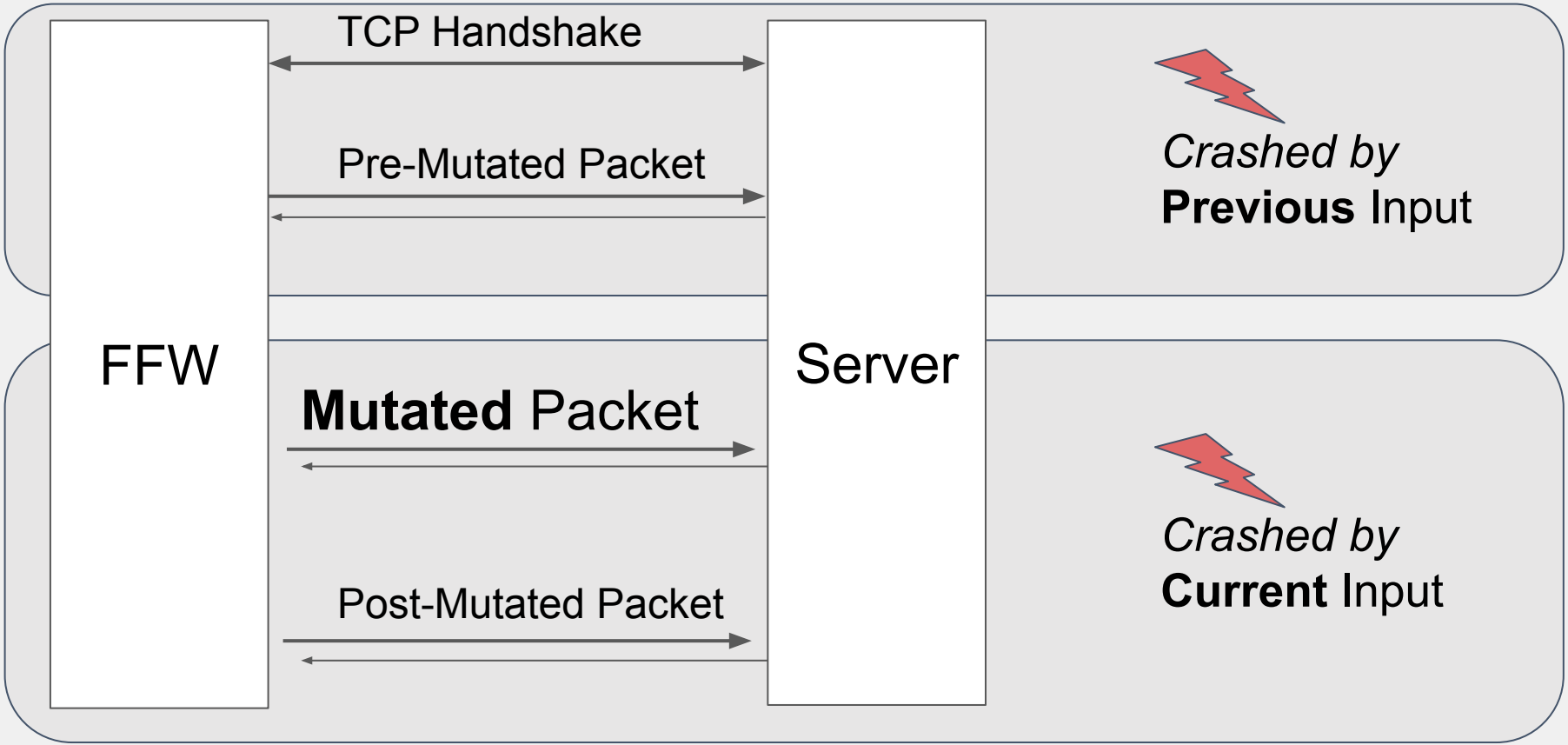

FFW
Client
Hello
Subscribe
1: cli
2: srv
3: cli
4: srv
Hello there!
Sure
Server
1: cli: data: "Hello"
2: srv: data: "Hello there!"
3: cli: data: "Subscribe"
4: srv: data: "Sure"


FFW
1: cli
2: srv
3: cli
4: srv
Server
Corpus
1: cli: data: "Hello"
2: srv: data: "Hello there!"
3: cli: data: "Subscribe%00../../"
4: srv: data: "Sure"

```
$ echo "aaa" | radamsa
aaaa
```

Here radamsa decided to add one 'a' to the input. Let's try that again.

```
$ echo "aaa" | radamsa
:aaa
```


TCP Handshake
Pre-Mutated Packet
Crashed by
Previous Input
FFW
Server
Mutated Packet
Crashed by
Current Input
Post-Mutated Packet

Crash Details:
Stack Trace

01 PTRACE
// Bad, Unreliable

02 ASAN
// Non-Stack BoF only

03 GDB
// Need Parser...

Fuzzing Hardware

What I have //

Xeon E5 2670 (from 2012)

20mb Cache!

8 cores, 16 threads!

32 GB RAM!

2.6ghz!

x2! rawr!

777$, natex.us

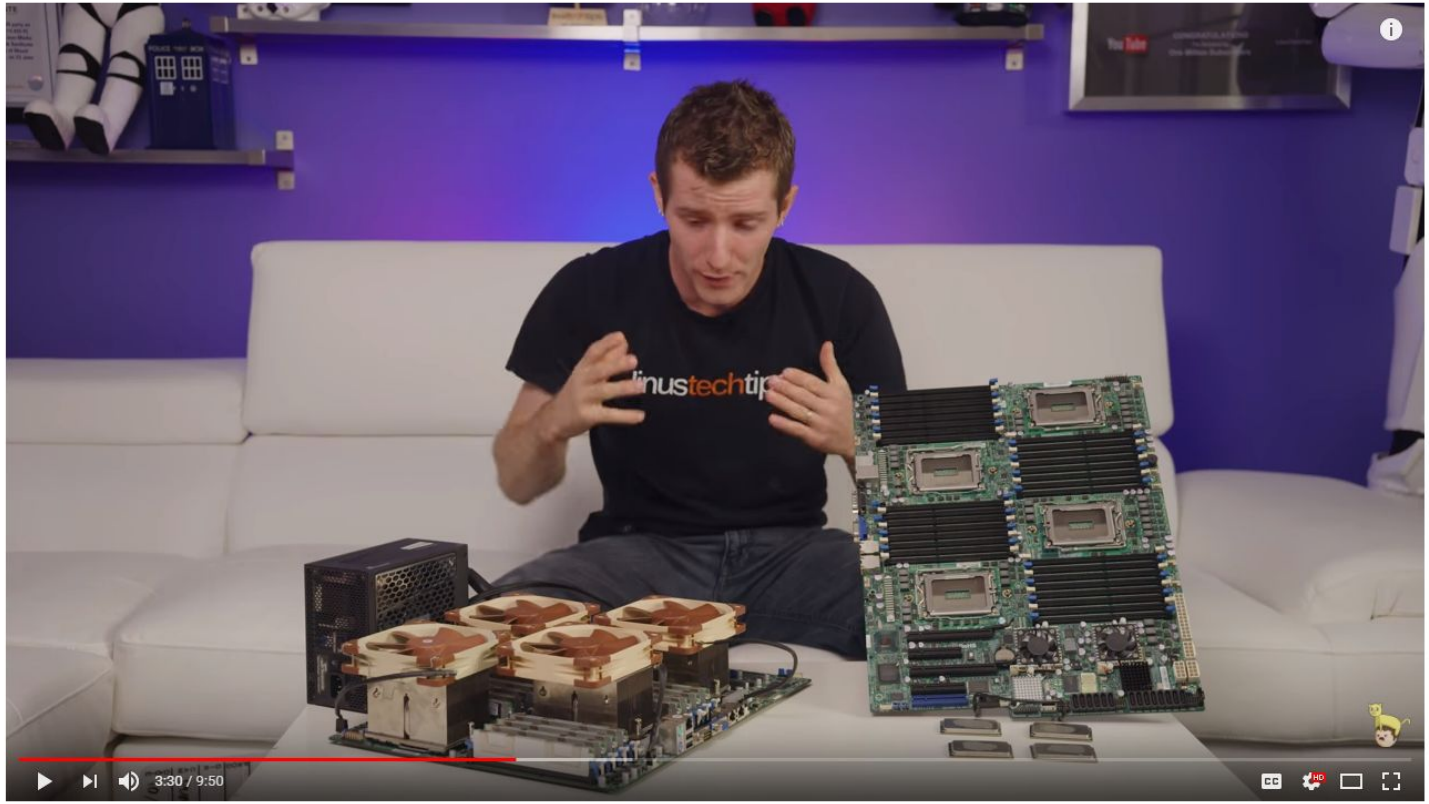
Prozessor

992.–
AMD Threadripper 1950X (TR4, 3.40GHz, Unlocked)

