AFL – American Fuzzy Lop

A short introduction
by Tobias Ospelt, March, 9th 2015
Silicon Valley Fuzzers, Fuzzing meetup,
Santa Clara, CA
Me

- Penetration Tester (usually CH, DE, UK, once in the USA)
- Android stuff, mona.py unicode alignment, tincd metasploit module, started fuzzing
  – floyd.ch / @floyd_ch
- AFL user (not an expert on all the internals)
• 6 IT security experts
• Keykeriki, Backtrack, Degate, remote-exploit.org, Die Datenkrake, Analysis of the German state trojan
• We do all areas of technical HW & SW security analysis (Penetration Testing, Crypto, Web, medical devices, etc.)
AFL – American Fuzzy Lop

• Fuzzer developed by Michal Zalewski (lcamtuf), Project Zero, Google
  – He's on holiday today 😞
• http://lcamtuf.coredump.cx/afl/
• "Under certain conditions you are crazy if you don't use AFL for your project" - me
Why use AFL?
It finds bugs

IJG jpeg 1 libjpeg-turbo 1 2 libpng 1 libtiff 1 2 3 4 5 mozjpeg 1 libbpg
Mozilla Firefox 1 2 3 4 5 Google Chrome 1 Internet Explorer 1 2 (3) (4)
LibreOffice 1 2 3 4 poppler 1 freetype 1 2 GnuTLS 1 GnuPG 1 2 (3) OpenSSH
1 2 3 bash (post-Shellshock) 1 2 tcpdump 1 2 3 4 5 6 7 8 Adobe Flash / PCRE 1
2 JavaScriptCore 1 2 3 4 pdium 1 ffmpeg 1 2 3 4 5 6 7 8 wireshark 1 ImageMagick
3 4 5 6 ... lcms (1) PHP 1 2 3 lame 1 ... FLAC audio library 1 2 libsndfile 1 2 3 less / lesspipe 1 2 3 strings (+ related
tools) 1 2 3 4 5 6 7 file 1 2 dpkg 1 rcs 1 systemd-resolved 1 2 sqlite 1 2 3
libyaml 1 Info-Zip unzip 1 2 OpenBSD pfctl 1 NetBSD bpf 1 man &
mmandoc 1 2 3 4 5 ... IDA Pro clamav 1 2 libxml2 1 glibc 1 clang / llvm 1 2 3 4 5
6 nasm 1 2 ctags 1 mutt 1 procmail 1 fontconfig 1 pdksh 1 2 Qt 1 wavpack 1
redis / lua-cmsgpack 1 taglib 1 2 3 privoxy 1 perl 1 2 3 4 5 6 libxmp radare2
1 2 fwknop metacam 1 exifprobe 1 capnproto 1
It's spooky

- Michal gave djpeg (IJG jpeg library) to AFL
- Plus a non-jpeg file as an input
  - $ echo 'hello' >in_dir/hello
- AFL started to produce valid jpeg files after a day or two
More reasons

• It's dead simple
• No configuration of AFL necessary, robust
• It's cutting edge
• It's fast
• Produces very very good input files (corpus) that can be used in other fuzzers
• Many targets that were never touched by AFL (and it will crush them)
When I read through lcatumf’s post on ‘less’ and ‘strings’
And because you will go
You won't believe what you are reading

- Source: http://lcamtuf.coredump.cx/afl/demo/
- afl-generated, minimized image test sets (partial) [...] 
- JPEG XR jxrlib 1.1 JxrDecApp¹ IE → Ditched ² 
- ² Due to the sheer number of exploitable bugs that allow the fuzzer to jump to arbitrary addresses.
When to use AFL
The usual use case

- You have the source code and you compile with gcc or clang
- Your are on 32bit or 64bit on Linux/OSX/BSD
- The to-be-fuzzed code (e.g. parser) reads it's input from stdin or from a file
- The input file is usually only max. 10kb
- This covers *a lot* of Linux libraries
What if something does not apply?

