Fuzzing for Software Security Testing AKOR JACOB TERUNGWA

Created	@February 21, 2025 8:22 PM
Attendance Required	☐

1. Install AFL:

AFL is a popular fuzzing tool that uses genetic algorithms to generate test cases. Here's how i installed AFL (American Fuzzy Lop)

First, i updated the package list:

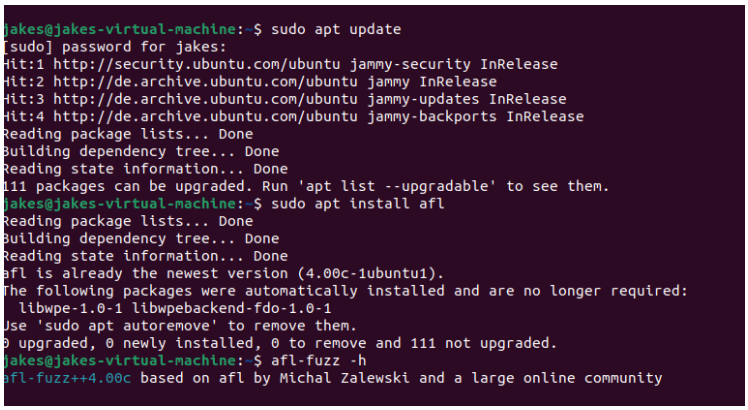
- sudo apt update

I then **Installed AFL by using the command:**

- sudo apt install afl

I also verified the installation:

- afl-fuzz -h



2. Create a Target Application: Save the following C code as vulnerable.c:

- I created a file and saved the code in a file named `vulnerable.c`

3. Compile the Target Application with AFL

For better performance and more efficient fuzzing, i considered using:

- `afl-gcc -o vulnerable vulnerable.c`

Step 2: Run the Fuzzer

I created an input directory and added a seed file and ran AFL

- afl-fuzz -i input -o output ./vulnerable @@

Step 4: Analyze the Results

1. **I checked the Output Directory**:

```
jakes@jakes-virtual-machine:~$ cd output
jakes@jakes-virtual-machine:~/output$ ls
default
jakes@jakes-virtual-machine:~/output$ cd default
jakes@jakes-virtual-machine:~/output/default$ ls
cmdline  crashes  fuzz_bitmap  fuzzer_setup  fuzzer_stats  hangs  plot_data  queue
jakes@jakes-virtual-machine:~/output/default$ cd crashes
jakes@jakes-virtual-machine:~/output/default/crashes$ ls
id:000000,sig:06,src:000000+000003,time:796,execs:325,op:splice,rep:8  README.txt
jakes@jakes-virtual-machine:~/output/default/crashes$
```

2. **I replayed the Crash using a command:**
    - ./vulnerable output/default/crashes/id:000000,sig:06,src:000000+000003,time:796,execs:325,op:splice,rep:8

```
jakes@jakes-virtual-machine: $ ./vulnerable output/default/crashes/id:000000,sig:06,src:000000+000003,time:796,execs:325,op:splice,rep:8
*** buffer overflow detected ***: terminated
Aborted
jakes@jakes-virtual-machine: $
```

The crash occurred due to a **buffer overflow** vulnerability.

Step 5: Fix the Vulnerability

To fix the vulnerability, i modified the code to replace `strcpy` with `strncpy` to prevent buffer overflow:

- `strncpy(buffer, input, sizeof(buffer) - 1);`
- `buffer[sizeof(buffer) - 1] = '\0';`

jakes@jakes-virtual-machine: ~

GNU nano 6.2 vulnerable.c

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

void vulnerable_function(char *input) {
    char buffer[100];
    strncpy(buffer, input, sizeof(buffer) - 1);
    buffer[sizeof(buffer) - 1] = '\0'; // Ensure null termination
}

int main(int argc, char *argv[]) {
    if (argc > 1) {
        FILE *file = fopen(argv[1], "r");
        if (!file) {
            perror("Error opening file");
            return 1;
        }
        char input[1024];
        size_t bytesRead = fread(input, 1, sizeof(input) - 1, file);
        fclose(file);
        input[bytesRead] = '\0';
        vulnerable_function(input);
    } else {
        printf("Usage: %s <filename>\n", argv[0]);
    }
    return 0;
}
```

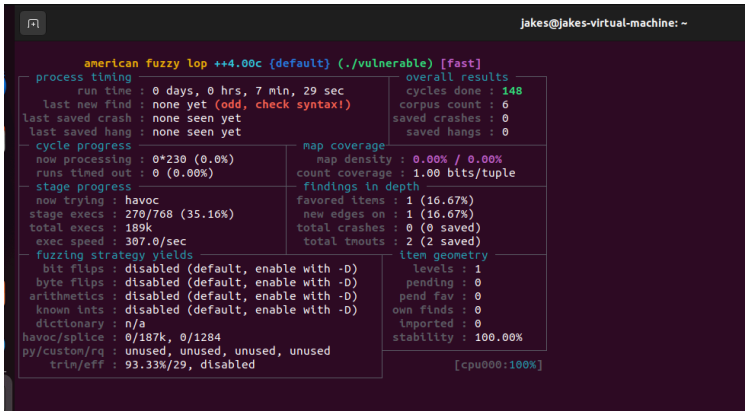
I run the command below to removed the vulnerable files:

- rm -rf vulnerable

Then i recompiled and rerun the Fuzzer

- afl-fuzz -i input -o output ./vulnerable @@

As seen below in the output, there are no crashes which indicates that the vulnerability has been fixed



QUESTIONS:

1. **What is the purpose of fuzzing in software security testing?**

- Fuzzing is used to discover vulnerabilities, crashes, and unexpected behavior in software by feeding it random or semi-random inputs.

2. **How does AFL generate test cases to find vulnerabilities?**
    - AFL uses genetic algorithms to mutate input seeds and generate new test cases. It prioritizes inputs that trigger new code paths or crashes.
3. **What other types of vulnerabilities can fuzzing detect besides buffer overflows?**
    - Fuzzing can detect:
        - Memory leaks.
        - Use-after-free vulnerabilities.
        - Integer overflows.
        - Format string vulnerabilities.
        - Logic errors.
4. **How can you improve the efficiency of a fuzzing campaign?**
    - Use high-quality seed inputs.
    - Combine fuzzing with static analysis.
    - Use parallel fuzzing.
    - Implement coverage-guided fuzzing.