★★★★★ 4

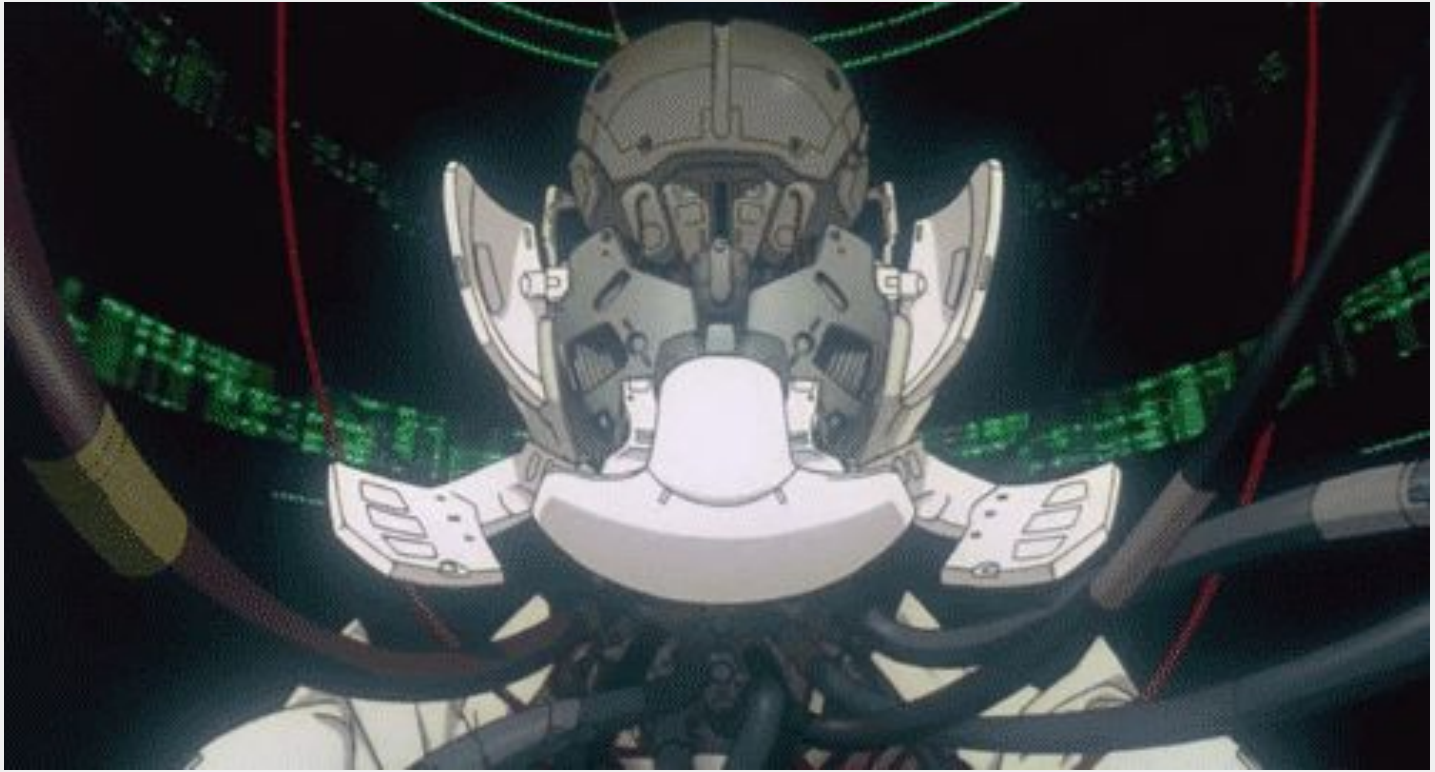
Prozessor

1'839.–
Intel Core i9-7960X (LGA 2066, 2.80GHz, Unlocked)