• No source code?
  – Try the experimental QEMU instrumentation

• Not on 32/64 bit?
  – There is an experimental ARM version

• Not reading from stdin or file?
  – Maybe your project has a utility command line tool that does read from file
  – Or you write a wrapper to do it
  – Same if you want to test (parts of) network protocol parsers
How to use AFL
Steps of fuzzing

1. Compile/install AFL (once)
2. Compile target project with AFL
   – afl-gcc / afl-g++ / afl-clang / afl-clang++ / (afl-as)
3. Chose target binary to fuzz in project
   – Chose its command line options to make it run fast
4. Chose valid input files that cover a wide variety of possible input files
   – afl-cmin / (afl-showmap)
Steps of fuzzing

5. Fuzzing
   – afl-fuzz

6. Check how your fuzzer is doing
   – command line UI / afl-whatsup / afl-plot / afl-gotcpu

7. Analyze crashes
   – afl-tmin / triage_crashes.sh / peruvian were rabbit
   – ASAN / valgrind / exploitable gdb plugin / ...

8. Have a lot more work than before
   – CVE assignment / responsible disclosure / ...
#!/bin/bash
#Download & compile new AFL version:
wget http://lcamtuf.coredump.cx/afl.tgz
tar xfz afl.tgz
rm afl.tgz
cd `find . -type d -iname "afl-*"|sort|head -1`
make
echo "Provide sudo password for sudo make install"
sudo make install
This is a helper application for afl-fuzz. It serves as a drop-in replacement for gcc or clang, letting you recompile third-party code with the required runtime instrumentation.
Instrumenting a project (step 2) – example: libtiff from CVS repository

```
/opt/libtiff-cvs-afl$ export CC=afl-gcc
/opt/libtiff-cvs-afl$ export CXX=afl-g++
/opt/libtiff-cvs-afl$ ./configure --disable-shared
/opt/libtiff-cvs-afl$ make clean
/opt/libtiff-cvs-afl$ make
```
Choosing the binary to fuzz (step 3) – they are all waiting for it

/opt/libtiff-cvs-afl$ ./tools/

bmp2tiff  fax2tiff  ppm2tiff  raw2tiff
thumbnail  tiff2pdf  tiff2rgba  tiffcp
tiffdither  tiffinfo  tiffset  fax2ps
gif2tiff  pal2rgb  ras2tiff  rgb2ycbcr
tiff2bw  tiff2ps  tiffcmp  tiffcrop
tiffdump  tiffmedian  tiffspli
Chose initial input files (step 4)

```
/opt/libtiff-cvs-afl$ mkdir input_all
/opt/libtiff-cvs-afl$ scp host:/bmps/ input_all/
/opt/libtiff-cvs-afl$ ls -l input_all | wc -l
886
```
Chose initial input files (step 4)

/opt/libtiff-cvs-afl$ afl-cmin -i input_all -o input -- /opt/libtiff-cvs-afl/tools/bmp2tiff @@ /dev/null

corpus minimization tool for afl-fuzz by <lcamtuf@google.com>

[*] Testing the target binary...
[+] OK, 191 tuples recorded.
[*] Obtaining traces for input files in 'input_all'...

Processing file 886/886...
[*] Sorting trace sets (this may take a while)...
[+] Found 4612 unique tuples across 886 files.
[*] Finding best candidates for each tuple...

Processing file 886/886...
[*] Sorting candidate list (be patient)...
[*] Processing candidates and writing output files...

Processing tuple 4612/4612...
[+] Narrowed down to 162 files, saved in 'input'.

Chose initial input files (step 4)

/opt/libtiff-cvs-afl$ ls -l input | wc -l
162
Fuzzing (step 5)

```
/opt/libtiff-cvs-afl$ screen -S fuzzing
/opt/libtiff-cvs-afl$ afl-fuzz -i input -o output
-- /opt/libtiff-cvs-afl/tools/bmp2tiff @@ /dev/null
```
How is our fuzzer doing? (step 6)

