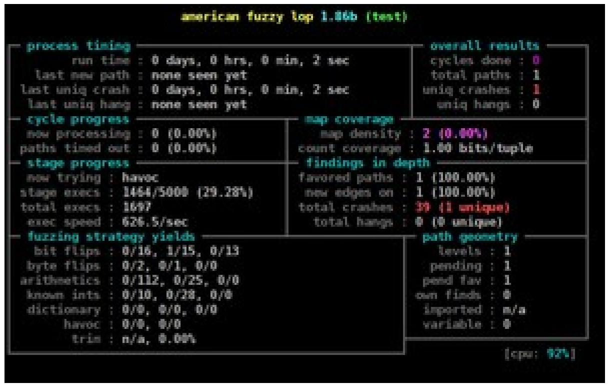
# Fuzzing on Golem

**Ladislav Sládeček**

Golem Mainnet Hackathon @ GR9 - Open Track



```
                  american fuzzy lop 1.86b (test)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 0 min, 2 sec        │  cycles done : 0      │
│   last new path : none seen yet                      │  total paths : 1      │
│ last uniq crash : 0 days, 0 hrs, 0 min, 2 sec        │ uniq crashes : 1      │
│  last uniq hang : none seen yet                      │   uniq hangs : 0      │
├─ cycle progress ────────────────────┬─ map coverage ─┴──────────────────────┤
│  now processing : 0 (0.00%)         │    map density : 2 (0.00%)            │
│ paths timed out : 0 (0.00%)         │ count coverage : 1.00 bits/tuple      │
├─ stage progress ────────────────────┼─ findings in depth ───────────────────┤
│  now trying : havoc                 │ favored paths : 1 (100.00%)           │
│ stage execs : 1464/5000 (29.28%)    │  new edges on : 1 (100.00%)           │
│ total execs : 1697                  │ total crashes : 39 (1 unique)         │
│  exec speed : 626.5/sec             │   total hangs : 0 (0 unique)          │
├─ fuzzing strategy yields ───────────┴───────────────┬─ path geometry ───────┤
│   bit flips : 0/16, 1/15, 0/13                      │    levels : 1         │
│  byte flips : 0/2, 0/1, 0/0                         │   pending : 1         │
│ arithmetics : 0/112, 0/25, 0/0                      │  pend fav : 1         │
│  known ints : 0/10, 0/28, 0/0                       │ own finds : 0         │
│  dictionary : 0/0, 0/0, 0/0                         │  imported : n/a       │
│       havoc : 0/0, 0/0                              │  variable : 0         │
│        trim : n/a, 0.00%                            ├───────────────────────┘
└─────────────────────────────────────────────────────┘             [cpu: 92%]
```

## Introduction

- My second CI/testing project on golem.
- The first one was about unit testing with RUST (golem-cargo-test).
- This time I am exploring a FUZZ-TESTING framework for golem.
- More useful (I hope) – an ideal application for cloud computing.
- Harder.
- Has also a potential “public good aspect”.

## What is Fuzz-testing?

- Advanced testing technique.
- The software is run many times fed by random-like inputs until it crashes.
- Cunningly constructed inputs to increase crash probability
  - Genetic algorithms (PhD level theory)
  - Software compiled with instrumentation to trace code coverage
- With memory unsafe languages (C, C++) a crash usually means a security hole.
- With memory safe languages a crash represents at least a DOS attack.
- Usually used for finished products.

## Example – input generated by afl:

- @@@w�7@@@A0+a'77@A0 +a'77�77777N7777XXXXXXXXXX7@@@@@@@@@@777a'77@A0+a'7777777 7N7777XXX77N7777777w70+a'77@A0+a'7�7777�77�
- Crashes libxml2.
- No human tester will likely invent this!