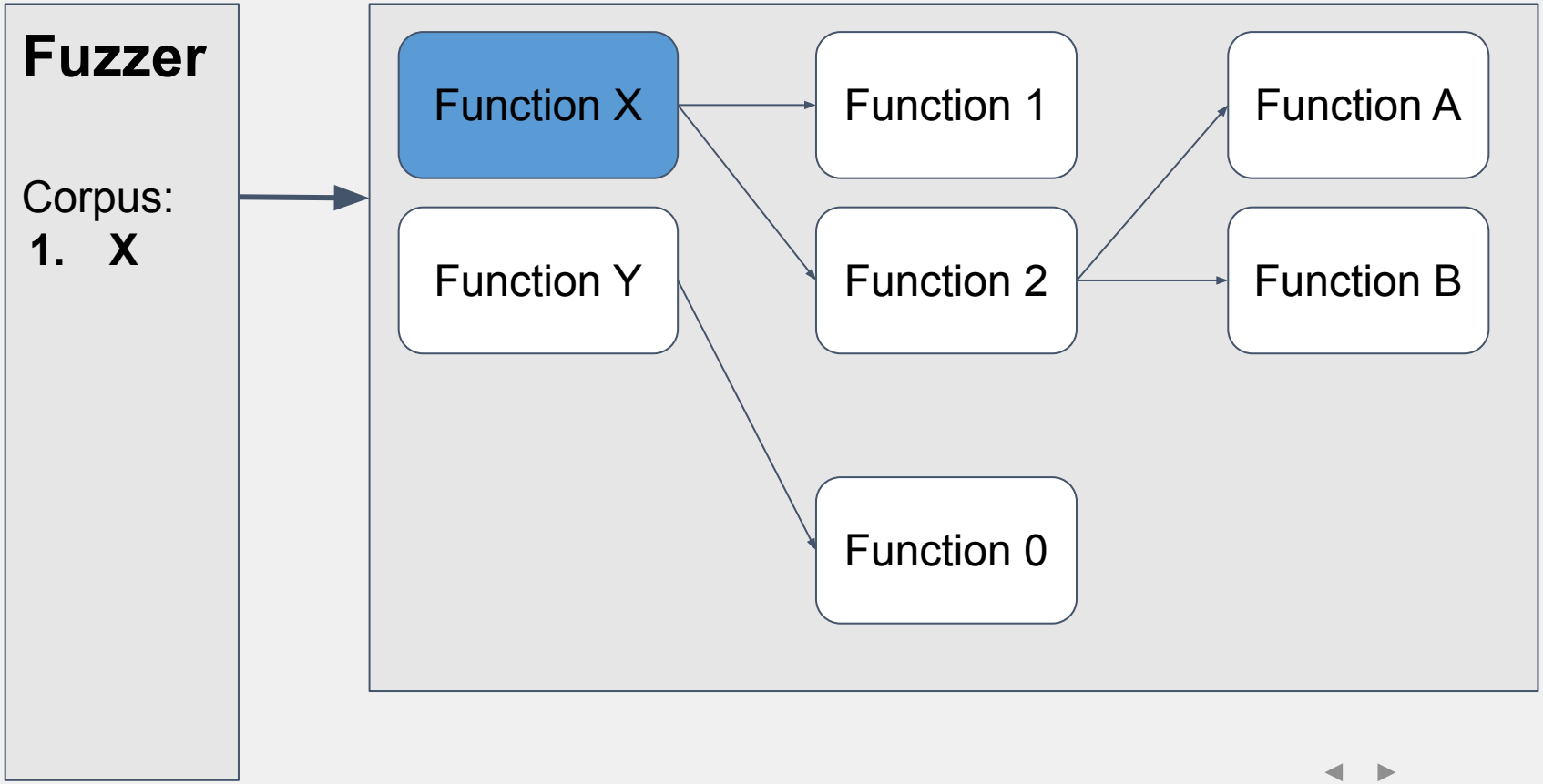
Fuzzing Hardware

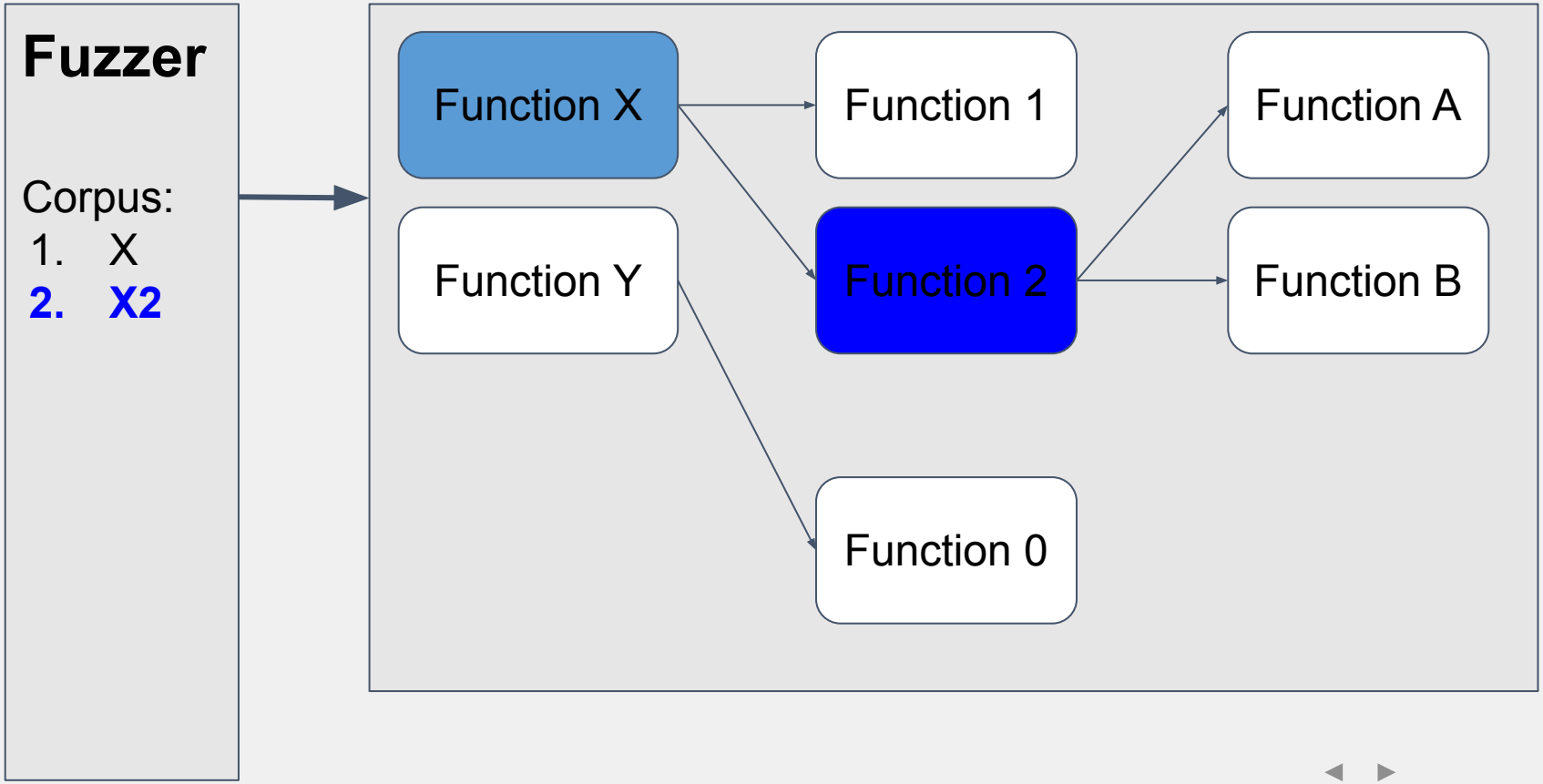
What I really need //

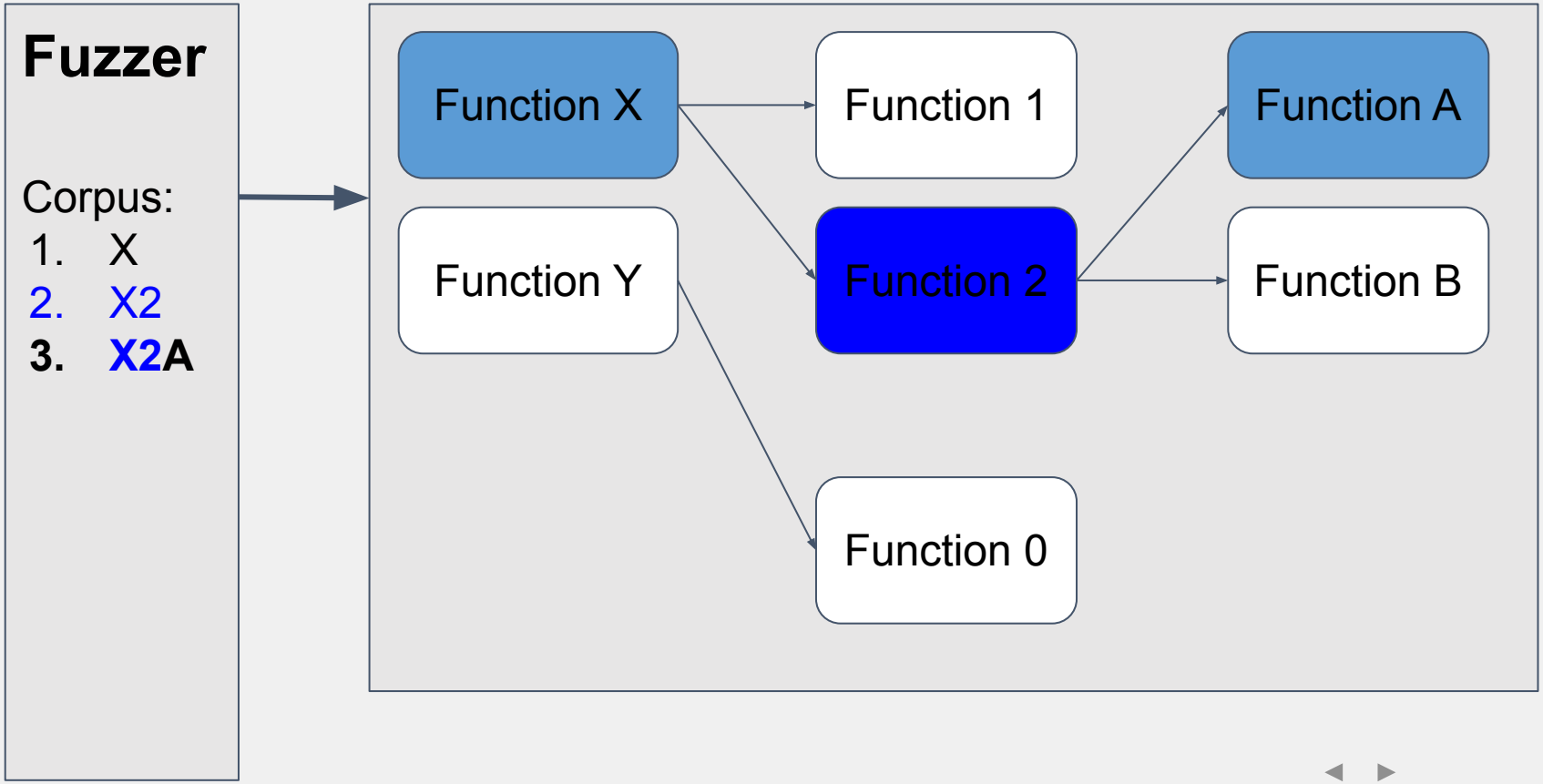


Puppet Master - Sentient Hacking AI, created by Intelligence Unit Section 6 (Ghost in the Shell, 1995)

Feedback Driven Fuzzing


Fuzzer
Corpus:
1. X
Function X
Function Y
Function 1
Function 2
Function 0
Function A
Function B