<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time</td>
<td>Cycles done: 0</td>
</tr>
<tr>
<td>Last new path</td>
<td>Total paths: 193</td>
</tr>
<tr>
<td>Last uniq crash</td>
<td>Uniq crashes: 2</td>
</tr>
<tr>
<td>Last uniq hang</td>
<td>Uniq hangs: 15</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: 1344 (2.05%)</td>
</tr>
<tr>
<td>Paths timed out</td>
<td>Count coverage: 3.53 bits/tuple</td>
</tr>
<tr>
<td>Stage progress</td>
<td>Findings in depth</td>
</tr>
<tr>
<td>Now trying:</td>
<td>Favored paths: 68 (35.23%)</td>
</tr>
<tr>
<td>Auto extras (over)</td>
<td>New edges on: 79 (40.93%)</td>
</tr>
<tr>
<td>Stage execs:</td>
<td>Total crashes: 19 (2 unique)</td>
</tr>
<tr>
<td>15/72 (20.83%)</td>
<td>Total hangs: 100 (15 unique)</td>
</tr>
<tr>
<td>Total execs:</td>
<td>Path geometry</td>
</tr>
<tr>
<td>86.9k</td>
<td>Levels: 2</td>
</tr>
<tr>
<td>Exec speed:</td>
<td>Pending: 190</td>
</tr>
<tr>
<td>71.11/sec (slow!)</td>
<td>Pend fav: 65</td>
</tr>
<tr>
<td>Fuzzing strategy yields</td>
<td>Own finds: 31</td>
</tr>
<tr>
<td>Bit flips:</td>
<td>Imported: n/a</td>
</tr>
<tr>
<td>12/704, 1/700, 1/692</td>
<td>Variable: 0</td>
</tr>
<tr>
<td>Byte flips:</td>
<td></td>
</tr>
<tr>
<td>0/88, 0/84, 0/76</td>
<td></td>
</tr>
<tr>
<td>Arithmetics:</td>
<td></td>
</tr>
<tr>
<td>4/4840, 0/4068, 0/2495</td>
<td></td>
</tr>
<tr>
<td>Known INTs:</td>
<td></td>
</tr>
<tr>
<td>1/404, 1/2333, 2/2842</td>
<td></td>
</tr>
<tr>
<td>Dictionary:</td>
<td></td>
</tr>
<tr>
<td>0/0, 0/0, 0/16</td>
<td></td>
</tr>
<tr>
<td>Havoc:</td>
<td></td>
</tr>
<tr>
<td>9/65.6k, 0/0</td>
<td></td>
</tr>
<tr>
<td>Trim:</td>
<td></td>
</tr>
<tr>
<td>8.33%/20, 0.00%</td>
<td></td>
</tr>
</tbody>
</table>

[CPU: 316%]
How is our fuzzer doing? (step 6)
How is our fuzzer doing? (step 6)

```
- process timing
  run time     : 0 days, 1 hrs, 27 min, 43 sec
  last new path: 0 days, 0 hrs, 28 min, 27 sec
  last uniq crash: 0 days, 0 hrs, 31 min, 10 sec
  last uniq hang : 0 days, 0 hrs, 29 min, 29 sec

- cycle progress
  now processing: 57 (20.28%)
  paths timed out: 0 (0.00%)

- stage progress
  now trying: arith 32/8
  stage execs: 3480/18.9k (18.37%)
  total execs: 938k
  exec speed: **18.23/sec (zzzz...)**

- fuzzing strategy yields
  bit flips: 40/24.8k, 4/24.7k, 4/24.7k
  byte flips: 0/3096, 0/2554, 1/2654
  arithmetics: 22/137k, 6/110k, 0/62.2k
  known ints: 0/10.5k, 6/67.6k, 17/97.3k
  dictionary: 0/0, 0/0, 3/6243
  havoc: 55/356k, 0/0
  trim: 14.63%/1266, 18.73%

- overall results
  cycles done: 0
  total paths: 281
  uniq crashes: 44
  uniq hangs: 76

- map coverage
  map density: 1375 (2.10%)
  count coverage: 3.67 bits/tuple

- findings in depth
  favored paths: 95 (33.81%)
  new edges on: 104 (37.01%)
  total crashes: 427 (44 unique)
  total hangs: 4681 (76 unique)

- path geometry
  levels: 2
  pending: 252
  pend fav: 72
  own finds: 119
  imported: n/a
  variable: 0
```

[cpu: 304%]
How is our fuzzer doing? (step 6)

