FitM

Binary-Only Coverage-Guided Fuzzing for Stateful Network Protocols

Binary Analysis Research (BAR) Workshop 2022
Fuzzing

TL;DR: Throw corner-case input at a program until it breaks.
Protocols are hard.
Motivation - Protocol Exploration

FTP Connection

Random TCP Port

TCP Setup

FTP Username

USER anonymous

FTP Password

derp@foocorp.com

Response 230

PORT 10.10.10.10 99

Response 200 - PORT Command Successful

NLST

DATA Connection

TCP Teardown

Server

Client
Motivation - Protocol Exploration

FTP Connection

Random
TCP Port

FTP-CTRL
TCP Port 21

TCP Setup

FTP Username

PASS
derp@foocorp.com

Response 230

PORT 10.10.10.99

Response 200 - PORT Command Successful

NLST

DATA Connection

TCP Teardown

USER anonymous
PASS anonymous

Client

Server
Motivation - Protocol Exploration

FTP Connection

Random TCP Port

FTP-CTRL

TCP Port 21

TCP Setup

FTP Username

USER anonymous

PASS anonymous

PORT 1337

FTP Password

FTP Username

FTP Password

PORT

LIST Password

NLST

DATA Connection

TCP Teardown

Client

Server
**Motivation - Protocol Exploration**

FTP Connection

- Random TCP Port
- FTP-CTRL
- TCP Port 21

TCP Setup

- USER anonymous
- Response 331
- PASS
derp@foocorp.com
- Response 230
- PORT 10.10.10.10 99
- Response 200 - PORT Command Successful

FTP Username

- USER anonymous

FTP Password

- PASS anonymous

PORT

- PORT 1337

LIST Password

- NLST

TCP Teardown

- TIME FOR THAT
Motivation - Structured Inputs

Typical problems:
- State dependent inputs required
- Highly structured input
- Checksums, length fields, etc.
Motivation - Exec Speed

Empty harness

```
$ cat libafl.c

volatile int test;

int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  test += 1;
}
```

Recv-in-a-loop

```
$ cat server.c

for (int i = 0; i < 1000000; i++){
  connfd = accept(sockfd, (SA*)&cli, &len);
  recv(connfd, buf, 4, 0);
  close(connfd);
}
```

```
$ ./fuzzer -i in -o out
Workdir: "/usr/local/google/home/dmk/tmp/LibAFL/fuzzers/fuzzbench"
Out dir at "out" already exists.
Spawning next client (id 0)
First run. Let's set it all up
Let's fuzz :)

Loading file "in/a" ...

[Stats #0] run time: 0h-0m-0s, clients: 1, corpus: 0, objectives: 0, executions: 0, exec/sec: 0
[Testcase #0] run time: 0h-0m-0s, clients: 1, corpus: 1, objectives: 0, executions: 1, exec/sec: 0
[LOG Debug]: Loaded 1 initial testcases.
We imported 1 inputs from disk.

[Stats #0] run time: 0h-0m-15s, clients: 1, corpus: 1, objectives: 0, executions: 3246583, exec/sec: 216413
[Stats #0] run time: 0h-0m-30s, clients: 1, corpus: 1, objectives: 0, executions: 6508710, exec/sec: 216944
[Stats #0] run time: 0h-0m-45s, clients: 1, corpus: 1, objectives: 0, executions: 9774893, exec/sec: 217284
[Stats #0] run time: 0h-1m-0s, clients: 1, corpus: 1, objectives: 0, executions: 13843834, exec/sec: 217304

$ time ./server
Socket successfully created..
Socket successfully binded..
Server listening..
./server 0.08s user 3.15s system 99% cpu 3.252 total
```
**Motivation - Exec Speed**

<table>
<thead>
<tr>
<th>Empty harness</th>
<th>Recv-in-a-loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>● &gt; 200,000 inputs/sec delivered</td>
<td>● ~30,000 inputs/sec delivered</td>
</tr>
<tr>
<td>● Dynamically produced input</td>
<td>● Static input</td>
</tr>
</tbody>
</table>

