Intro to Fuzzing

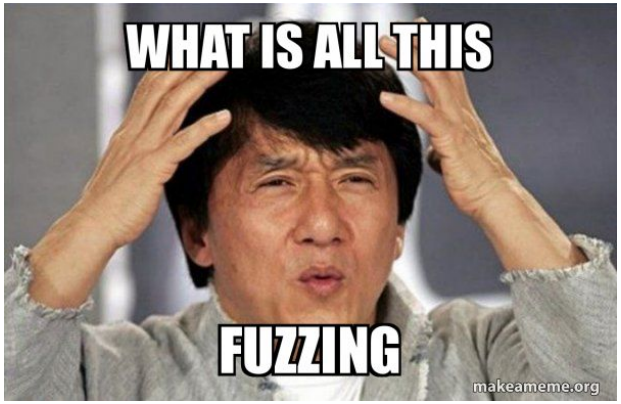
How to bug hunting
for fun and profit

17/02/2022

1 What and Why Fuzzing
2 The Zen of Fuzzing
3 Fuzzing Kata
4 Embedded Fuzzing

What and Why Fuzzing

#define Fuzzing



“Fuzzing is an automated software testing technique that attempts to find hackable software bugs by randomly feeding invalid and unexpected inputs and data into a program in order to find coding errors and security loopholes.”

#why Fuzzing ?

Fuzzing has a good track record of being efficient at bug finding

Compared to standard functional testing that focus on the "happy path", fuzzing tries to cover the edge cases

Recommended by PCI as the de facto dynamic testing method

The bug-o-rama trophy case

Yeah, it finds bugs. I am focusing chiefly on development and have not been running the fuzzer at a scale, but here a uniquely interesting bugs that are attributable to AFL (in large part thanks to the work done by other users):

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] ...
BIND [1] [2] [3] ...	QEMU [1] [2]	lcms [1]
Oracle BerkeleyDB [1] [2]	Android / libstagefright [1] [2]	iOS / ImageIO [1]
FLAC audio library [1] [2]	libsndfile [1] [2] [3] [4]	less / lesspipe [1] [2] [3]
strings (+ related tools) [1] [2] [3] [4] [5] [6] [7]	file [1] [2] [3] [4]	dpkg [1] [2]
rcs [1]	systemd-resolved [1] [2]	libyaml [1]
Info-Zip unzip [1] [2]	libtasn1 [1] [2] ...	OpenBSD pfctl [1]

AFL homepage

The Zen of Fuzzing

Fuzzing vs the world

Technique	Easy to start ?	Accuracy	Scalability
static analysis	easy	low	relatively good
dynamic analysis	hard	high	uncertain
symbolic execution	hard	high	bad
fuzzing	easy	high	good

Types of fuzzers

	Easy to start ?	Priori knowledge	Coverage	Ability to pass validation
Generation based	hard	needed, hard to acquire	high	strong
Mutation based	easy	not needed	low, affected by initial inputs	weak

What can a fuzzer find?

- Crashes, crashes, crashes !!!

- Bugs
  - Stack / Heap overflows
  - Null pointer dereferences
  - Off by ones
  - Uninitialized variables

The revolution: Coverage guided fuzzing

Leverage a feedback loop to guide input mutation

Coverage guided fuzzing principle

The Old Way: Stress test

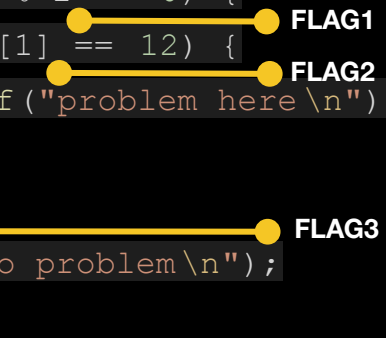
```
while (1)
    input[0] = rand() % 20;
    input[1] = rand() % 20;

    if (input[0] % 2 == 0) {
       if (input[1] == 12) {
          printf("problem here\n");
    }}
    else {
       printf("no problem\n");
    }
```