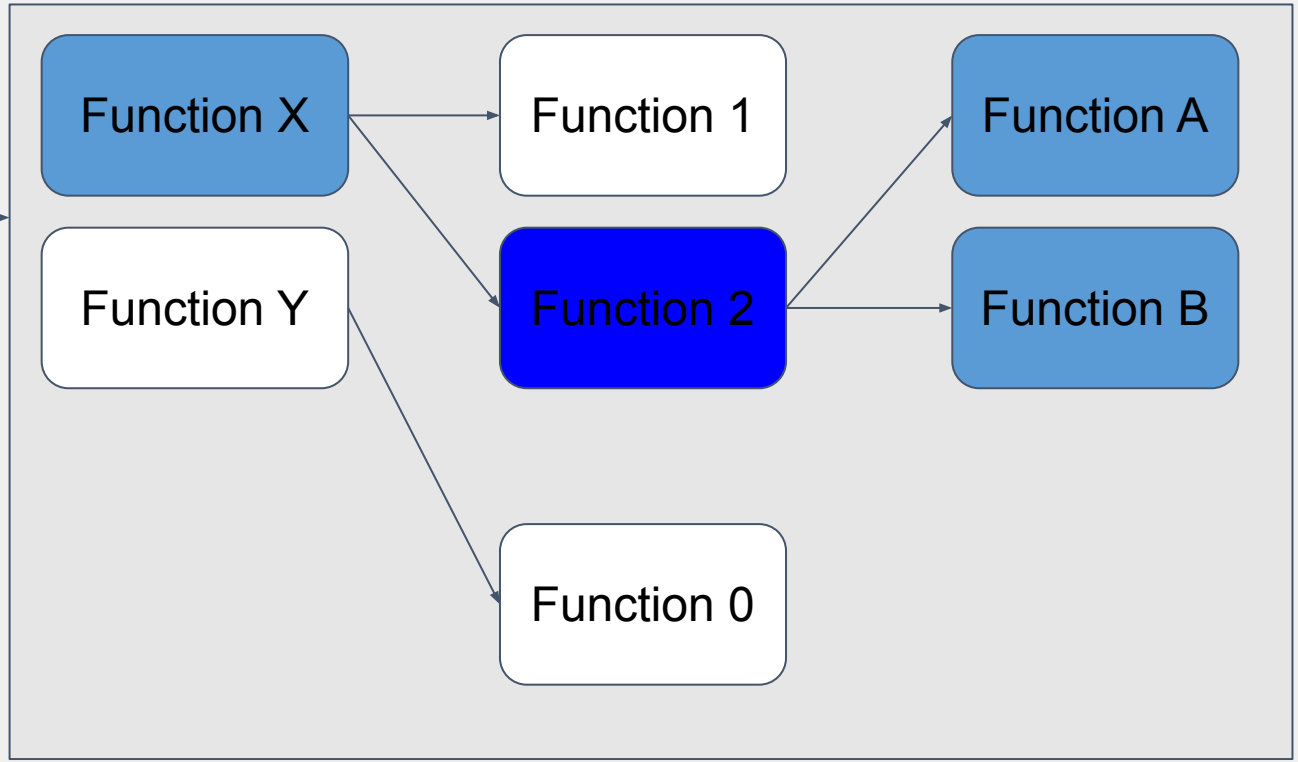

Fuzzer
Corpus:
1. X
2. X2
Function X
Function Y
Function 1
Function 2
Function 0
Function A
Function B


Fuzzer
Corpus:
1. X
2. X2
3. X2A
Function X
Function 1
Function A
Function Y
Function 2
Function B
Function 0

Fuzzer

Corpus:

1. X
2. X2
3. X2A
4. **X2B**



```
w = 0;
x = x + y;
y = 0;
if( x > z)
  {
   y = x;
   x++;
  }
else
  {
   y = z;
   z++;
  }
w = x + z;
```

Source Code

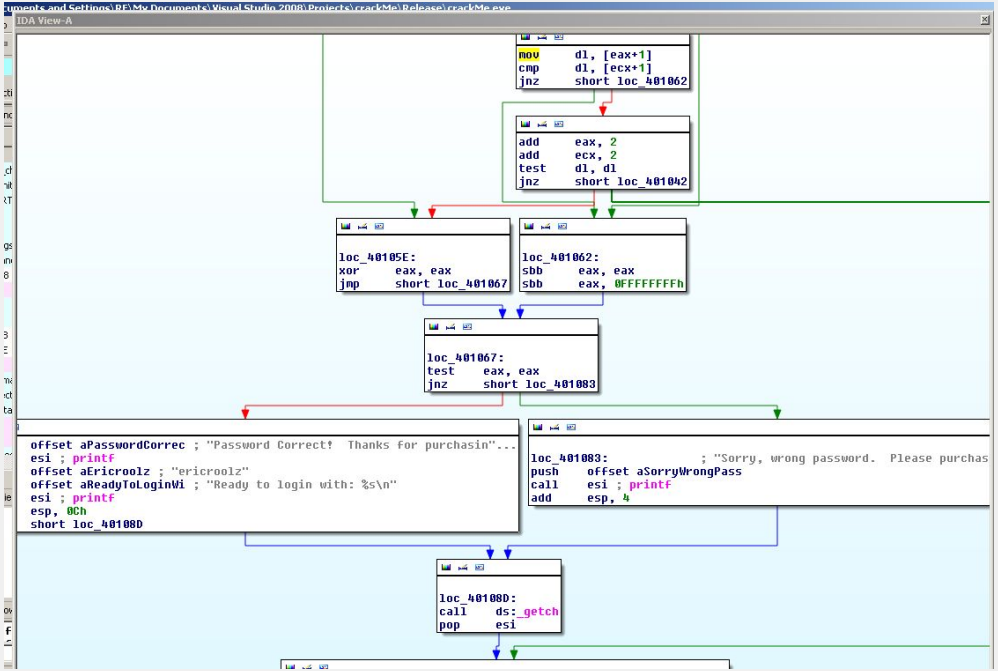
```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

```
y = x;
x++;
```

```
y = z;
z++;
```

```
w = x + z;
```

Basic Blocks



Coverage-guiding in action

The following code wants "ABCD" input:

```
if input[0] == 'A' {
    if input[1] == 'B' {
        if input[2] == 'C' {
            if input[3] == 'D' {
                slice[input[4]] = 1  // out-of-bounds here
}}}}
```

Blind generation needs O(2^8^4) = O(2^32) tries.

Corpus progression:

```
0. {}
1. {"A"}
2. {"A", "AB"}
3. {"A", "AB", "ABC"}
4. {"A", "AB", "ABC", "ABCD"}
```

Coverage-guided fuzzer needs O(4 * 2^8) = O(2^10) tries.

Fix all the bugs American Fuzzy Lop and Address Sanitizer, Hanno Böck

```
if input[0] == 0x41
```

```
if input[1] == 0x42
```

```
if input[2] == 0x43
```

```
if input[3] == 0x44
```

CRASH

0	0	0	0	0	0
0	0	0	**1**	0	0
0	0	0	0	**0**	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	**0**	0	0	0
0	0	0	0	0	**0**

At each basic block, add:

```
cur_location = (block_address >> 4) ^ (block_address << 8);

shared_mem[cur_location ^ prev_location]++;

prev_location = cur_location >> 1;
```

```
if input[0] == 0x41
```

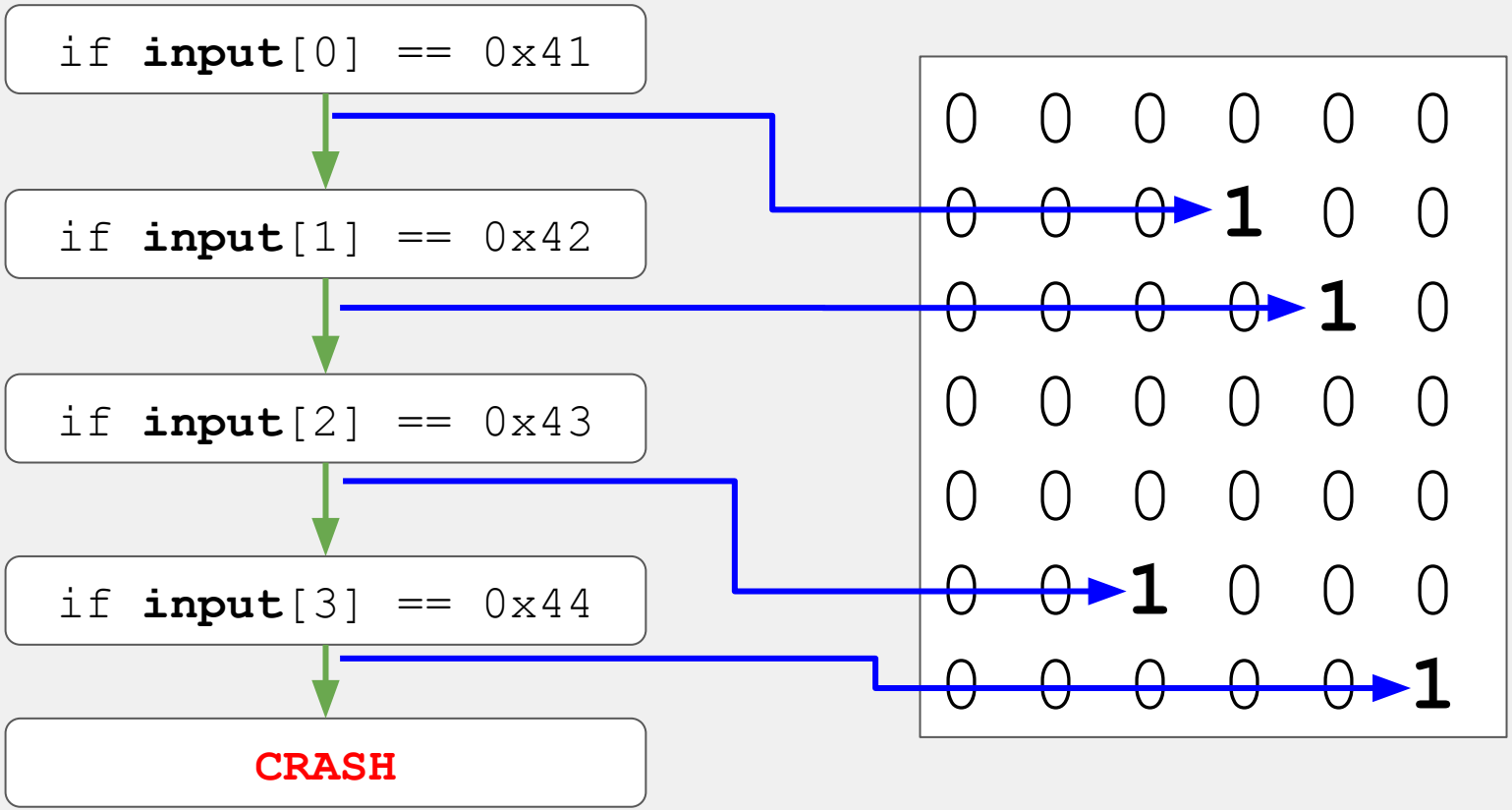
```
if input[1] == 0x42
```