## Is Fuzz-testing usefull?

- https://lcamtuf.coredump.cx/afl/ claims that the **afl** fuzzing tool alone has found security bugs in:
- IJG jpeg libjpeg-turbo libpng libtiff mozjpeg PHP Mozilla FirefoxInternet Explorer Apple Safari Adobe Flash / PCRE sqlite OpenSSLLibreOffice poppler freetype GnuTLS GnuPG OpenSSH PuTTY ntpd nginxbash (post-Shellshock) tcpdump JavaScriptCore pdfium ffmpeglibmatroska libarchive wireshark ImageMagick BIND QEMU lcms OracleBerkeleyDB Android / libstagefright iOS / ImageIO FLAC audio librarylibsndfile less / lesspipe strings (+ related tools) file dpkg rcssystemd-resolved libyaml Info-Zip unzip libtasn1 OpenBSD pfctl NetBSDbpf man & mandoc IDA Pro [reported by authors] clamav libxml2 glibcclang / llvm nasm ctags mutt procmail fontconfig pdksh Qt wavpackredis / lua-cmsgpack taglib privoxy perl libxmp radare2 SleuthKitfwknop [reported by author] X.Org exifprobe jhead [?] capnprotoXerces-C metacam djvulibre exiv Linux btrfs Knot DNS curlwpa_supplicant libde26 [reported by author] dnsmasq libbpg (1) lamelibwmf uudecode MuPDF imlib2 libraw libbson libsass yara W3C tidy-htmlVLC FreeBSD syscons John the Ripper screen tmux mosh UPX indentopenjpeg MMIX OpenMPT rxvt dhcpcd Mozilla NSS Nettle mbed TLS Linuxnetlink Linux ext4 Linux xfs botan expat Adobe Reader libav libicalOpenBSD kernel collectd libidn MatrixSSL jasper MaraDNS w3m XenOpenH232 irssi cmark OpenCV Malheur gstreamer Tor gdk-pixbuf audiofilezstd lz4 stb cJSON libpcre MySQL gnulib openexr libmad ettercap lrzipfreetds Asterisk ytnef raptor mpg123 Apache httpd exempi libgmime pevLinux mem mgmt sleuthkit Mongoose OS iOS kernel
- and other  bugs in:
- v8, sed, awk, make, m4, yacc, PHP, ImageMagick, freedesktop.org, patch, libtasn1, libvorbis, zsh, lua, ninja, ruby, busybox, gcrypt, vim, Tor, poppler, libopus, BSD sh, gcc, qemu, w3m, zsh, dropbear, libtorrent, git, rust, gravity, e2fsprogs, etc); found security issues in all sorts of less-widespread software (e.g., parrot, lodepng, json-glib, cabextract, libmspack, qprint, gpsbabel, dmg2img, antiword, arj, unrar, unace, zoo, rzip, lrzip, libiso*, libtta, duktape, splint, zpaq, assimp, cppcheck, fasm, catdoc, pngcrush, cmark, p7zip, libjbig2, aaphoto, t1utils, apngopt, sqlparser, mdp, libtinyxml, freexl, bgpparser, testdisk, photorec, btcd, gumbo, chaiscript, teseq, colcrt, pttbbs, capstone, dex2oat, pillow, elftoolchain, aribas, universal-ctags, uriparser, jq, lha, xdelta, gnuplot, libwpd, teseq, cimg, libiberty, policycoreutils, libsemanage, renoise, metapixel, openclone, mp3splt, podofo, glslang, UEFITool, libcbor, lldpd, pngquant, muparserx, mochilo, pyhocon, sysdig, Overpass-API, fish-shell, gumbo-parser, mapbox-gl-native, rapidjson, libjson, FLIF, MultiMarkdown, astyle, pax-utils, zziplib, PyPDF, spiffing, apk, pgpdump, icoutils, msitools, dosfstools, schoco, MojoShader,

