CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Security Testing

WS 2021/2022

Prof. Dr. Andreas Zeller
Leon Bettscheider
Marius Smytzek

Project 1

Due: 02. January 2022

> The lecture is based on The Fuzzing Book (https://fuzzingbook.org/), an *interactive textbook that allows you to try out code right in your web browser*.
> The Fuzzing Book code is additionally available as a Python pip package. To work on the exercises, please install the package locally:
> `pip3 install fuzzingbook`

Submit your solutions as a Zip file on your status page in the CMS (https://cms.cispa.saarland/fuzzing2122/students/view).

We will provide you a structure to submit your solutions where each task has a dedicated file. You can add new files and scripts if you want, but you may not delete any provided ones. You can verify whether your submission is valid by executing `verify.py`:

```
python3 verify.py
```

The output provides an overview if a required file, variable, or function is missing and if a function pattern was altered. If you do not follow this structure or change it, we cannot evaluate your submission. A non evaluable project will result in 0 points, so make sure to verify your work before submitting it. Note that the script does not reveal if your solutions are correct.

Project Description

For this project, you will build a **grammar-based blackbox fuzzer** to test the Internet Relay Chat (IRC) protocol. Although the text-based IRC protocol dates back to the year 1988, it is still widely used today. The fuzzer you will build acts as an **IRC client** that connects to the **IRC server**, which we want to test. Both programs will be running locally on your machine.

The **IRC server** under test, miniircd (https://github.com/jrosdahl/miniircd), is implemented in Python.

If you're not familiar with IRC, it might be helpful to download and try out an IRC client, such as HexChat (https://hexchat.github.io/). Before getting started with the tasks below, please read this introduction to IRC (http://chi.cs.uchicago.edu/chirc/irc.html) and these IRC Communication Examples (http://chi.cs.uchicago.edu/chirc/irc_examples.html). These links are part of an assignment provided by the University of Chicago, which is not related to fuzzing. Please ignore any details that are specific to their assignment. However, most of their descriptions can be applied to our fuzzing project just as well.

Server Framework

server.py contains the server framework. It runs the server for 60 second and connects 10 clients to it that join the channel `#main` and then idle. These dummy clients enable your fuzzer to perform IRC requests that require other connected clients (and thus increase code coverage).

You must not change neither this file nor any code in miniircd.

Fuzzing Framework

We provide you with an IRC fuzzing framework in **fuzzer.py** that you can leverage as a starting point for your implementation.

The framework first establishes a connection to the IRC server and logs into the server by sending a `NICK` and a `USER` request. Once the fuzzer has logged in, it can interact with the server, i.e., fuzz it. IRC servers periodically send `PING` requests which our framework already handles by responding with a `PONG`.

You are allowed to change the code of the fuzzing framework in any way you want or even rewrite it from scratch. However, you are not allowed to change the server host and port, which must be **localhost:6667**. Also, the fuzzer must be runnable using the command `python3 fuzzer.py`, and it should only use fuzzers provided by The Fuzzing Book or implemented by yourself.

How to fuzz

- Step 1: Start the server (which runs for 60 seconds).
- Step 2: Start the fuzzer.
- Step 3: After `60` seconds, the server will shut down and produce a coverage file named `.coverage` in the current directory. Based on this file, you can generate a coverage report using the command `coverage report`. To run `coverage`, install it via `pip` by running `pip3 install coverage`. If your system does not link `coverage` as an executable command, you can run it with `python3 -m coverage report`.

```
$ coverage report
Name                Stmts   Miss  Cover
-------------------------------------
miniircd/miniircd     758    362    52%
-------------------------------------
TOTAL                 758    362    52%
```

In addition, we provide you with the `run.py` script that automates this process if you want. The script starts the server, waits 2 seconds to allow the server to establish a connection, and then starts the `fuzzer.py`. The stdout of the server and the fuzzer are flushed to `server.log` and `fuzzer.log`, respectively.

Windows

We additionally provide support for Windows. To enable the fuzzing, please install `console-ctrl` by running `pip3 install console-ctrl`. If you have no Windows machine, you do not need to install this package. Now you can follow the steps above to run your fuzzer. Do not be alarmed because a new console pops up. This behavior is normal and wanted. Do not try to close this console because it runs the server. As soon as the time runs out, the console will close automatically.

Requests

For a documentation of the IRC protocol, you can refer to the rfc 2812 (https://datatracker.ietf.org/doc/html/rfc2812). The syntax of the requests your fuzzer uses should be encoded as a *context-free grammar* in The Fuzzing Book format. Additionally, you are allowed to add arbitrary code to your fuzzer. Moreover, you can apply all techniques we discussed in the lecture.

It can be helpful to parse the responses received from the server to learn about stateful information from which your fuzzer can then benefit. For instance, you can only send a private message to other connected clients if you know their names.

Notes

- Converting an extensive IRC trace to a context-free grammar and using this as a *fuzzer* is not a valid solution. Your fuzzer should be able to come up with a large set of different combinations of IRC requests.

Evaluation Guidelines

You should work individually on this project. Group work is not permitted.

Your fuzzer will be evaluated across three dimensions and needs at least **50%** over all dimensions (not for each) to pass the project:

1. Code coverage in the server (33.3%)

We will measure how much code your fuzzer can cover in the server. To be able to cover much code, your fuzzer needs to support a large chunk of the IRC protocol. During the evaluation, we will start an instance of the server, and your fuzzer will be allotted a time budget of **60 seconds** to test the server. We will repeat this measurement five times. The fuzzer **must run in one thread** only.

2. Bug finding capability (33.3%)

If your fuzzer is able to trigger an exception or a crash in the server, you will get extra points. Since finding a bug in miniircd might be difficult, we will also seed several bugs in the server and evaluate the ability of your fuzzer to discover them.

3. Generality (33.3%)

To reduce the incentive to overspecialize your fuzzer to the specific IRC server under test, we will additionally test your fuzzer on a different IRC server implementation.

Guaranteed passing criterium

If you not pass by our evaluation guidelines, you are still guaranteed to pass the project if your fuzzer achieves at least **70%** code coverage (measured in statement coverage by `coverage.py`) in miniircd.

Note that code coverage is only one of the metrics we evaluate your fuzzer on.