```
if input[2] == 0x43
```

```
if input[3] == 0x44
```

CRASH

0	0	0	0	0	0
0	0	0	**1**	0	0
0	0	0	0	**1**	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	**0**	0	0	0
0	0	0	0	0	**0**


if input[0] == 0x41
if input[1] == 0x42
if input[2] == 0x43
if input[3] == 0x44
CRASH
0 0 0 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 0 0 1

```
109        24 :            } else if (a == 0xFB2Au) { /* SHIN WITH SHIN DOT */
110         0 :                *ab = 0xFB2Cu;
111         0 :                found = true;
112        24 :            } else if (a == 0xFB2Bu) { /* SHIN WITH SIN DOT */
113         0 :                *ab = 0xFB2Du;
114         0 :                found = true;
115           :            }
116        35 :            break;
117           :        case 0x05BFu: /* RAFE */
118        49 :            switch (a) {
119           :            case 0x05D1u: /* BET */
120        12 :                *ab = 0xFB4Cu;
121        12 :                found = true;
122        12 :                break;
123           :            case 0x05DBu: /* KAF */
124        11 :                *ab = 0xFB4Du;
125        11 :                found = true;
126        11 :                break;
127           :            case 0x05E4u: /* PE */
128        14 :                *ab = 0xFB4Eu;
129        14 :                found = true;
130        14 :                break;
131           :            }
132        49 :            break;
133           :        case 0x05C1u: /* SHIN DOT */
134        22 :            if (a == 0x05E9u) { /* SHIN */
135        12 :                *ab = 0xFB2Au;
136        12 :                found = true;
137        10 :            } else if (a == 0xFB49u) { /* SHIN WITH DAGESH */
138         0 :                *ab = 0xFB2Cu;
139         0 :                found = true;
140           :            }
141        22 :            break;
142           :        case 0x05C2u: /* SIN DOT */
143        22 :            if (a == 0x05E9u) { /* SHIN */
144        10 :                *ab = 0xFB2Bu;
145        10 :                found = true;
146        12 :            } else if (a == 0xFB49u) { /* SHIN WITH DAGESH */
147         0 :                *ab = 0xFB2Du;
148         0 :                found = true;
149           :            }
```

Picture Source:

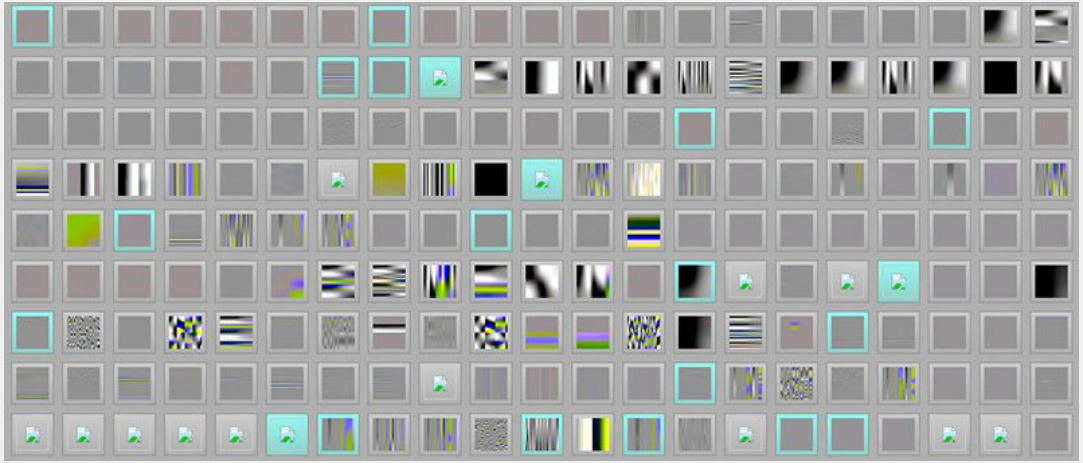
"Circumventing Fuzzing Roadblocks with Compiler Transformations", lafintel

November 07, 2014

Pulling JPEGs out of thin air

This is an interesting demonstration of the capabilities of afl; I was actually pretty surprised that it worked!

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



Feedback-driven fuzzing

Honggfuzz is capable of performing feedback-guided (code coverage driven) fuzzing. It can utilize the following sources of data:

- (Linux) Hardware-based counters (instructions, branches)
- (Linux) Intel BTS code coverage (kernel >= 4.2)
- (Linux) Intel PT code coverage (kernel >= 4.2)
- Sanitzer-coverage instrumentation (`-fsanitize-coverage=bb`)
- Compile-time instrumentation (`-finstrument-functions` or `-fsanitize-coverage=trace-pc[-guard],indirect-calls,trace-cmp` or both)

Developers should provide the initial file corpus which will be gradually improved upon. It can even comprise of a single 1-byte initial file, and honggfuzz will try to generate better inputs starting from there.

google / honggfuzz

Watch 93 | Star 880 | Fork 197

Code | Issues 2 | Pull requests 0 | Projects 0 | Insights

Branch: master | honggfuzz / docs / FeedbackDrivenFuzzing.md | Find file | Copy path

robertswiecki cmdline: make the compile-time instrumentation the default option, ca… 930e12f 11 days ago

3 contributors

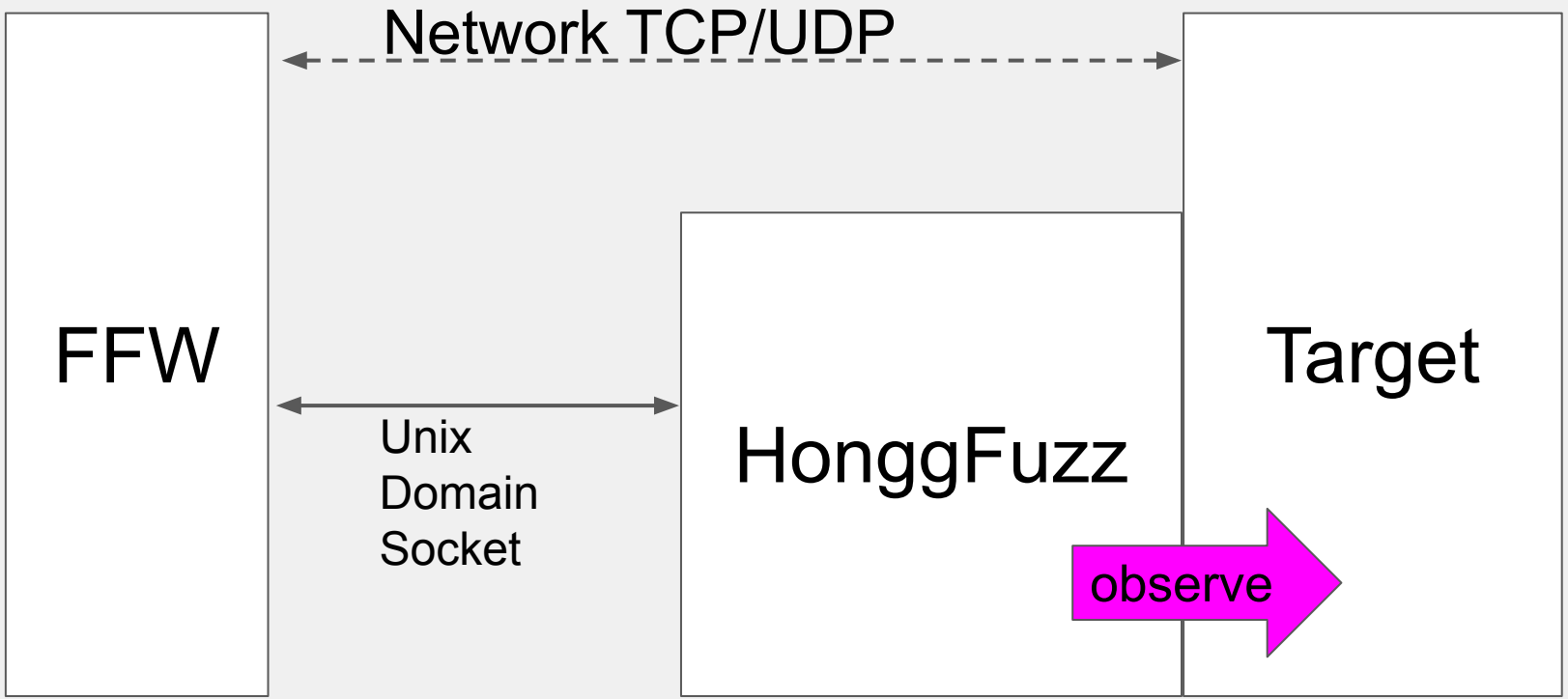
244 lines (204 sloc) | 11.6 KB | Raw | Blame | History

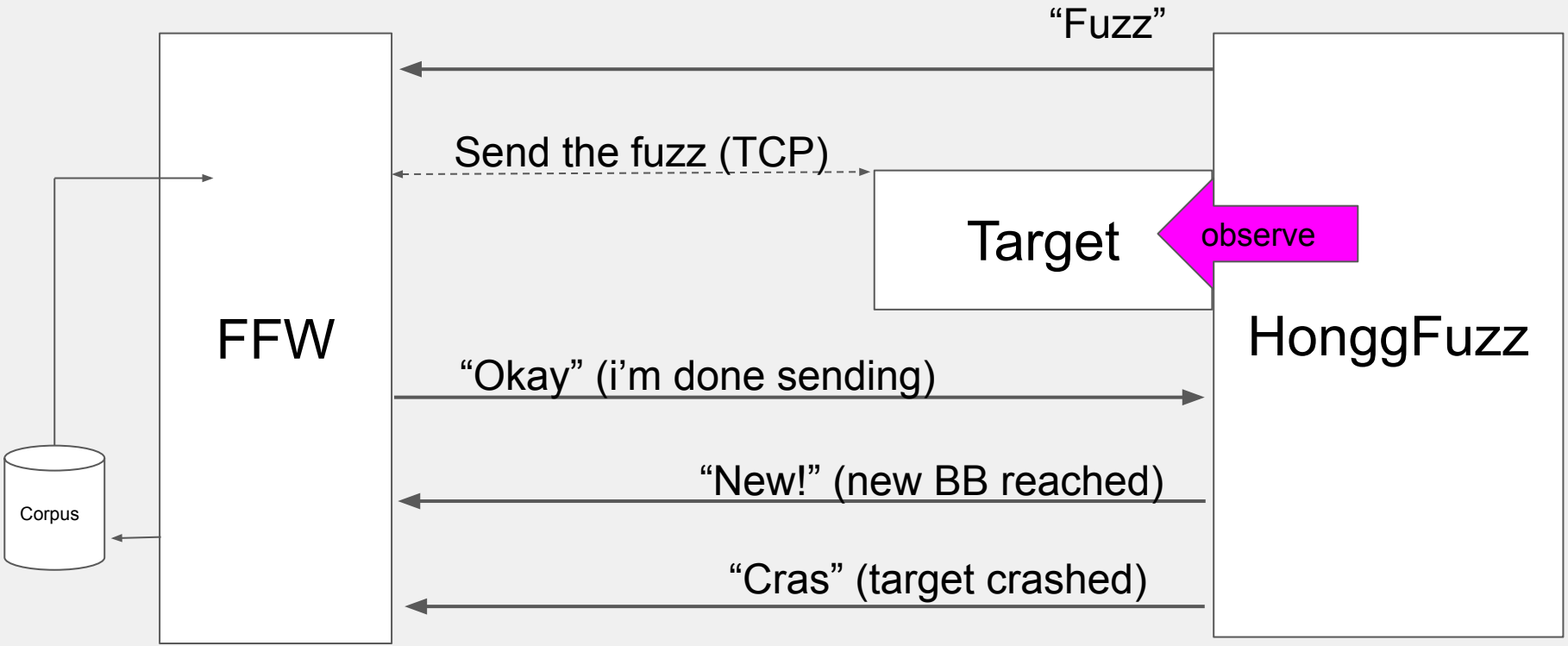
Feedback-driven fuzzing

Honggfuzz is capable of performing feedback-guided (code coverage driven) fuzzing. It can utilize the following sources of data:

- (Linux) Hardware-based counters (instructions, branches)
- (Linux) Intel BTS code coverage (kernel >= 4.2)
- (Linux) Intel PT code coverage (kernel >= 4.2)
- Sanitzer-coverage instrumentation (`-fsanitize-coverage=bb`)
- Compile-time instrumentation (`-finstrument-functions` or `-fsanitize-coverage=trace-pc[-guard],indirect-calls,trace-cmp` or both)

Developers should provide the initial file corpus which will be gradually improved upon. It can even comprise of a single 1-byte initial file, and honggfuzz will try to generate better inputs starting from there.


Network TCP/UDP
FFW
Unix
Domain
Socket
HonggFuzz
Target
observe


"Fuzz"
Send the fuzz (TCP)
Target
observe
FFW
HonggFuzz
"Okay" (i'm done sending)
"New!" (new BB reached)
"Cras" (target crashed)
Corpus

FFW

Mongoose 6.9

Feedback driven fuzzing with honggfuzz

Demo

Praise the Demo Gods

Performance

About **30 iterations/s** per thread :-(

AFL: Threads = core_count / 2

FFW: Threads = core_count * 2

- 30 iterations/s * 32 * 2 =~ 2000 (okish)

Parallel fuzzing -> parallel processes