## Fuzzing on clouds

- A typical fuzzing session takes days.
- Resource intensive.
- Pays itself (security is not cheap, security bounties).
- There is a fuzzing framework for each major cloud platform:
  - GOOGLE CLUSTER FUZZ https://google.github.io/clusterfuzz/
  - AZURE ONE FUZZ https://github.com/microsoft/onefuzz
  - AWS
- Has a “**public good” aspect** – security testing of free independent software (google offers free https://github.com/google/oss-fuzz which has found 25000 bugs in 375 OSS projects)

## Fuzzing on Golem

- My project **https://github.com/sladecek/golem-afl**
- Using the **afl** open source fuzzing framework on golem.
- Just a first steps / exploration (a polished product would require years of work).
- So far works for two simple demos.
- Instructions for evaluation are in the README.md file (**default** demo)
- In this presentation we’ll go through another demo: **fuzzgoat**

## 1. afl

- **https://github.com/google/AFL**
- (the best) open source fuzzing framework for C/C++
- There is a fork **https://github.com/AFLplusplus/AFLplusplus**
- Contains:
  - Instrumentation (modified compilers such as afl-gcc or afl-clang)
  - Main binary afl-fuzz – runs instrumented binary
  - Auxiliary tools – investigate crashes, statistics, graphs…
- Principle of work:
  - Input directory contains a collection of typical inputs.
  - Output directory contains result – the main product are **crashes – files that contains an input that crashes the app**,
  - There is a queue directory containing paths through the software to be investigated
  - The genetic algorithm takes the paths from the queue, evaluates them and creates new paths.
- Both versions (afl, afl++) have been inserted into a custom golem image

## 2. image

- **https://github.com/sladecek/golem-afl/docker**
- *golemfactory/base+python+both the fuzzerd*
- Dockerised, converted to gvmi format, uploaded
-

  image_hash="3fc93eca1f720bb7a2d6fa6394a03b883b28ff3fbe3d768f6f858924",

## 3. fuzzgoat

- Simple application – used as a fuzzer benchmark
- **https://github.com/fuzzstati0n/fuzzgoat**
- Compiled by afl-gcc
- CC=../AFL/afl-gcc make
- Stored in prj/fuzzgoat directory

## 4. run

- Running (on yagna) like this:
- **YAGNA_APPKEY=b32088f8c6a442b1810980b454ffea22 YAGNA_API_URL=http://172.19.0.6:6000 GSB_URL=tcp://172.19.0.6:6010 python golem-afl.py --nodes=10 --prj=fuzzgoat --run-time=15 --cycle=10 --subnet=goth**
- Executes **golem-afl.py** (my code) on ten nodes, with fuzzgoat app (in directory prj/fuzzgoat)), ten times 15 minutes each
- First cycles run on a single node only (deterministic run). When all deterministic paths were explored, the software switches to stochastic mode on 10 nodes.
- Each time the afl-fuzz is executed for 15 minutes, then the queue is distributed from all nodes to all other nodes and the job is restarted

