Simulation of Fault Injection Attacks

IMPORTANT: do not distribute the files provided for this assignment!

Teacher	Email
Łukasz Chmielewski	lukchmiel@gmail.com

Goals: After completing this assignment, you will be able to analyze the resilience of a code snippet to fault injection attacks (FI). During the assignment, you will use an open-source simulator and learn how to limit amount of FI vulnerabilities in a software.

Grading: This assignment has a total number of 5 points, obtained by answering the questions below. For a breakdown of how the points are awarded, please check the questions.
When and where to submit: Submit before 15.3.2022 23:59 into IS HW vault
Soft deadline: -1.5 points for every started 24 hours
Handing in your answers:

- You can hand in your solutions via the Homework Vault in IS.
- Make sure that your name and student number are on top of the first page!
- For submitting the answers, please submit a solution in `.pdf` or `markdown` formats.
- For this assignment, you need to deliver your code in C format compatible with the provided software environment.

Before you start: To complete this assignment, you need to download the related software and the provided example. To solve the assignment, follow the below steps:

1. Install the FiSiM software. Although this is an open-source software (that you can to build for your machine), it is only fully supported on Windows. Therefore download the version released here: https://github.com/Riscure/FiSim/releases. If you do not have a Windows machine then please use the machine in the lab.
2. You can introduce yourself to FiSim here: https://github.com/Riscure/FiSim. To learn even more, you can follow a presentation covering fault attacks here:

- https://www.youtube.com/watch?v=_ZLJraOtrDA.and
- https://www.youtube.com/watch?v=7DYkV4uZqS8..

3. Download the provided example code here:
   https://mega.nz/file/h1QCRDAR#8NYZMnIfwO9J_fxPbrlDI_vNm_mIxcY8WeQPIdts57E.
   After you download the example, please copy the files in the "./Content" directory where the "./" represents the FiSim root directory.

4. Do the assignment.

5. Submit your solution including your code and report.

1 Assignment 2: Simulation of Fault Injection Attacks

The provided example code is an authentication procedure (password checker) for a secure boot-loader in an ARM microprocessor. After loading the provided example code, we do the build and verify steps, respectively. The verify step is aimed at checking the functionality of the code. As the example code is not complete, you will get "authenticated" message in the software console, which is an incorrect behavior of the boot-loader, see Fig 1. Your goal is to write code to compare two variables, simulate your code by using the FiSim, analyze the simulation results for FI vulnerabilities, and try to remove them from your code by using the simulator feedback.

1.1 Complete the code

The first step of your assignment is to write a code snippet that compares the two variables, `provided_password` and `real_password`. If they are equal, you need to assign the value "0" to the `authenticate` variable. A section to write your code is provided, as shown in Fig 1.

Note 0: Every time you change the code in the IDE, please save the file (as it's not done automatically), then clean, build and verify your code respectively.

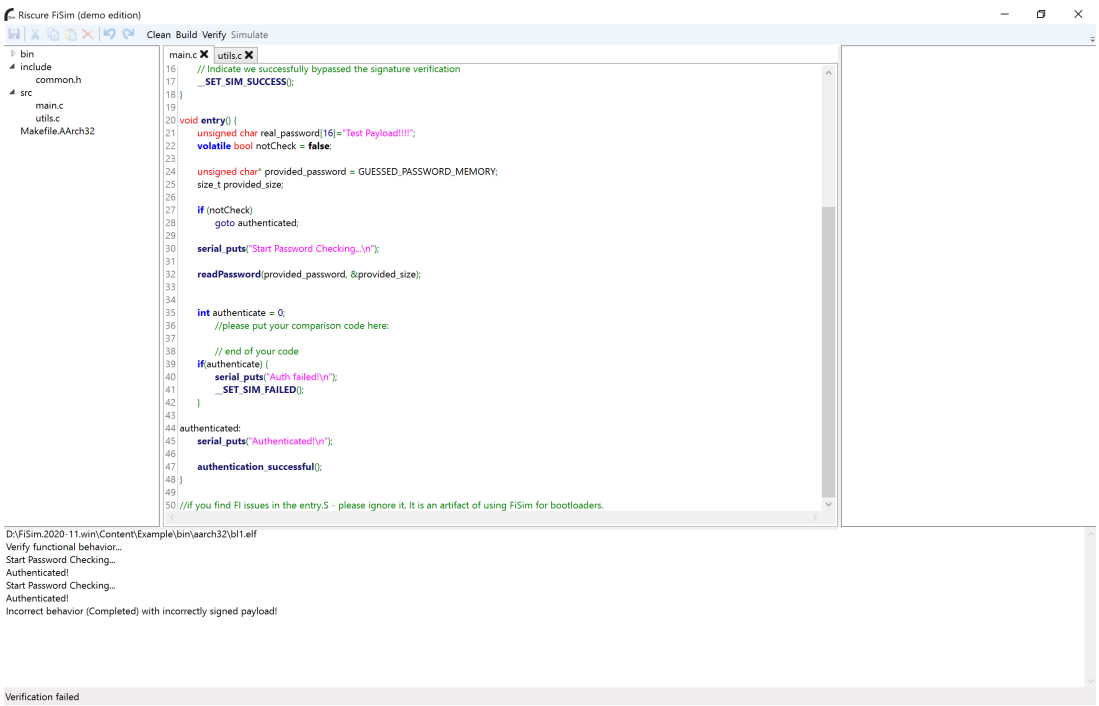

Figure 1: FiSim Example Code

1.2 Code Simulation

If your code works properly, you will get "Auth failed!" message in the console as the provided password is not the correct one and you can see the "verification finished " message in the console.
As the next step, please simulate your code and find FI vulnerabilities. When you find vulnerabilities, please explain the root cause for each vulnerability in your code.