```
-listen %(port)
```

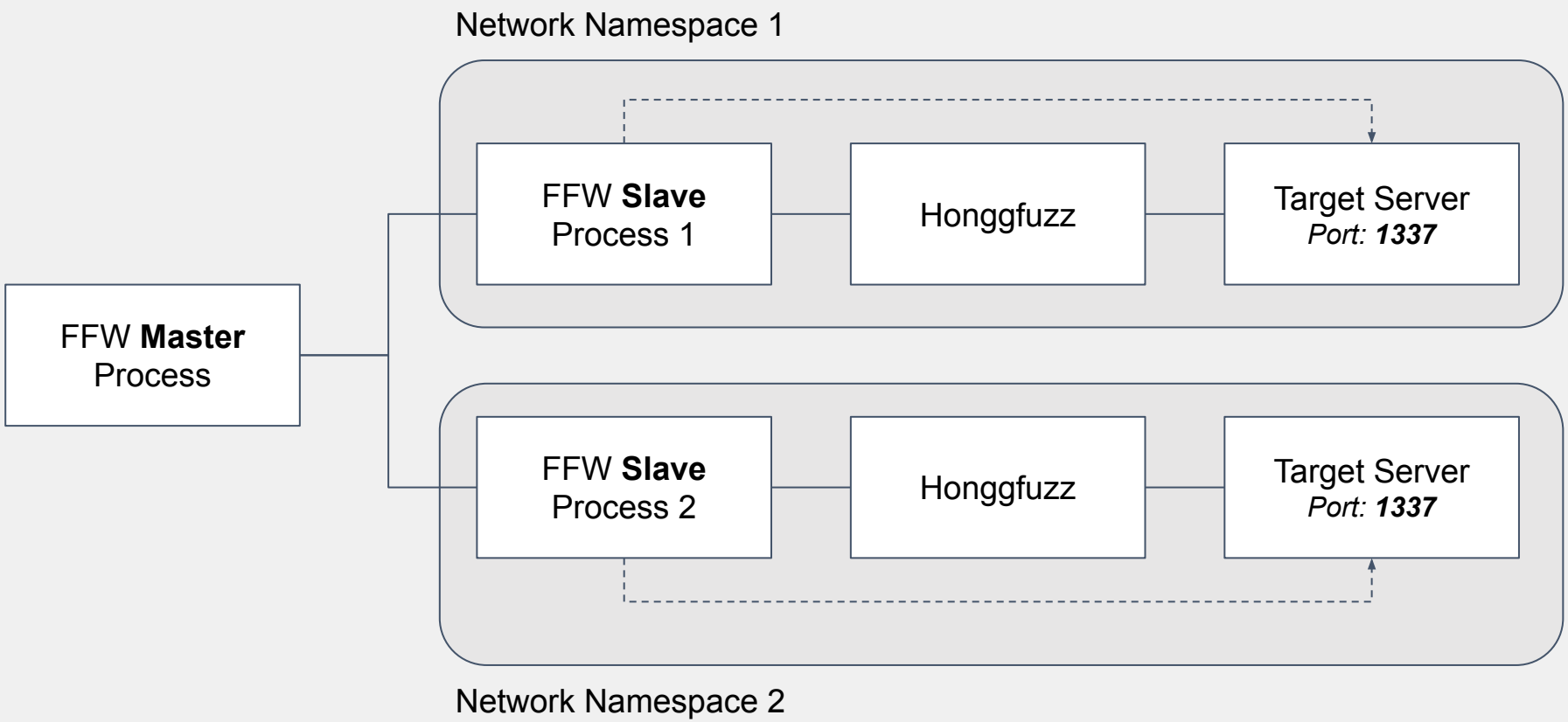
Port directly as Argument