Input generation outperforms socket interactions almost by an order of magnitude
⇒ Potentially huge performance losses in fuzzing
⇒ Bad scaling due to kernel interactions
Motivation

Recap:
- Exploring state space gets harder with depth
- Structured input generation needs additional insight
- OS network stack is slow
FitM

Fuzzer in the Middle
Solution - Coverage Guided Fuzzers

FTP Connection

- Random TCP Port
- FTP-CTRL TCP Port 21
- TCP Setup
- FTP Username
- USER anonymous
- PASS anonymous
- phill@foocorp.com
- Response 230
- PORT
- Response 200: PORT Command Successful
- LIST Password
- DATA Connection
- TCP Teardown

Client

Server

snapshot

snapshot

snapshot

snapshot

Response 331

Response 200
Available commands of an FTP server (HELP)

ABOR ACCT ALLO APPE CDUP CWD DELE EPRT
EPSV FEAT HELP LIST MDTM MKD MODE NLST
NOOP OPTS PASS PASV PORT PWD QUIT REIN
REST RETR RMD RNFR RNTO SITE SIZE SMNT
STAT STOR STOU STRU SYST TYPE USER XCUP
XCWD XMKD XPWD XRMD
Solution - Slow Network Stack
Overview - Technologies

- **FitM**
- **QEMU**
- **AFL++**

1. **Manage**
   - **CRIU Server**
   - **QEMU Client Bin**
   - **AFL++**

2. **Input/Output**
   - **gen_{2n}**
   - **gen_{2n+1}**

3. **Snapshot**
   - **CRIU Server**
   - **QEMU Server Bin**

References:
[1], [2], [3]
Assumption:

Network-apps

==

"recv-send-recv"-loops
Implementation Model
Implementation Model

FTP Connection

TCP Setup

Random TCP Port

FTP-CTRL

TCP Port 21

FTP Username

User anonymous

Response 331

PASS

phil@foocorp.com

Response 230

PORT 10.10.10.10 99

Response 200 - PORT

Command Successful

LIST Password

TCP Teardown

Client

Server
Implementation Model

FTP Connection

Random TCP Port

FTP-CTRL
TCP Port 21

TCP Setup

FTP Username

FTP Password

PORT

Response 230

Response 331

Response 200 - PORT Command Successful

NLST

LIST Password

TCP Teardown

DATA Connection

Client

Server
Implementation Model

FTP Connection

Random TCP Port

FTP-CTRL
TCP Port 21

TCP Setup

USER anonymous

Response 331

PASS phil@foocorp.com

Response 230

PORT 1.10.10.10 99

Response 200 - PORT Command Successful

NLST

LIST Password

DATA Connection

TCP Teardown

Client

Server

Binary B

Binary A

Corpus

Snapshot

Queue

Filter

gen_{i-1}

gen_{i-2}

gen_{i+1}

gen_{i+2}

Time $t$

State transition
Network Emulation with QEMU

- Deliver input via shmem
- Remove kernel code as much as possible
- Supports sync & async

⇒ Big speedup

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept(4)</td>
<td>fnctl</td>
<td>read</td>
<td>sendmsg</td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td>epoll</td>
<td>recv</td>
<td>write</td>
<td></td>
</tr>
<tr>
<td>clone/fork</td>
<td>(p)poll</td>
<td>recvfrom</td>
<td>socket</td>
<td></td>
</tr>
<tr>
<td>connect</td>
<td>(p)select</td>
<td>recvmsg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dup(2)</td>
<td>exit</td>
<td>send(to)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
○ Designated FD (999) returned by `socket()`
○ Special handling in hooked functions
Hook recv if \( fd == \text{FITM\_FD} \)

**Diagram:**

- **EnteringRecv**
- **Sent once?**
- **MoreData?**
- **Snapshot**
- **Forkserver**
- **Exit**
- **Return**

