Ira A. Fulton Schools of Engineering
Arizona State University

CSE 543: Information Assurance and Security

Fuzz Them All Project

Purpose

The purpose of this project is to test your understanding of fuzz testing (fuzzing) and guide you to develop your own binary fuzzer. You will learn how to develop your own simple fuzzer, monitor for crashes, generate high-quality seed input, etc.

Objectives

Learners will be able to:

- Determine how fuzzers for binary programs work.
- Differentiate between types of fuzzers
- Categorize advantages of each type of fuzzer (e.g., smart fuzzers vs. dumb fuzzers and mutational vs. generative fuzzers).
- Understand the importance of seed input selection.
- Develop a simple fuzzer to fuzz binary programs when source code is available.
- Develop a simple fuzzer to fuzz binary programs when source code is not available.

Technology Requirements

You may use any programming language to implement the fuzzer. However, for a better speed, native languages, such as C, C++, Go, and Rust are strongly recommended. You may also use Java or C# with some acceptable level of speed penalty compared to native languages. You are not encouraged to use Python, but you can if you wish.

Your fuzzer will work against Linux programs that accept input through stdin and generate output to stdout. Windows or MacOS programs are not valid targets. If you are using Windows or MacOS, you may want to install a virtual machine application. You can get VirtualBox for free (strongly recommended) and install Ubuntu 20.04 LTS in a VM.

Project Description

For this project, you are supposed to develop a dumb mutative fuzzer. You may want to develop a smart fuzzer. However, it is important to know that developing a good fuzzer takes a lot of experience and time. Developing a good smart fuzzer within two weeks is usually infeasible. If you are still interested, please talk to the instructor to receive this bonus challenge!

You may refer to “Directions” for a step-by-step guide of fuzzer development. Your fuzzer will take an initial seed and two arguments, `prng_seed` and `num_of_iterations` as input. Your fuzzer will generate output to stdout, and the output will be used as the input to target programs that will be fuzzed. Your fuzzer may write anything to stderr; any data going to stderr will be discarded.

Directions

Technology Setup Reminder

If you have not already joined the course’s pwn.college, please review the setup directions in *Module 0: Welcome and Start Here* of your course to properly gain access and start your work.

Accessing the Environment

For this project, you may use the pwn.college challenge environment to develop and test your solution. Your final work must be submitted in the course by following the directions under “Submission Directions for Project Deliverables”.

To access the pwn.college environment:

1. Navigate to https://pwn.college.
2. Click “**Login**” in the upper right corner of the screen and enter your account credentials.
    a. Click “Forgot your password?” if you have trouble logging in.
3. Navigate to “**Dojos**”, second option from the left at the top of the screen.
4. Under “**Courses**”, select “**CSE 543 - Session X Year**”.
5. Under “**Modules**”, select “**Fuzz Them All Project**”.

6. Under “**Challenges**”, click on each level (for example, “**Level 1**”), read the details, and then click “**Start**” when you are ready to work.

    a. The program that your fuzzer wants to crash is under `/challenge/prog`.

    b. While there are ten levels (one for each level), you should only develop one (1) fuzzer for all challenges.

    c. Your fuzzer may not be able to crash all 10 challenges. Don’t panic. It is totally normal and expected! Fuzzers are not perfect.

Project Directions

Your fuzzer will take an initial seed and two arguments, `prng_seed` and `num_of_iterations` as input. Your fuzzer will generate *one* output in a deterministic manner, i.e., for the same combination of initial seed, `prng_seed`, and `num_of_iterations`, your fuzzer should generate the same output file. This output file will be used as input to the fuzzing target.

- Your fuzzer will be executed using the following command line:

```
./fuzzer prng_seed num_of_iterations
```

where `prng_seed` is a 32-bit integer that you may use to seed your PRNG(s), and `num_iterations` is a 32-bit integer that determines how many iterations the fuzzer will run.

- Your fuzzer will read an initial seed file called _seed_. The _seed_ file is located under the current working directory.

- Optionally, your fuzzer will use `prng_seed` to seed any PRNG(s) that will be used during fuzzing.

- Your fuzzer will iterate for `num_iterations` times. In every iteration, your fuzzer should change each byte of the input to a random byte with 13% probability. Do not change or overwrite the `seed` file on the disk.

- Your fuzzer will extend the input string by adding 10 random characters to the end of the input every 500 iterations. Again, do not change or overwrite the `seed` file on the disk.

- After all iterations, your fuzzer will write the mutated input to stdout. This means that your fuzzer should not write to stdout anything that is not part of your mutation result. For debugging purposes, your fuzzer may write to stderr. Any data that is written to stderr will be ignored.

- Your fuzzer terminates.

Your fuzzer must behave in a deterministic manner. This means if you execute your fuzzer twice with the same set of parameters, you will get the same mutated output in both executions.

