4. Exercises

ASI36

2021

1 Introduction

This exercise sheet uses AFL to fuzz small C programs. Here are some external resources to help you use the tool

- **Tutorials on AFL**
  - `https://fuzzing-project.org/tutorial3.html`
  - `https://labs.nettitude.com/blog/fuzzing-with-american-fuzzy-lop-afl/`
  - fuzzing-with-afl-fuzz-a-practical-example-afl-vs-binutils
  - `https://research.aurainfosec.io/hunting-for-bugs-101/`
- **Important command-line options**
  - `AFL_SKIP_CPUFREQ=1`
  - `AFL_USE_ASAN=1`

Since AFL depends very much on randomness, it is important to run the experiments multiple times to draw conclusions. If you find something once, you might just have been lucky; if you find it 90% of the time on a consequent number of runs, it's another matter.

2 Magic bytes (`magic.c`)

Let's consider the following program.

```c
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

void crash()
{
     raise (SIGSEGV);
}

#define BUFSIZE 1024

int main(int argc, char* argv[])
{
     char inp[BUFSIZE] = { 0 };
     if (argc > 1)
     {

        int f = open(argv[1], O_RDONLY);
        read(f, inp, BUFSIZE);
        int in = atoi(inp);
        if (in == 0xdeadbeef) {
            printf("Aaargh!\n");
            crash();
        }
        printf("You lose\n");
        return 0;
    }
    printf("Please, at least one arg !\n");
    return 0;
}
```

1. Fuzz this program for 5 minutes *with an empty seed.* Did you find a crash?
2. Fuzzers include a fair bit of randomization, maybe you just were not lucky. Now rerun this for 5 more minutes (Try it twice more).

   Did you find a crash this time?
3. Try out fuzzing with non-empty seeds.
   - Try with the expected solution – it should find the crash right away.
   - Give smaller and smaller prefixes to the solution. When does the fuzzer not reach the target anymore ?
4. Rewrite the program so that it is semantically equivalent to the original program (no loss of functionality) but so that the fuzzer can reach the buggy path *with an empty seed.*

3 CROMU_00007

The `CROMU_00007` directory contains a challenge that was part of the Cyber-Grand-Challenge.

The small program implemented is described in the `README.md` file ot this directory.

There are two vulnerabilities in this program. One is easy to find, the other harder. The `README.md` file also describes the two vulnerabilities.

1. Find the easy vulnerability You can use the provided `sample.input` or craft other ones.

   (a) Fuzz until AFL tells you it has found at least 20 unique crashing inputs.

   (b) Have a look at the crashing inputs. Do they actually exercises different paths?

   (c) What is the smallest input found by AFL? Can you try to find a smaller one?

   (d) Fix the vulnerability you found in 1.
2. *(Bonus)* Can you find a crashing input by fuzzing for the second bug?

4 Hard-to-find events (uafuzz.c)

```c
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>
# include <fcntl.h>

int f, *p, *p_alias;
char inp[10], *buf[5];

#define KO (-1)
#define OK 0

void bad_func(int *p) {
    free(p);
}

int benign_func(int *p) {
    if (inp[2] == 'F' && inp[3] == 'o' && inp[4] == 'o') {
        free(p);
        return KO;
    }
    return OK;
}

void func() {
    if (inp[1] == 'A') {
        bad_func(p);
        if (inp[2] == 'F' && inp[3] == 'u' && inp[4] == 'z') {
            *p = 1;
        } else {
            p = malloc(sizeof(int));
            p_alias = p;
            if (benign_func(p_alias) == -1) return;
            *p_alias = 1;
            free(p);
        }
    }
}

int main (int argc, char *argv[]) {
    f = open(argv[1], O_RDONLY);
    read(f, inp, 10);

    if (inp[0] == 'U') {
        p = malloc(sizeof(int));
        p_alias = p; // p_alias points to the same area as p
        func();
    }
    return OK;
}
```

1. Find and explain the vulnerability contained in this program.
2. Run the fuzzer multiple times (5 minutes) on the above program. Did you find any crash ? If not, can you guess why ?
3. Recompile your program with AddressSanitizer activated, and fuzz it again, multiple times. Do you find crashing inputs ?