Note 1: depending on how you wrote your code, there might be instructions for which is hard to determine the cause of fault (like NOP(2x)). We do not expect you to find the exact reason of the issue but we encourage you to experiment and find as much as you can (preferably also to avoid glitches).
Note 2: In the simulation, there are two fields for the result. You need to focus on the "TransientNopInstructionModel" part which is indicated by a red box in Fig 2. Note 3: The code is compiled with -O0 flag. Please do not change it.

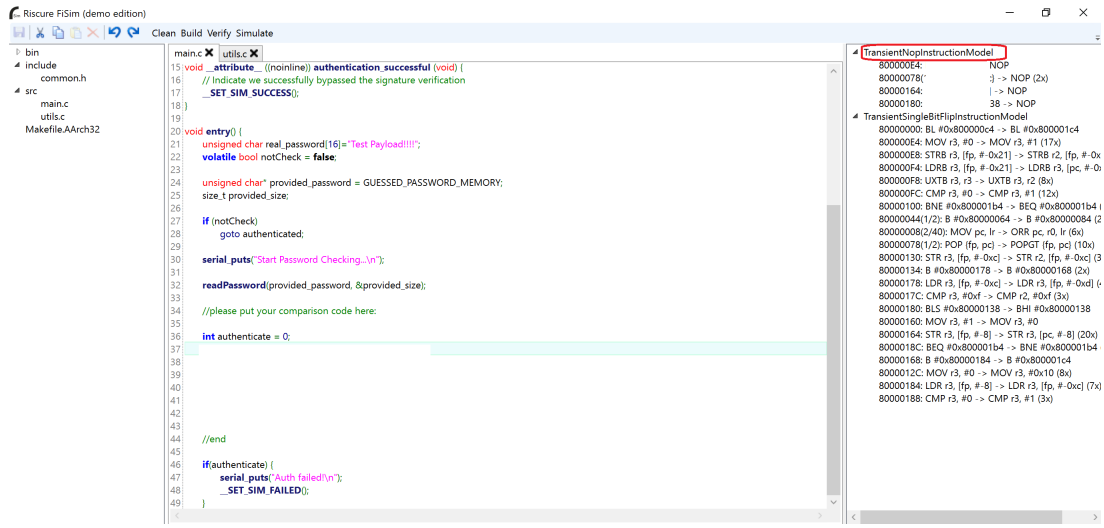

Figure 2: FiSim Result

1.3 Code Robustness

In this step, please modify your code to make it more robust against FI vulnerabilities (if there are any). As an example, you might use redundancy.

Note 4: There might be some instructions that you cannot modify easily to make the code 100% robust. As mentioned before we do not expect you to analyze these instructions in detail but it give a try to eliminate the possible glitches.

1.4 Questions

(1p) Q1: Implement the comparison code to compare two variables (`provided_password` and `real_password`). Please provide your implementation in the report you will submit.

(2p) Q2: Simulate your code with FiSim and (try to) explain the root cause for the vulnerabilities found by the software and how an attacker can take a benefit of it.

(1.5p) Q3: Modify your code to make it more robust by removing FI vulnerabilities. Please provide your **modified** implementation in the report you will submit.

(0.5p) Q4: What can you say about faults listed in "TransientSingleBitFlipInstructionModel"? Can you explain some of them? You do not need to try to eliminate them.

Note 5: For the code submission, please submit two files containing the initial and the modified version of your code in addition to the report.

Update (10/03/2022):

As pointed out by some students, the example code contains a small mistake. In the `entry` function the following variable is declared:

```
unsigned char real_password[16]="Test Payload!!!!";
```

The length of the password (16) is too short. It does not include the space for the string termination character '\0'. Hence the `real_password` is not null-terminated.

Since this issue does not affect the fault injection resistance you can either:

- leave the code as it is and perform a password comparison up to 16 characters or
- correct the issue by declaring `real_password[17]`.

Both approaches are good for me and will be considered correct.

Update (11/03/2022):

Below I include some small clarifications:

1. To harden the fault-injection resistance it is allowed to modify the code outside my original comments, namely, outside `//please put your comparison code here: ... //end`.
   It is important that the implementation protected against FI is functionally equivalent to your code containing the password check.

2. I recommend not to use any external libraries to implement password comparison. This would be cumbersome and it would be hard to determine where faults are really happening. Therefore, I suggest to write your own code for the password comparison.

3. Using "Verify" is a good first step to check whether your new code is functionally correct.