## 5. outputs

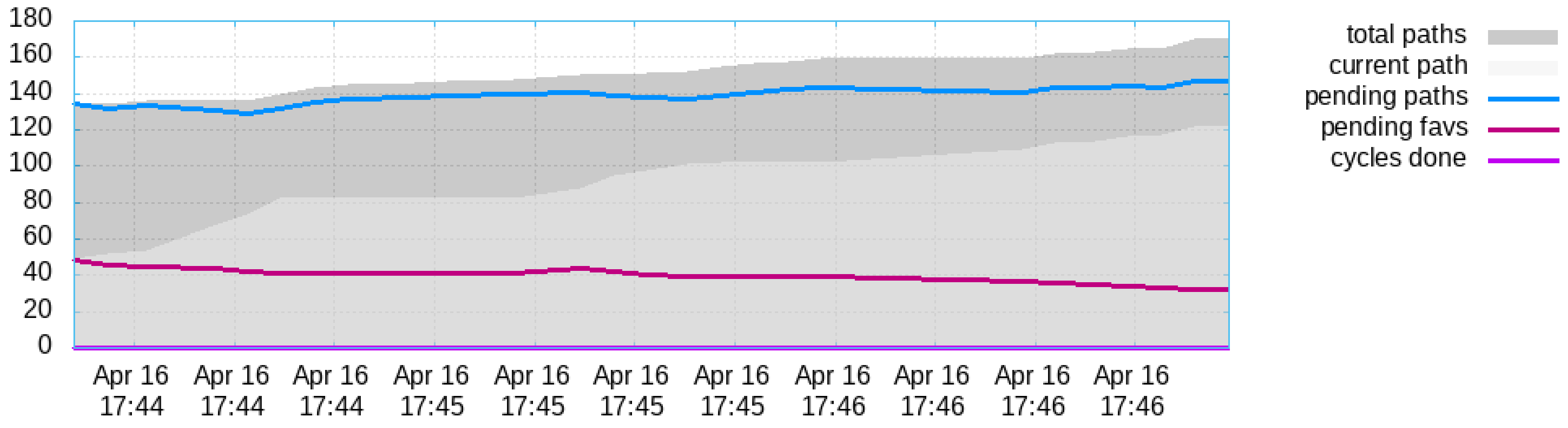
- Directory _out_results contains subdirectory for each cycle eg.  _out_results/cycle2
- Each cycle contains
  - Plots
  - Stats for each fuzzer (each node)
  - All crashes for each fuzzer

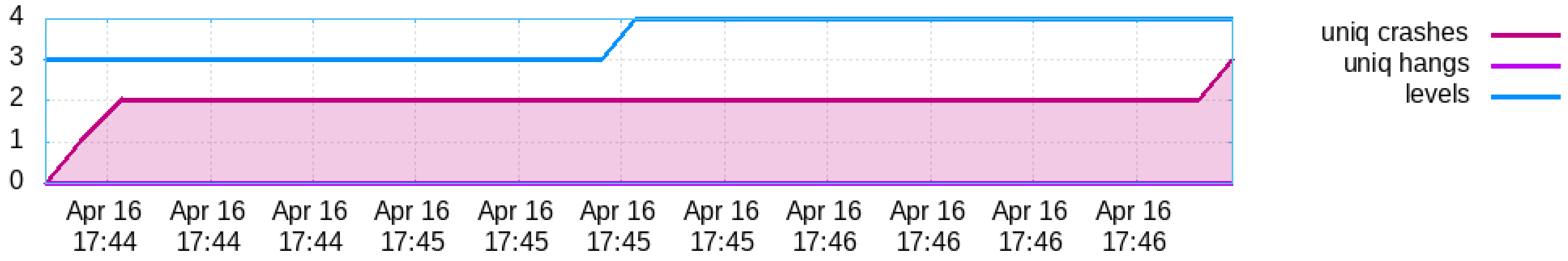
## Crash example

```
sla@dell9360:~/prj/golem-afl$ ls _out_results/cycle2/out/fuzzer0/crashes -1
```

- id:000000,sig:11,src:000122,op:havoc,rep:2
- id:000001,sig:06,src:000122,op:havoc,rep:2
- id:000002,sig:06,src:000144,op:havoc,rep:128
- id:000003,sig:06,src:000164,op:havoc,rep:32

# Plot example





all times in UTC

## stats example

```
execs_done        : 102774
execs_per_sec     : 570.15
paths_total       : 196
paths_favored     : 59
paths_found       : 26
paths_imported    : 0
max_depth         : 6
cur_path          : 178
pending_favs      : 42
pending_total     : 177
variable_paths    : 0
stability         : 100.00%
bitmap_cvg        : 0.64%
unique_crashes    : 4
unique_hangs      : 0
last_path         : 1618595397
last_crash        : 1618595359
last_hang         : 0
execs_since_crash : 30347
peak_rss_mb       : 1
```

## Conclusion

- **It works (for carefully selected demo projects)**
- For real projects it still has a lot of problems and requires plenty of  work
- Golem QEMM emulator makes execution slow.
- Some combinations of app/fuzzer crash without clear debugging output.