American Fuzzy Lop

Alex Wilson
**What is fuzzing?**

<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time: 0 days, 0 hrs, 0 min, 54 sec</td>
<td>Cycles done: 0</td>
</tr>
<tr>
<td>Last new path: 0 days, 0 hrs, 0 min, 30 sec</td>
<td>Total paths: 7</td>
</tr>
<tr>
<td>Last uniq crash: 0 days, 0 hrs, 0 min, 26 sec</td>
<td>Uniq crashes: 1</td>
</tr>
<tr>
<td>Last uniq hang: none seen yet</td>
<td>Uniq hangs: 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now processing: 0 (0.00%)</td>
<td>Map density: 0.06% / 0.07%</td>
</tr>
<tr>
<td>Paths timed out: 0 (0.00%)</td>
<td>Count coverage: 1.00 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Findings in depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now trying: 0 (0.00%)</td>
<td>Levels: 2</td>
</tr>
<tr>
<td>Stage execs: 2109 (25%)</td>
<td>Pending: 7</td>
</tr>
<tr>
<td>Total execs: 2109</td>
<td>Pend fav: 1</td>
</tr>
<tr>
<td>Exec speed: 39.26/sec (slow!)</td>
<td>Own finds: 6</td>
</tr>
<tr>
<td>Fuzzing strategy yields</td>
<td>Imported: n/a</td>
</tr>
<tr>
<td>bit flips: 1/48, 0/47, 1/45</td>
<td>Stability: 100.00%</td>
</tr>
<tr>
<td>byte flips: 0/6, 0/5, 0/3</td>
<td></td>
</tr>
<tr>
<td>arithmetics: 1/336, 0/93, 0/0</td>
<td></td>
</tr>
<tr>
<td>Known ints: 0/30, 0/130, 0/132</td>
<td></td>
</tr>
<tr>
<td>Dictionary: 0/0, 0/0, 0/0</td>
<td></td>
</tr>
<tr>
<td>HAVOC: 0/0, 0/0</td>
<td></td>
</tr>
<tr>
<td>Trim: 0.00%</td>
<td></td>
</tr>
</tbody>
</table>

[cpu: 320%]
Vocabulary

- Corpus
- Minimize
- Reproducibility
- White-box
- Gray-box
  - Instrumentation
- Black-box
<table>
<thead>
<tr>
<th>process timing</th>
<th>overall results</th>
</tr>
</thead>
</table>
| run time       | cycles done:
|                | 0 days, 0 hrs, 0 min, 54 sec |
|                 | total paths: 7 |
| last new path  | uniq crashes: 1 |
| 0 days, 0 hrs, 0 min, 30 sec |
| last uniq crash| uniq hangs: 0 |
| 0 days, 0 hrs, 0 min, 26 sec |
| last uniq hang | none seen yet |

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
</tr>
</thead>
</table>
| now processing | map density:
| 0 (0.00%) | 0.06% / 0.07% |
| paths timed out | count nodes |
| 0 (0.00%) | 1.000 |
| stage progress | new edges on:
| now trying: havoc |
| stageness | 100.00% |
| stage execs: 1228/1632 | (75.25%) |
| total execs: 2169 |
| exec speed: 39.28 sec |
| fuzzer: havoc |
| bit flips: 1/49, 0/47, 1/43 |
| byte flips: 0/0, 0/5, 0/3 |
| arithmetics: 1/336, 0/93, 0/0 |
| known ints: 0/30, 0/136, 0/132 |
| dictionary: 0/0, 0/0, 0/0 |
| havoc: 0/0, 0/0 |
| trim: 0.00%/1, 0.00% |

What is AFL?

lcamtuf.coredump.cx
Basic AFL fuzzing workflow

- Compile using afl compiler wrapper
- Seed with (minimized) test cases
- Run afl
- Minimize, re-seed, re-run
- Triage crashes
- Evaluate coverage
AFL limitations

<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time</td>
<td>Cycles done</td>
</tr>
<tr>
<td>Last new path</td>
<td>Total paths</td>
</tr>
<tr>
<td>Last uniq crash</td>
<td>Uniq crashes</td>
</tr>
<tr>
<td>Last uniq hang</td>
<td>Uniq hangs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now processing</td>
<td>Map density</td>
</tr>
<tr>
<td>Paths timed out</td>
<td>Count coverage</td>
</tr>
<tr>
<td>Stage progress</td>
<td>Findings for depth</td>
</tr>
<tr>
<td>Now trying</td>
<td>Findings depth</td>
</tr>
<tr>
<td>Stage execs</td>
<td>Findings depth</td>
</tr>
<tr>
<td>Total execs</td>
<td>Findings depth</td>
</tr>
<tr>
<td>Exec speed</td>
<td>Findings depth</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzing Strategy Yields</th>
<th>Path Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit flips</td>
<td>Levels</td>
</tr>
<tr>
<td>Byte flips</td>
<td>Pending</td>
</tr>
<tr>
<td>Arithmetics</td>
<td>Pend fav</td>
</tr>
<tr>
<td>Known ints</td>
<td>Own finds</td>
</tr>
<tr>
<td>Dictionary</td>
<td>Imported</td>
</tr>
<tr>
<td>Havoc</td>
<td>Stability</td>
</tr>
<tr>
<td>Trim</td>
<td></td>
</tr>
</tbody>
</table>
Why not afl-unicorn?