AFL++ input via ShMem
Evaluation
Number of Executions

<table>
<thead>
<tr>
<th>Fuzzer</th>
<th>Traces</th>
<th>BBs</th>
<th>Hangs</th>
<th>Crashes</th>
<th>Depth</th>
<th>Total Execs</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFLNet</td>
<td>17</td>
<td>5880</td>
<td>22</td>
<td>0</td>
<td>5</td>
<td>424,701</td>
</tr>
<tr>
<td>FitM</td>
<td>146</td>
<td>6158</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>113,683,192</td>
</tr>
</tbody>
</table>

- ~267x faster exec speed from network emulation.
- More protocol levels through snapshotting

- 1 core
- Intel i7-6850K
- 128GB Ram
- LightFTP v2.1 FTP server
- GNU Inteutils v1.9.4 FTP client
- No target patches / harness
Fuzzed Protocol Stages

- Bigger fraction of inputs ends up in later stages of the protocol
- More state exploration
Filtering

- Retained snapshots per cycle trend downwards
- Filtering seems to work
Recap
FitM Building blocks

- Init Runs
- Gen0 Snap
- Gen1 Snap
- Gen2 Snap
- GenN Snap

Namespace:
- Fuzzer
- Minimizer
- Output Collector

Shmem Input:
- CRIU
- QEMU

Target N:
- Forkserver
- Net Emu

New Snapshots

recv
send
We created FitM, a fuzzer for network interactions
We fuzz client and server at the same time
FitM emulates binaries and the network layer
It uses snapshotting to reach deeper protocol states
Open Source at https://github.com/fgsect/fitm
while (questions());

char buf[16];
strncpy(buf, ""
   "Thank you for your attention."
   "\n", sizeof(buf));
printf("%s", buf);
FITM_FD - socket

○ Check if TCP socket
○ Check if initial socket calls should be skipped
○ Return FITM_FD
FITM_FD - send

Only if fd is FITM_FD

- Touch AFL_MAP: Emphasize send paths
- CREATE_OUTPUTS:
  - Env variable to control output generation
  - If true, write output to snapshot-specific file
  - If false, return len from args
FITM_FD - recv

Only if fd is FITM_FD

- Touch AFL_MAP: Emphasize read paths
- Check if previously called send()
- TIMEWARP_MODE:
  - Env variable to control snapshot generation
  - If true, snapshot and exit
  - If false, exit
- Start AFL forkserver
- Read input from file provided by FitM
FTP Control Connection

- Random TCP Port
- FTP-CTRL TCP Port 21
- TCP Handshaking: Client to Server

FTP Username
- USER anonymous
- Response 331

FTP Password
- PASS phil@foocorp.com
- Response 230

PORT
- PORT 10.10.10.10 99
- Response 200 - PORT Command Successful
- LIST Password

NLST
- Response 230

DATA Connection Initiated

TCP Teardown

FTP Connection

- Random TCP Port
- FTP-CTRL TCP Port 21
- TCP Setup

FTP Username
- USER anonymous
- Response 331

FTP Password
- PASS phil@foocorp.com
- Response 230

PORT
- PORT 10.10.10.10 99
- Response 200 - PORT Command Successful
- LIST Password

NLST

DATA Connection

TCP Teardown
Available commands of an FTP server (HELP)

ABOR ACCT ALLO APPE CDUP CWD DELE EPRT
EPSV FEAT HELP LIST MDTM MKD MODE NLST
NOOP OPTS PASS PASV PORT PWD QUIT REIN
REST RETR RMD RNFR RNTO SITE SIZE SMNT
STAT STOR STOU STRU SYST TYPE USER XCP
XCWD XMKD XPWD XRMD
Challenge #1 - State Explosion

State Explosion: Each state has >1 children

Tackle with:

- AFL-cmin
- Random select
- String distance filter on "produced output"
Challenge #2 - Dead Ends

Dead Ends: error states, setup/teardown states

Tackle with:

Random Restarts
Challenge #3 - Desync

Desync between generations

Tackle with:

Cross generational input