The New Way: Feedback guided

```
while (1)
    input[0] = rand() % 20;
    input[1] = rand() % 20;

    if (input[0] % 2 == 0) {
       if (input[1] == 12) {
          printf("problem here\n");
    }}
    else {
       printf("no problem\n");
    }
```



Fuzzing process with AFL, the coverage guided ⭐#1 fuzzer⭐

C Sources

Compile with GCC

Binary

Run with input

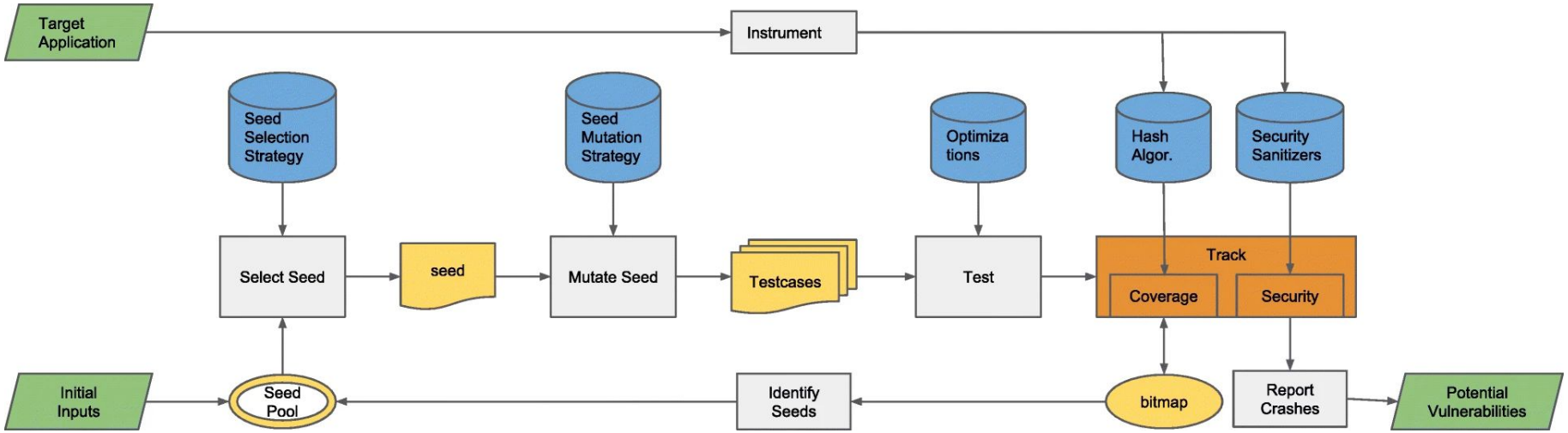
C Sources

Compile and *Instrument* with ***AFL*-GCC**