/opt/libtiff-cvs-afl$ afl-gotcpu
afl-gotcpu 1.56b (Mar 9 2015 02:50:32) by <lcamtuf@gmail.com>
[*] Measuring preemption rate (this will take 5.00 sec)...
[+] Busy loop hit 79 times, real = 5001 ms, slice = 2448 ms.
>>> FAIL: Your CPU is overbooked (204%). <<<
How is our fuzzer doing? (step 6)

- `afl-plot`
How is our fuzzer doing? (step 6)

- `afl-plot`
Other examples
Crash analysis (step 7)
minimizing crash input

```
afl-tmin 1.56b (Mar 9 2015 02:50:31) by <lcamtuf@google.com>
[+] Read 36 bytes from 'output/crashes/id:000000,sig:11,src:000003,op:int16,pos:21,val:+1'.
[*] Performing dry run (mem limit = 25 MB, timeout = 1000 ms)...
[+] Program exits with a signal, minimizing in crash mode.
[*] --- Pass #1 ---
[*] Stage #1: Removing blocks of data...
Block length = 2, remaining size = 36
Block length = 1, remaining size = 34
[...]
```
Crash analysis (step 7)
minimizing malicious input


/opt/libtiff-cvs-afl$ ls -als minimized-crash 4 -rw-------- 1 user user 34 Mär 9 05:51 minimized-crash
Crash analysis (step 7)
example of manual analysis

uncompr_size = width * length;
...
uncomprbuf = (unsigned char *)_TIFFmalloc(uncompr_size);

(gdb) p width
$70 = 65536
(gdb) p length
$71 = 65544
(gdb) p uncompr_size
$72 = 524288

524289 is (65536 * 65544) % MAX_INT
Crash analysis (step 7)
peruvian were-rabbit
Crash analysis (step 7)

peruvian were-rabbit

• Using crashes as inputs, mutate them to find different crashes (that AFL considers "unique")

/opt/libtiff-cvs-afl$ afl-fuzz -i output/crashes/ -o peruvian_crashes -C /opt/libtiff-cvs-afl/tools/bmp2tiff @@ /dev/null
Crash analysis (step 7)
peruvian were-rabbit

<table>
<thead>
<tr>
<th>process timing</th>
<th>overall results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time : 0 days, 0 hrs, 3 min, 3 sec</td>
<td>cycles done : 0</td>
</tr>
<tr>
<td>last new path : 0 days, 0 hrs, 0 min, 21 sec</td>
<td>total paths : 170</td>
</tr>
<tr>
<td>last uniq crash : 0 days, 0 hrs, 0 min, 20 sec</td>
<td>uniq crashes : 34</td>
</tr>
<tr>
<td>last uniq hang : 0 days, 0 hrs, 0 min, 0 sec</td>
<td>uniq hangs : 29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cycle progress</th>
<th>map coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>now processing : 1 (0.59%)</td>
<td>map density : 816 (1.25%)</td>
</tr>
<tr>
<td>paths timed out : 0 (0.00%)</td>
<td>count coverage : 3.39 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 : havoc</td>
<td>favored paths : 30 (17.65%)</td>
</tr>
<tr>
<td>stage execs : 47.5k/60.0k (79.16%)</td>
<td>new edges on : 52 (30.59%)</td>
</tr>
<tr>
<td>total execs : 57.7k</td>
<td>new crashes : 7987 (34 unique)</td>
</tr>
<tr>
<td>exec speed : 374.1/sec</td>
<td>total hangs : 369 (29 unique)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fuzzing strategy yields</th>
<th>path geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>arithmetics : 19/1981, 3/1919, 0/1227</td>
<td>pend fav : 30</td>
</tr>
<tr>
<td>known ints : 0/162, 8/944, 4/1252</td>
<td>own finds : 82</td>
</tr>
<tr>
<td>dictionary : 0/0, 0/0, 0/32</td>
<td>imported : n/a</td>
</tr>
<tr>
<td>havoc : 0/0, 0/0</td>
<td>variable : 2</td>
</tr>
<tr>
<td>trim : 0.00%/8, 0.00%</td>
<td></td>
</tr>
</tbody>
</table>
You want more?

• Try it yourself
• Subscribe to afl-users
• Convince Michal to come here
int 3

- Twitter: @floyd_ch
- http://floyd.ch
- tobias@modzero.ch
- Twitter: @mod0
- http://www.modzero.ch