SAPIENZA
UNIVERSITÀ DI ROMA

Security in Software Applications Proj 2

Alex Parri - 2140961 - Master's Degree in Cybersecurity

A.Y. 2024/25

Abstract

This is the report for the **second project** of the Security in Software Applications course directed by Daniele Friolo for the Academic Year 24/25 for the Master's Degree in **Cybersecurity** at Sapienza University of Rome. In this homework, the goal was to experiment with **fuzz testing**, a form of software dynamic analysis.

Specifically, it was asked to use the AFL tool to **test** the image manipulation software ImageMagick, and summarize the obtained results. It was decided to use one of its forks AFL++ as it a **superior, modern and maintained** fork of the former.

The **hardware** utilized for testing is Ryzen 5800X 8-Core 16-Thread @ 4.850GHz with clang v18.1.3, AFL++ v4.32c in Ubuntu 24.04.02 LTS x86_64 and 16GB of RAM.

AFL++

AFL++ (American Fuzzy Lop ++) is a modern, improved fork of the original AFL (American Fuzzy Lop) binary. It is a **powerful fuzz tester** for finding bugs and vulnerabilities automatically, which involves the input in a target program of **carefully mutated** inputs to more likely trigger unexpected behavior such as crashes and hangs.

Setting up the tool

Setting up AFL++ can be **quite overwhelming** for the faint of heart. This is also because it allows a significant degree of **customizeability** and alternatives from instrumenting, to preparing and finally managing the fuzzing campaigns.

Instrumentation

First things first we **instrument** the target program, this is done to prepare it to be fuzzed efficiently and consequently improve efficacy in yielding unexpected behavior. This is done by compiling it with a **special compiler** (`afl-cc`) and in this specific case with **LTO mode** (`afl-clang-lto`)

```
cd ImageMagick-6.7.7-10

# set up AFL++ environment
export CC=afl-clang-lto
export CXX=afl-clang-lto++
```

```
# sanitizer (optional)
export AFL_USE_$sanitizer=1

# build from source
./configure --disable-shared --enable-static \
    --without-magick-plus-plus --without-perl \
    --without-x --without-opencl
make -j$(nproc)
```

The tool also allows to use **sanitizers** (`AFL_USE_$sanitizer=1`) to find bugs that not necessarily result in a crash. If selected, only **one instance** at the time of the fuzzer (`afl-fuzz`) is enough, using multiple concurrent instances would be a waste of computing power.

`$sanitizer`	**Purpose**
ASAN	memory corruption vulnerabilities
MSAN	read accesses to uninitialized memory
UBSAN	actions with undefined behaviour
CFISAN	instances where the control flow is illegal
TSAN	thread race conditions
LSAN	memory leaks

Alternatively it is also possible to do **black box fuzzing** but it is a much better option to exploit the fact that the source code is **publicly available** for everyone to download and play around with.

Setting up the corpus

In order to begin its operation, the fuzzer requires a **corpus**: few small input files the application considers as valid. It was decided to randomly generate noisy ones using **ImageMagick itself** to try to maximise the chance for mutations

```
mkdir afl-tests/png-in afl-tests/png-out
code generate_corpus.sh
for i in {1..5}; do
  SIZE=$((RANDOM % 40 + 10))    # between 10x10 and 50x50
  ImageMagick-6.7.7-10/utilities/convert -size ${SIZE}x${SIZE}
  ↪  plasma:fractal afl-tests/png-in/noise_$i.png
done
```

Running `afl-fuzz`

Once all of the above is complete we run the fuzzer and let the **magic happen**

```
afl-fuzz -i afl-tests/png-in \
    -o afl-tests/png-out \
    -- $command
```

If **no sanitizer** is being used then it is possible to **concurrently run** multiple instances for better efficiency. One of the possibilities is launching a **master fuzzer** (`-M`) on the first CPU core which coordinates all the work with its slave fuzzers (`-S`)

```
# running master fuzzer on first CPU core
taskset -c 0 afl-fuzz -M masterfuzzer \
    -i afl-tests/png-in \
    -o afl-tests/png-out \
    -- $command
```

The chosen number of slaves is five, each on its own **terminal** handled by its own **CPU core** (`-c $i`)

```
# running $i-th slave fuzzer on CPU core $i
taskset -c $i afl-fuzz -S slavefuzzer$i \
    -i afl-tests/png-in \
    -o afl-tests/png-out \
    -- $command
```

Since fuzzing very usually takes a while to **bear fruit**, AFL++ also supports resuming of a previous **ongoing session** with the `-i-` command-line option while keeping the same output and command options

```
afl-fuzz -i- \
    -o afl-tests/png-out \
    -- $command
```

The option in question is also valid for the **master-slave configuration**

```
taskset -c 0 afl-fuzz -i- -M masterfuzzer \
      -o afl-tests/png-out \
      -- $command

taskset -c $i afl-fuzz -i- -S slavefuzzer$i \
      -o afl-tests/png-out \
      -- $command
```

The variable `$command` depends on the **specific command** it was decided to fuzz.