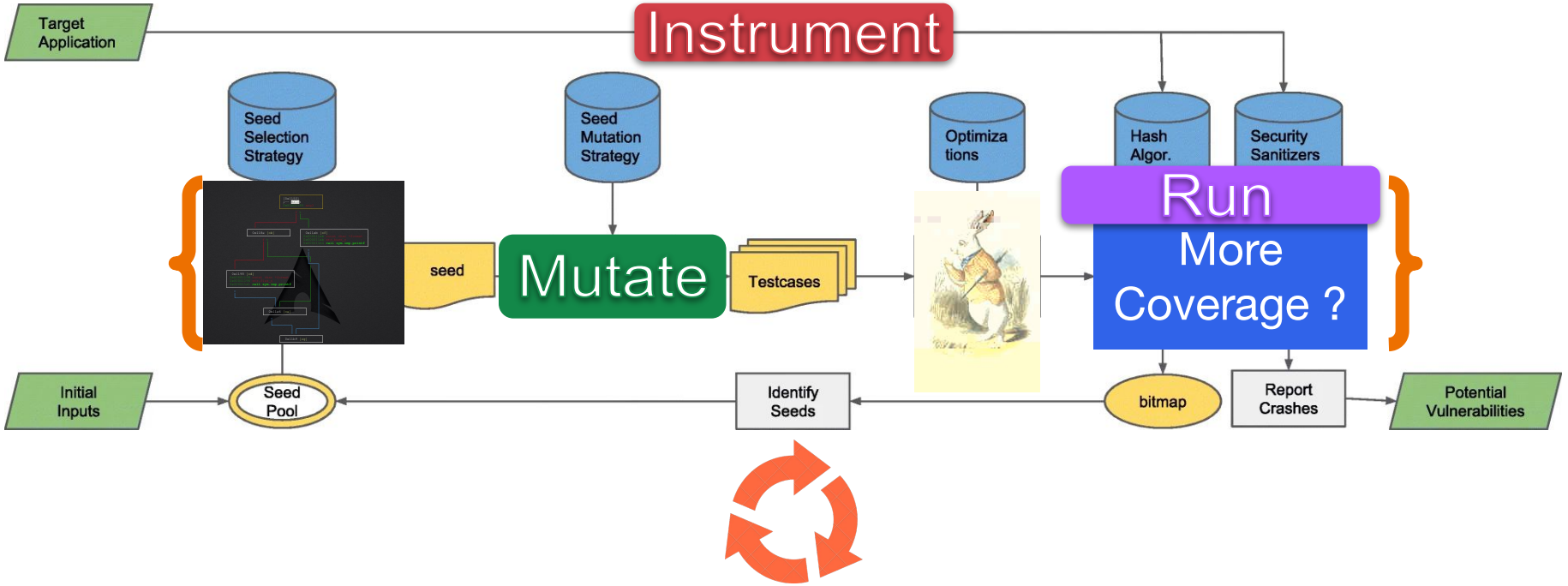
Instrumented Binary for *Coverage*

AFL-FUZZ Runs with *Mutated* input guided by *Coverage*
& Loop-Mutate until Maximize Coverage

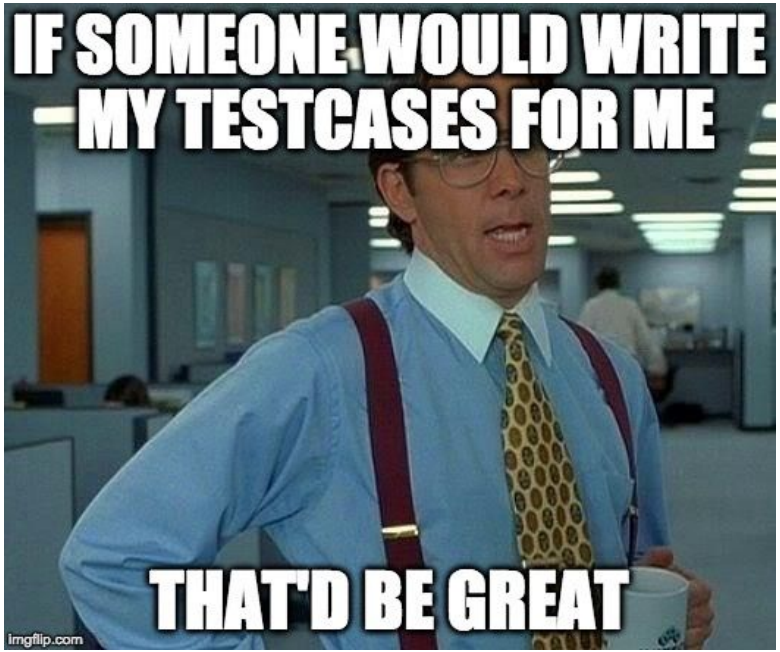
Fuzzer architecture



Fuzzer architecture explained



I know what you all think



Ahem … https://github.com/UnitTestBot/UTBotCpp

Fuzzing exercise

Fuzzing exercise

Click here
Open VM
user/pass: fuzz/fuzz
Clone the repository in the VM
And get fuzzing !