```
-config srvcfg-%(port).conf
```

Port via Config File

```
usenetnamespace: "True"
```

Namespaces! yay!


Network Namespace 1
FFW Slave Process 1
Honggfuzz
Target Server
Port: 1337
FFW Master Process
FFW Slave Process 2
Honggfuzz
Target Server
Port: 1337
Network Namespace 2

Fuzzing Results

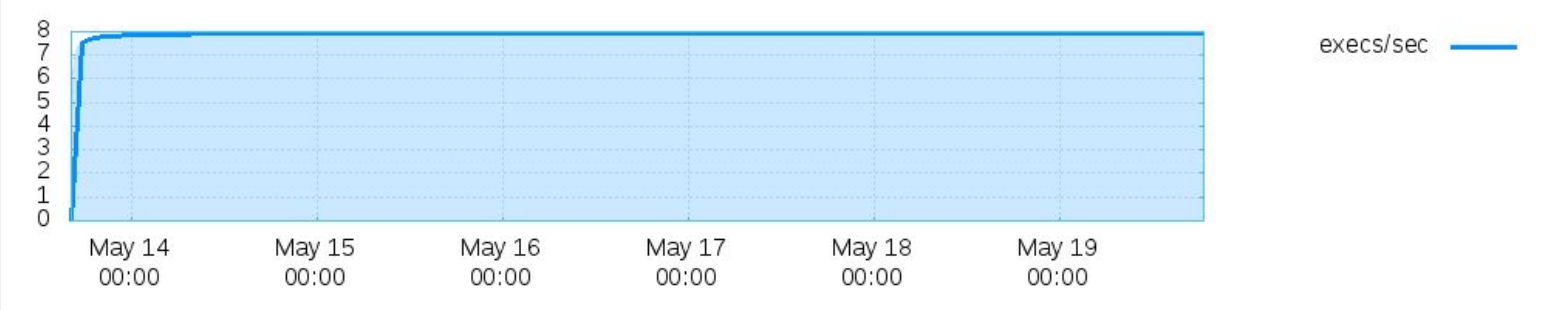
Bugs:

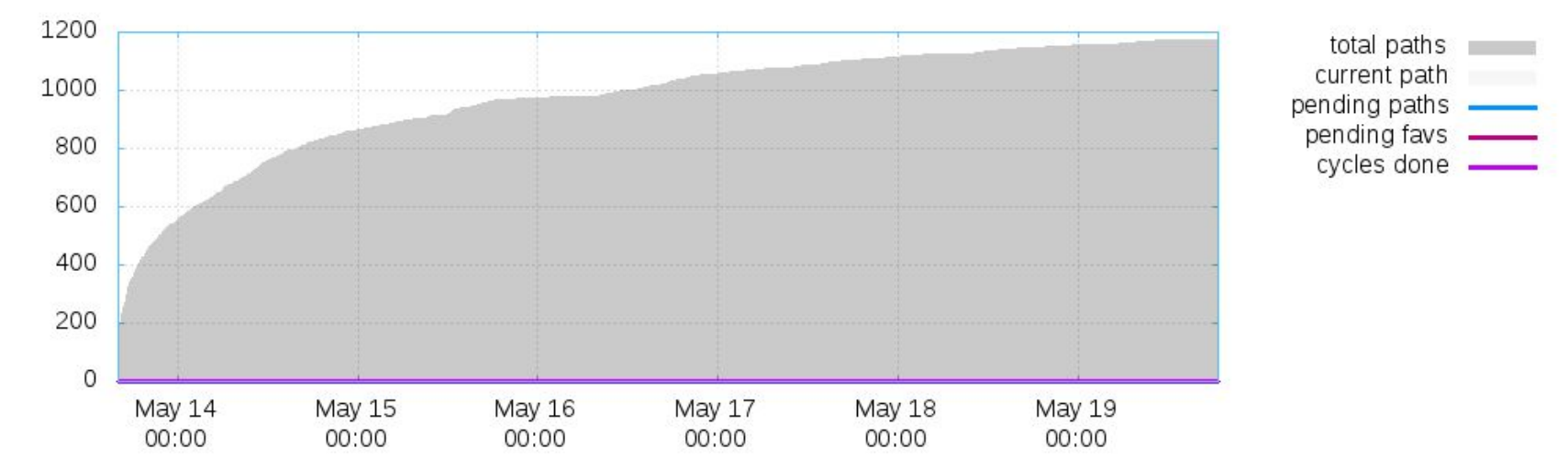
- Mongoose MQTT 6.8
- Mongoose MQTT 6.9
- libcoap
- libiec61850
- vnc4server
- libmodbus

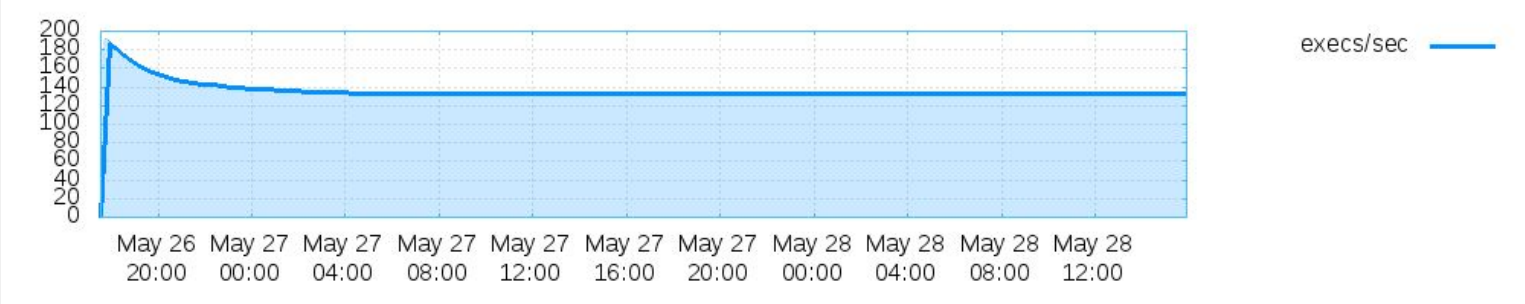
Check docker dobin/ffw:0.2

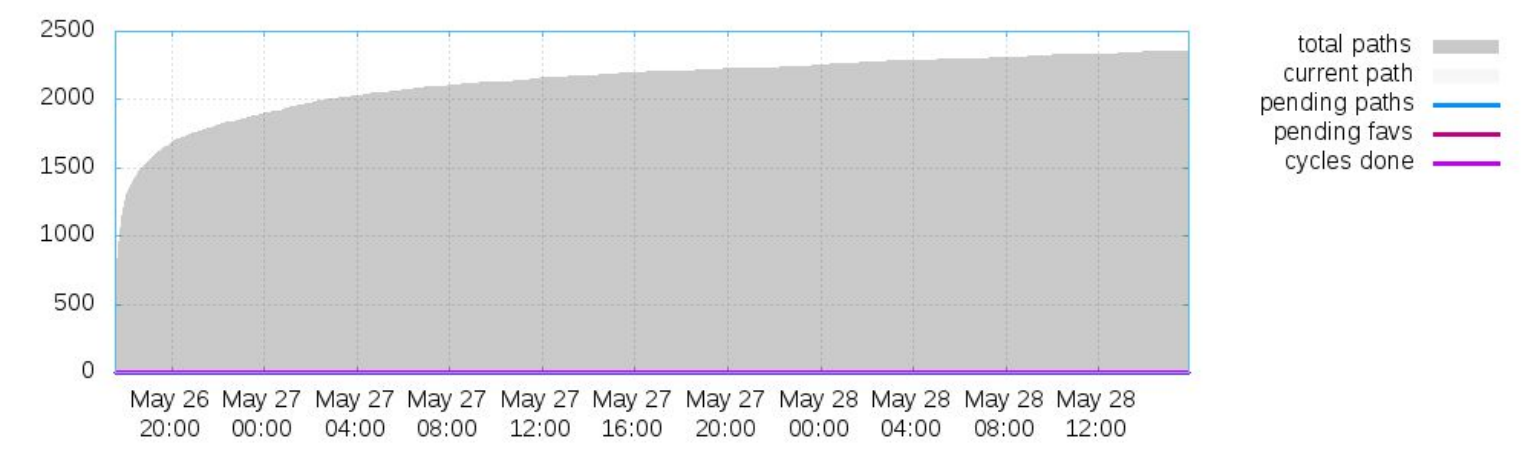
No Bugs:

- *Mongoose Web*
- *Mongoose DNS*
- *Mongoose DHCP*
- Synergy
- Ngircd
- Inspircd
- unrealircd
- portmapper

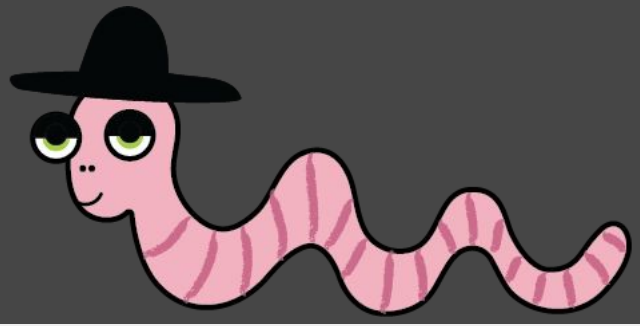

8
7
6
5
4
3
2
1
0
May 14 00:00
May 15 00:00
May 16 00:00
May 17 00:00
May 18 00:00
May 19 00:00
execs/sec


1200
1000
800
600
400
200
0
May 14 00:00
May 15 00:00
May 16 00:00
May 17 00:00
May 18 00:00
May 19 00:00
total paths
current path
pending paths
pending favs
cycles done


200
180
160
140
120
100
80
60
40
20
0
May 26 20:00
May 27 00:00
May 27 04:00
May 27 08:00
May 27 12:00
May 27 16:00
May 27 20:00
May 28 00:00
May 28 04:00
May 28 08:00
May 28 12:00
execs/sec


2500
2000
1500
1000
500
0
May 26 20:00
May 27 00:00
May 27 04:00
May 27 08:00
May 27 12:00
May 27 16:00
May 27 20:00
May 28 00:00
May 28 04:00
May 28 08:00
May 28 12:00
total paths
current path
pending paths
pending favs
cycles done

Conclusion



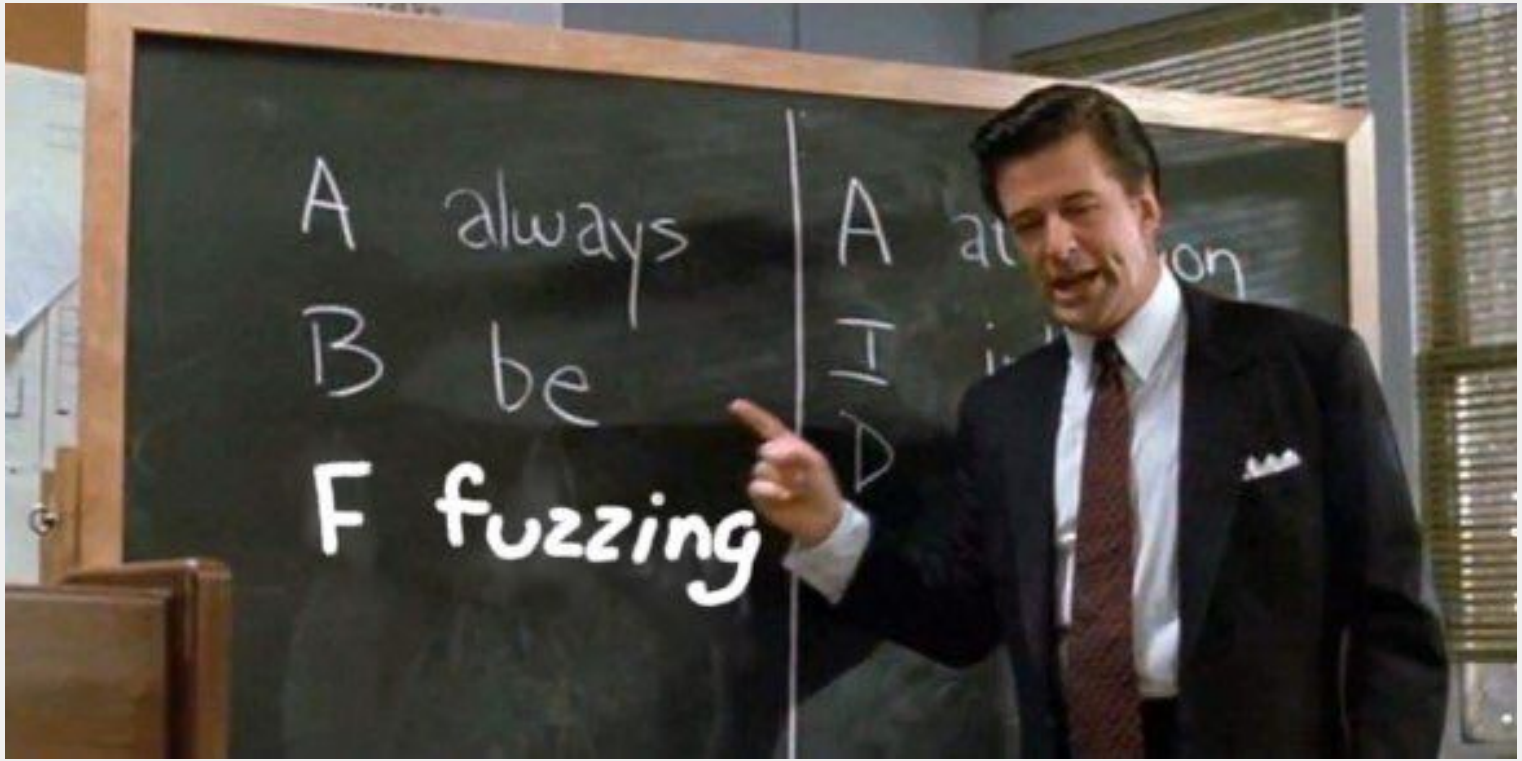
01 **INTERCEPT**

02 **RETRANSMIT (fuzzed)**

03 **DETECT**

- Crashes
- New Basic Blocks

- Swat low hanging bugs
- Use time before *real* fuzzing
- Quick-Check of a server


A always
B be
F fuzzing

Code: **github.com/dobin/ffw**

Me: **twitter.com/dobinrutis**

Blog: **dobin.ch/blog/tag/ffw**

0days: **github.com/dobin/ffw-docker**

Reddit: **reddit.com/r/fuzzing**

Get started:

```
docker run -ti --privileged -lxc-conf="aa_profile=unconfined" dobin/ffw:0.2
```

Q&A

Not in this presentation:

- Taint Tracking / Analysis
- SMT Solvers
- Automated root cause identification
- Automated exploit generation
- Automated patch generation
- Machine Learning / AI
- Kernel Fuzzing
- Syscall Fuzzing
- Instruction-Set Fuzzer
- Structure-aware fuzzing
- Anti-Fuzzing
- Binary Instrumentation (Intel PIN, DynamoRio)
- Blockchain (Smart Contracts / Ethereum)
- Symbolic Execution / Concolic Execution (Angr, Manticore)
- Compiler Transformations / Deoptimization

Fuzzotron

Available via https://github.com/denandz/fuzzotron. "Fuzzotron is a simple network fuzzer supporting TCP, UDP and multithreading."

Support network fuzzing, also uses Radamsa. Can use coverage data, but it is experimental.

Con's:

- Does not restart target server
- Unreliable crash detection
- Experimental code coverage

Mutiny

Available via https://github.com/Cisco-Talos/mutiny-fuzzer. "The Mutiny Fuzzing Framework is a network fuzzer that operates by replaying PCAPs through a mutational fuzzer."

Con's:

- No code coverage
- Only one commit (no development?)
- Rudimentary crash detection

Future

Fitness:
Fuzz which damn input?

- [] Input Corpus
  - [] Corpus Network Message

01 # Crashes

02 # Hangs

03 # Children

04 Latency

Generic Mutation Algorithmn

Per InputCorpus

01 Crossover
(copy partially to other network messages)

02 Mutation
(re-order network messages)

03 Swap
(swap messages)

Crash Analysis

Code Coverage in IDE

Adjust network timeout dynamically

Client program fuzzing


I wrote a vulnerability scanner that abstracts all the predicates in a binary, traverses the callgraph and generates phormulaes to run then with a SMT solver.
I found 1 vuln in 3 days with this tool.
He wrote a dumb ass fuzzer and found 5 vulns in 1 day.
Good thing I'm not a n00b like that guy.

The reason for all this:

C

C++

Feedback Mechanism

	Closed-Source	Kernel	Stable	Fast
Compile-Time Instrumentation	✗	✓	✓	++
Static Rewriting	✓	-	✗	++
Dynamic Binary Instrumentation	✓	-*	✓	-
Emulation	✓	✓	✓	- -
Intel Branch Trace Store	✓	✓	✓	+
Intel Processor Trace	✓	✓	✓	+++

* Peter Feiner, et al., DRK: DynamoRIO as a Linux Kernel Module