Fuzzing results

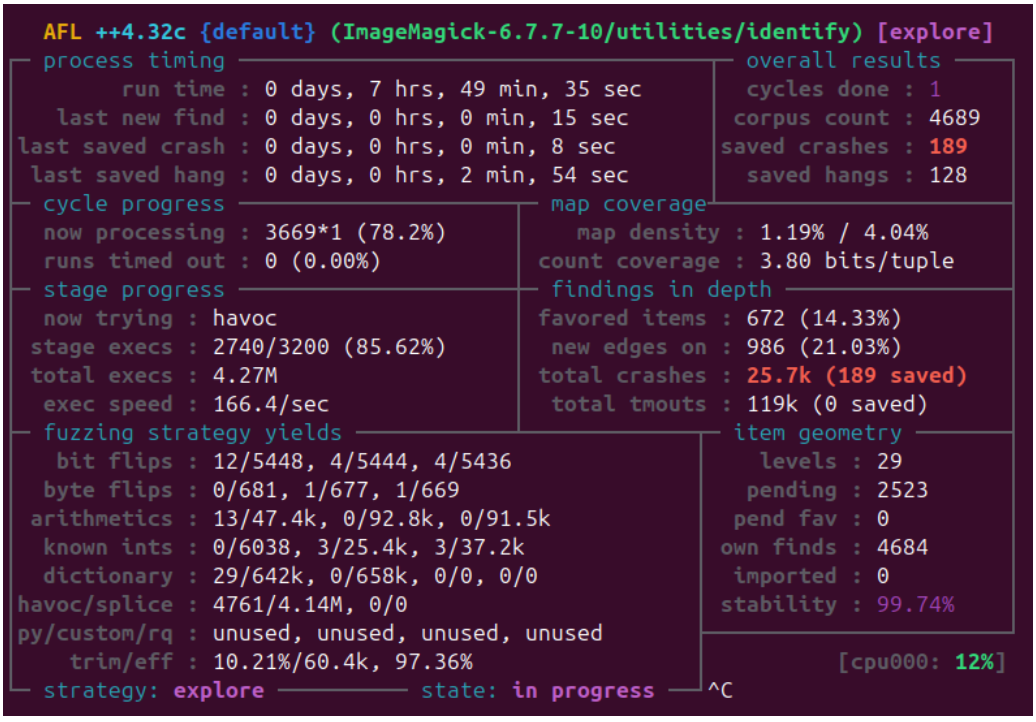
The purpose of this homework was to play around with ImageMagick. It was decided to use the v6.7.7-10 to **more easily** find and analyze crashes and hangs, as testing for a more modern version would imply spending **significantly more time** fuzzing to obtain a comparable number of results.

The arbitrarily **chosen** command to mess around with is `identify`, whose **purpose** is reported from the official documentation as: *"The `identify` command describes the format and characteristics of one or more image files."*

The `$command` variable that was previously mentioned would be assigned as follows

```
command = ImageMagick-6.7.7-10/utilities/identify @@
```

In this context it was decided to specify the **address sanitizer** (`AFL_USE_ASAN=1`) and thus running a single master fuzzer, whose result of is shown in the following figure



The fuzzer was ran for **almost 8 hours** (7h49m), having attempted over 4.27 million overall **mutations** with a developed corpus of 4689 (up from the starting 5) and an overall number of detected flaws amounting to 256. These are very promising numbers, signifying that the fuzzer was **set up properly** and that it beared its fruits.

The raw fuzzing part is **complete**, now it is time to analyze the crashes.

Triaging the crashes

The tool reported a number of crashes that is **too great** for any patience-armed individual to be analyzed one-by-one, therefore it was decided to **deduplicate** them based off of the hash of the top of the stack trace (#0) obtained by inputting each crash file to the `identify` command.

```
#!/bin/bash

# create directory if not there
mkdir -p "$UNIQUE_DIR"
declare -A seen_traces

# save current ASLR setting
ASLR_SETTING=$(cat /proc/sys/kernel/randomize_va_space)

# temporarily disable ASLR
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space > /dev/null
echo "Temporarily disabled ASLR (was $ASLR_SETTING)"

# loop through all the crashes
for crash in afl-tests/png-out/default/crashes/id:*; do
    [[ -f "$crash" ]] || continue

    # execute the line and filter the top line
    trace_line=$(./ImageMagick-6.7.7-10/utilities/identify "$crash" 2>&1 |
    ↪  grep -m1 '^[[:space:]]*#0')

    # hash it for fast comparison
    hash=$(echo "$trace_line" | md5sum | awk '{print $1}')

    # check if already exists
    if [[ -z "${seen_traces[$hash]}" ]]; then
        echo "Unique crash: $crash ($trace_line)"
        cp "$crash" unique_crashes/
        seen_traces["$hash"]=1
    fi
done

# restore ASLR settings
echo "$ASLR_SETTING" | sudo tee /proc/sys/kernel/randomize_va_space >
↪  /dev/null
echo "Restored ASLR to $ASLR_SETTING"
```

The above script performs what exactly has been explained: the result is a much **more contained** 19 unique crash files. It is worth noting that ASLR had to be disabled for it to work, otherwise every crash file would have a **unique** stack trace.