Your code should be understandable and well documented in case the instructor or the TA decides to manually grade your submission.

You will be given a set of three target programs and a seed input file for each of them. You may test your fuzzer against the target programs with different PRNG seeds and numbers of iterations. You may monitor the return value of target programs after running them with input that is generated by your fuzzer. When the target program crashes, write down your `prng_seed` and `num_iterations`.

Hints: Note that your fuzzer will have access to the executable itself. You may build a dictionary of strings in the executable to help your fuzzer generate input with higher quality.

Submission Directions for Project Deliverables

Prepare Your Deliverables

Your **fuzzer code** should include a `fuzzer` executable that takes command line parameters as previously described.

There will be **10 test programs provided in total**. For each test program, submit a text file named after each level (e.g., “level-1” for Level 1). The text file should contain two lines. The first line is the `prng_seed`, and the second line is `num_iterations`. Your fuzzer should generate a crashing input within half an hour if we feed the specified `prng_seed` and `num_iterations` to your fuzzer.

Moreover, for each test program that your fuzzer successfully crashed, you should submit the **_exact_ crashing input** that your fuzzer generated for crashing that test program. The input should be stored in a `.crash` file.

Here is an example: If the test program is called `test`, then your text file should be named `test.txt`. `test.txt` contains two lines. The first line is `abcdef` (the seed), and the second line is `500` (the number of iterations). The TA will run your fuzzer with `./fuzzer abcdef 500`, and determine if it generates the content in `test.crash` (the crashing input) within 30 minutes.

The grader will automatically run `test.crash` against the test program, and if the test program crashes with your given input, you will be given points for this test program.

The grader will also test if your submitted crashing input is the same as the one that your fuzzer generates. Because your fuzzer should work deterministically, we will contact you if your fuzzer does not generate the crashing input file that you submitted.

Note that the TA will examine your fuzzer code. You should not special-case or hard code in your fuzzer any logic that is specific to the seed content or iteration times. Violation of this rule will lead to a zero for this project.

A **document** listing any dependencies that your fuzzer has, and detailing your input generation strategy.

Submission Directions

You are given an unlimited number of attempts to submit your best work. The number of attempts is given to anticipate any submission errors you may have in regards to properly submitting your best work within the deadline (e.g., accidentally submitting the wrong paper). It is not meant for you to receive multiple rounds of feedback and then one (1) final submission. Only your most recent submission will be assessed.

You must complete your Fuzz Them All Project deliverables in **pwn.college** and then **submit the deliverables in its submission space in the course**. Carefully review submission directions outlined in the overview document in order to correctly earn credit for your work. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

The Fuzz Them All Project includes two (2) deliverables

- **Fuzzer Code**: A `fuzzer` executable that contains
    - Ten (10) test programs each submitted as a `.txt` file and titled accordingly
    - _exact_ crashing input for each test program as a `.crash` file
- **Report**: A document (DOC, DOCX, or PDF) listing any dependencies that your fuzzer has, and detailing your input generation strategy
    - Please include your name, course title, and date with your writing
    - Your file should be titled using the format: “Last Name_First Name_CSE543_Fuzz Them All Project”

Making File Submissions in Canvas

Before submitting, confirm that your deliverables follow the requirements for the project, and then submit your work in the designated submission space in the course. Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated into your grade.

1. In your course, go to **Submission: Fuzz Them All Project**.
2. Click **Start Assignment**.
3. Click **Choose File**.
4. Locate and select **one (1)** deliverable file from your device.
5. If needed, click **+Add Another File** and repeat Steps 3 and 4 until all deliverables are added.
6. Select the **agreement** and then click **Submit Assignment**.
7. (If needed and allowed) To resubmit files:
    a. Return to the Canvas submission space, click **New Attempt**, and repeat the process from Step 3.

Evaluation

Your fuzzer will be first evaluated against the three test programs with the provided `prng_seed` and `num_iterations`. Additionally, your fuzzer will be tested on a comprehensive test suite with both programs that are similar to the test targets that you have and programs that are not related to any test targets. There will be ten (10) test programs. For each test program, your fuzzer will have one (1) hour of CPU time. Crashing each target will earn you 10% credit.

You should upload your crashing input file to the challenge VM, and then run `/challenge/challenge` with your input file. You will get a flag if your input file successfully crashes `/challenge/prog` and then submit the flag.

After the due date passes, those scores and your submission will be reviewed by the course team, aligned with the course grade breakdown, and then populated in your course grade in Canvas. Please refer to the syllabus PDF and the assignment submission space in Canvas so you know how many points each assignment is worth.

Review the course syllabus for details regarding late penalties.

Bonus

You may be interested in developing a smart fuzzer, such as a coverage-guided fuzzer, or a fuzzer that is aware of string comparisons. Please contact the instructor ahead of time and get approval before starting.