- Execution speed!
- Requires a debugger to dump program state
- Requires reverse engineering
  - Must locate and redirect troublesome functions
  - Must write input into emulated memory
  - Must write a test harness
afl-unicorn

- afl-unicorn mutates input
- afl-unicorn forks test harness
- Test harness loads input
- Test harness executes binary section via UE
- Harness’s UE provides coverage to afl-unicorn
What is a Unicorn Engine (harness)?

- Allows emulating raw binary code

```c
uc_hook_add (uc, &h, UC_HOOK_CODE, hook_instr, NULL, 1, 0);
uc_mem_map  (uc, addr, size, UC_PROT_ALL);
uc_mem_write(uc, addr, data, sizeof(data));
uc_reg_write(uc, UC_X86_REG_RCX, &rcx);
uc_reg_read (uc, UC_X86_REG_RIP, &rip);
uc_emu_start(uc, emu_start_addr, emu_end_addr, 0);
```
How to afl-unicorn

- Platform/QEMU
- Target image
- gdb server
- Target binary
- Input
- Unicorn harness
- gdb
- Target binary
How to afl-unicorn

Platform/QEMU
- Target image
  - gdb server
  - Target binary
  - Input

Unicorn harness

Execute

GDB server

Target binary

Break
How to afl-unicorn

Platform/QEMU

Target image

Target binary

State

Input

Unicorn harness

Target binary
How to afl-unicorn

Platform/QEMU
- Target image
  - gdb server
    - Target binary
- Unicorn harness
  - State
    - Input

Target binary
How to afl-unicorn

<table>
<thead>
<tr>
<th>Process timing</th>
<th>Overall results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time: 0 days, 0 hrs, 0 min, 54 sec</td>
<td>cycles done: 0</td>
</tr>
<tr>
<td>last new path: 0 days, 0 hrs, 0 min, 30 sec</td>
<td>total paths: 7</td>
</tr>
<tr>
<td>last uniq crash: 0 days, 0 hrs, 0 min, 26 sec</td>
<td>uniq crashes: 1</td>
</tr>
<tr>
<td>last uniq hang: none seen yet</td>
<td>uniq hangs: 0</td>
</tr>
<tr>
<td>cycle progress: none seen yet</td>
<td>map coverage: 0.06% / 0.07%</td>
</tr>
<tr>
<td>now processing: 0 (0.00%)</td>
<td>map density: 1.00 bits/tuple</td>
</tr>
<tr>
<td>paths timed out: 0 (0.00%)</td>
<td>flushing in depth: 0</td>
</tr>
</tbody>
</table>

Stage progress:
- havoc: 1228/1632 (75.3)
- total execs: 2169
- exec speed: 39.26/sec (slow)

Fuzzing strategy yields:
- bit flips: 1/48, 0/47, 1/4
- byte flips: 0/6, 0/5, 0/3
- arithmetics: 1/336, 0/93, 0
- known ints: 0/30, 0/136, 0
- dictionary: 0/0, 0/0
- havoc: 0/0, 0
- trim: 0.00%/1, 0.00%
How to afl-unicorn
How to afl-unicorn

state
input

Unicorn harness v2

execute

input

state

input
# How to afl-unicorn

<table>
<thead>
<tr>
<th>Process Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time : 0 days, 0 hrs, 0 min, 54 sec</td>
</tr>
<tr>
<td>last new path : 0 days, 0 hrs, 0 min, 30 sec</td>
</tr>
<tr>
<td>last uniq crash : 0 days, 0 hrs, 0 min, 26 sec</td>
</tr>
<tr>
<td>last uniq hang : none seen yet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing : 0 (0.00%)</td>
</tr>
<tr>
<td>paths timed out : 0 (0.00%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>now trying : havoc</td>
</tr>
<tr>
<td>stage execs : 1228/1632 (75.2%)</td>
</tr>
<tr>
<td>total execs : 2169</td>
</tr>
<tr>
<td>exec speed : 39.26/sec (slow)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unicorn harness v2</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
</tr>
<tr>
<td>Input</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
</table>

| Fail : ( |
How to afl-unicorn
How to afl-unicorn

Unicorn harness v3

State
Input
Input
Input
How to afl-unicorn

Unicorn harness v3

State

Input

Input

Input
How to afl-unicorn

Unicorn harness v3

State

Input

Input

Mutated inputs

Mutated inputs

Crash cases

Crash cases
How to afl-unicorn

Unicorn harness v3
State
Input
Input
Input

Target
binary

Mutated inputs
Mutated inputs
Mutated inputs
Mutated inputs
Crash cases
Crash cases
Crash cases
Crash cases
How to afl-unicorn

Unicorn harness v3

State

Crash case

Target binary

Real target

Dump trace / final state

Triage crash state
Fuzzing world

- AFL and derivatives
- Peach
- Sulley/Boofuzz
- Radamsa
- Custom fuzzers