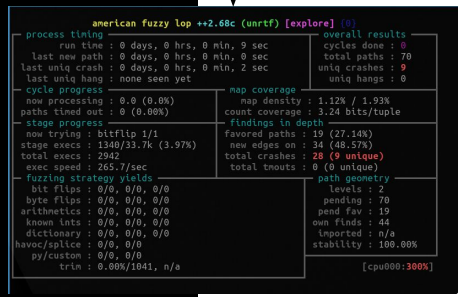
First steps

```
# download VM (5.1 GB)   https://drive.google.com/file/d/1UydxinlwKD847JHdbenO5gv7Xy7p3vJO/view
unzip Virtual\ Machine.zip

# install virtualbox on your OS
sudo apt-get install virtualbox
# or get it from here https://www.virtualbox.org/wiki/Downloads
# mac m1 users follow (next slide)
# run Virtualbox and file / import the image
# start the image and (recommended)
# take a snapshot after booting

# login with user/passwd : fuzz/fuzz and inside the image
cd ~/Desktop/WORKSHOP/Fuzz\ 0/unrtf
echo core | sudo tee /proc/sys/kernel/core_pattern
afl-fuzz -i ./tests -o afl-output  -- ./bin/unrtf --verbose -P ./lib/unrtf/ @@
```

You should get something like this



Mac M1

 # install afl natively on mac m1

via rosetta 2

follow the two links below to install rosetta and afl x86 version on arm

https://medium.com/mkdir-awesome/how-to-install-x86-64-homebrew-packages-on-apple-m1-macbook-54ba295230f

https://vineethbharadwaj.medium.com/how-to-compile-and-run-afl-fuzzer-on-m1-mac-with-apple-silicon-for-x86-instrumentation-support-4f1700eaafb6

 # use rosetta to change arch

```
alias arm="env /usr/bin/arch -arm64 /bin/zsh --login"
alias intel="env /usr/bin/arch -x86_64 /bin/zsh --login"
```

 # for intel x86_64 brew

```
alias intelbrew='arch -x86_64 /usr/local/homebrew/bin/brew'
```

* intelbrew install afl-fuzz
* git clone https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop
* follow workshop slides

 # alternative method ... only as a last resort if previous solution did not work

* download and install UTM from https://mac.getutm.app/
* follow https://mac.getutm.app/gallery/ubuntu-20-04
* sudo apt install afl++
* git clone https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop
* follow workshop slides replacing afl-gcc with afl-clang-fast

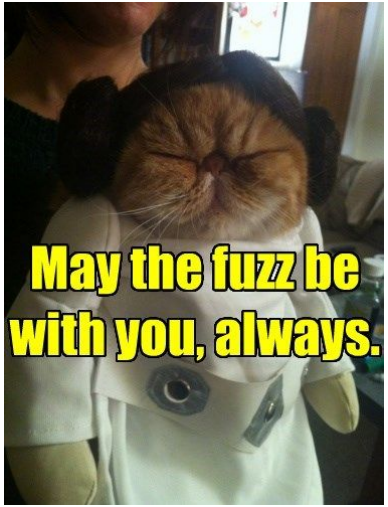
First steps

```
cd ~/Desktop/WORKSHOP/Fuzz\ 0/unrtf
echo core | sudo tee /proc/sys/kernel/core_pattern
afl-fuzz -i ./tests -o afl-output  -- ./bin/unrtf --verbose -P ./lib/unrtf/ @@

cd ~/Desktop/WORKSHOP/
git clone https://github.com/antonio-morales/Hackfest_Advanced_Fuzzing_Workshop
cd Fuzz1

# GOAL: Find the bug with AFL fuzzer
# Follow the hints
```

Follow the white rabbit



Embedded Fuzzing

```
# Run afl workshop inside docker emulating raspberry pi

docker run --network host -it -v $HOME/.dockerpi:/sdcard lukechilds/dockerpi
// or use podman

# user / pass : pi / raspberry

# install afl / clone workshop and fuzz like previously

# GOAL: Find the bug with AFL fuzzer
# Follow the hints
```


FUZZ
ALL THE THINGS