Prof. Dr. Oscar Nierstrasz
Pascal Gadient, Pooja Rani

# Solution
# Assignment 12 — 02.12.2020 – v1.0
# Fuzz Testing

Please submit this exercise by email to pascal.gadient@inf.unibe.ch before 09. December 2020, 10:15am.

## Exercise 1: General questions (6 pts)

a) Explain the term *fuzz testing*. **Answer:**

*Fuzz testing is an approach to test software automatically by feeding randomized or pattern-based inputs to a target application. The application under test might crash depending on the generated input. Such crashes are indicators for code bugs and might cause a vulnerability.*

b) Explain the term *smart fuzzer*. **Answer:**

*Smart fuzzers are armed with knowledge about the expected input format. For instance, a smart fuzzer knows exactly the structure of a protocol like http or a file format like jpg. As a result, they can produce valid input with a higher certainty. Moreover, they may simply fuzz parts of the input to discover crashes in the target program.*

c) What is the role of symbolic execution engines in white box fuzzers? **Answer:**

*Unlike traditional dumb fuzzers, symbolic execution tools accurately capture the computation of each value. In symbolic execution engines,* Satisfiability Modulo Theories (SMT) *solvers help to extract data from the source code that afterwards fosters the creation of fuzzed input that can trigger more complex branches, eventually leading to a better code coverage.*

d) Explain one limitation of symbolic execution? **Answer:**

*On paper, symbolic execution can discover inputs for any feasible path in a program but this ability makes it fairly slow compared to mutation-based fuzzers and impractical in real-world scenarios. Another limitation can be the fact that loops or recursions create an infinite execution tree.*

e) Name three concolic execution engines. **Answer:**

*Savior, Qsym, and Driller.*

f) How does concolic execution extend symbolic execution? **Answer:**

*The main distinction is that concolic execution expands symbolic values with concrete values (therefore the term concolic was used). The concrete values in the starting point of a program provide a clue for the search heuristics concerning which paths to practice first.*

## Exercise 2: AFL tool (1 pt)

Answer which of the statements below are correct with respect to the *AFL* fuzzer. You do not need to justify or elaborate your answer. **Answer:**

☑ *AFL is a grey box fuzzer.*

☑ *AFL instruments the source code of the target program to measure code coverage.*

☑ *AFL also supports the QEMU mode when the source code is unavailable.*

☐ *AFL released in 2016 and became an international standard in fuzz testing.*

## Exercise 3: Fuzzing in practice (3 pts)

Preparation: For this exercise, we work with a Debian-based operating system, because all required tools are already available from its package management system. We already prepared a virtual machine (VM) for you with all the necessary tools and our compiled *VulnerableApp* that can be run with VirtualBox.

Please perform the following steps:

1. Download and install the latest version of *VirtualBox* from here (you can find the relevant downloads for popular operating systems labeled as "platform packages").
2. Download the pre-configured virtual machine (VM) from here.
3. When you start VirtualBox for the first time, it asks whether it should download an extension package. If you download and install the extension pack you will have better hardware support in the VMs, *e.g.,* USB3, etc.
4. Import the downloaded VM into VirtualBox (click on the "File" menu, then select "Import Appliance").
5. In the upcoming assistant you do not have to change any options except specifying the previously downloaded file for the import.
6. After the import succeeded, you can select the "Lubuntu" VM in the list and click on the large "Start" button to start the VM.
7. You do not need this information for this exercise, but anyway: the user name is "Playground" and the password is "1234".

Your tasks:

a) Create a text file that *only* contains the term `Fuzztesting`. Next, parametrize the *zzuf* fuzzer with a seed value of 2 and a ratio of 0.01, and pass the content of your text file (*e.g.,* test.txt) to it.

   What is the resulting command string, and what is the output message? (2 pts) **Answer:**

   *A resulting command string could be*
   ```
   zzuf -s 2 -r 0.01 cat test.txt
   ```
   *and it prints the term "BuzztesTing" to the console.*

b) Run the `VulnerableApp` program from the terminal and enter a single character within the range `a-z` or `A-Z`. Then press the return key on your keyboard to confirm your input. The program should return the ASCII code of the character you entered. Now, we want to explore whether the application crashes from arbitrary input generated by the zzuf fuzzer. Such crashes could be buffer overflows that frequently enable remote code execution. For that, we are interested in observing which seeds cause the program to crash. As starting point for the fuzzer's input generation we use a text file that *only* contains the character `a`. As before, we provide that text file as input for the fuzzer and then we pipe the output of zzuf into the *VulnerableApp*. You can achieve piping in Linux shells with the vertical bar, *e.g.,* `ls | wc` forwards the console output of `ls` to the input of `wc` (`wc` will count the words returned by `ls`). However, this time we use a different seed (starting from 20, only integer numbers) and ratio parameter (0.5).

Your task is to find seed values between 20 and 30 where *zzuf* leads to a crash in VulnerableApp. (1 pt) **Answer:**

```
#!/bin/bash
for i in {20..30}
do
  zzuf -s $i -r 0.5 cat char.txt | ./VulnerableApp
done
```

*There exist several seeds that cause the application to crash,* e.g., *2, 3, 4, 5, 6, 8, 10, 12, 13, 15, 16, 18, 19,* ***21****,* ***24****,* ***25****,* ***26****,* ***27****,* ***28****,* ***29****, 31, etc.*

## Exercise 4: Exam preparation (3 pts BONUS)

Please start reviewing the content of this course and ask questions, if any, by mail. The questions will be discussed during next practical session which will be a Q&A session for the final exam. **Deadline for questions: upcoming Monday, 07/12/2020, 23:59 (midnight).** **Answer:**

*Please have a look at the Q&A summary on the SCG webpage.*