Security Testing

WS 2021/2022

Prof. Dr. Andreas Zeller
Leon Bettscheider
Marius Smytzek

Exercise 12 (10 Points)

Due: 06. February 2022

The lecture is based on The Fuzzing Book, 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.

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 exercise 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.

Exercise 12-1: The Internet Is New Land for All of Us (8 Points)

In this exercise you are a penetration tester and need to identify bugs in the registration interface of the company that hired you. The server is implemented in **server.py**.

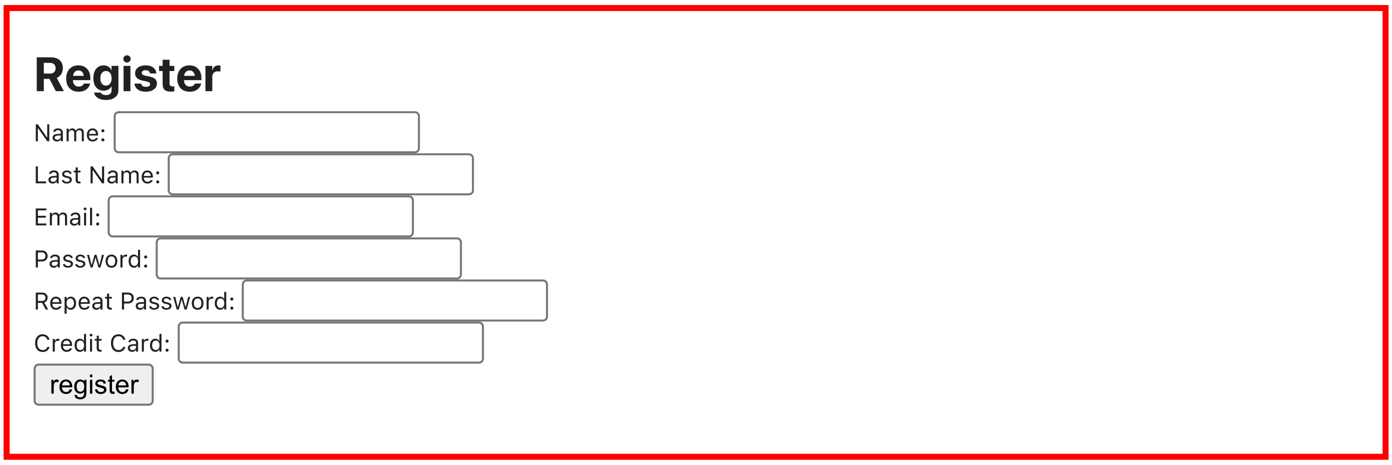
Please upgrade the `fuzzingbook` package for this exercise with `pip3 install --upgrade fuzzingbook`.

The Interface

The interface is a form that consists of the following fields:

- `name`: The first name of the user.
- `lastname`: The last name of the user.
- `email`: The email address of the user.
- `password`: The password the user wants to choose.
- `password2`: A check for the password to detect typing errors.
- `banking`: The credit card number of the user.

The interface looks like this:



Register

Name:	
Last Name:	
Email:	
Password:	
Repeat Password:	
Credit Card:	

When pressing `register` the request will be directed to the URL `/register`, that handles the registration. This is the part you should test.

A request should look like this:

```
"/register?name=Leon&lastname=Bettscheider&"
"email=leon.bettscheider@cispa.de&password=password%23&"
"password2=password2%23&banking=6983683624040672"
```

Keep in mind that you need to CGI encode special character in the values of the request. You can leverage the `cgi_encode()` function of `fuzzingbook.WebFuzzer` to encode a string.

In the following, the company has provided you some guidelines and hints.

Name and Last Name

The `name` and `lastname` should only consist of letters.

Email

The `email` should be a valid email.

Password

The `password` should contain at least one character that is not alphanumerical.

Password Check

`password` and `password2` should be equal.

Banking

The credit card number provided in `banking` can be a number of arbitrary length but the last digit should be a checksum calculated with the Luhn algorithm. You can leverage the implementation in **luhn.py**, if you want to use this.

Implementation

Please provide your solution in **exercise_1.py**. The file contains a function `get_fuzzer(httpd_url)` that should return your final fuzzer, with all configurations applied. You are allowed to use any fuzzer from the Fuzzing Book. We would recommend to stick to one of the following:

- `GrammarFuzzer`
- `ProbabilisticGrammarFuzzer`
- `GeneratorGrammarFuzzer`
- `ProbabilisticGeneratorGrammarCoverageFuzzer`
- `WebFormFuzzer`

You are allowed to be specific to the provided implementation. But you are not allowed to provide a list of requests, for instance, in a grammar. Your tool must generate requests dynamically.

You are also allowed to make your fuzzer aware of the found bugs, e.g. as a state machine.

The Bug Hunt

There are a total of *8* bugs in the server. You are allowed to check the implementation as you have full access to it from the company.

You know that other students also working on this server. You are allowed to share information about the bugs you found with others, this means you can share under which conditions the bug occurs (e.g. a bug occurs if `name` consists of more the *64* characters) and you can also share bug triggering requests if you like. You are not allowed to share parts of your solution, e.g. code, the grammar, or what techniques you use.

Evaluation

To evaluate your implementation execute `python3.9 test_1.py`. The last line printed will provide you with the number of triggered bugs. Each found bug in this exercise will reward you one point.

The found bugs will be written into **bugs.py** as a dictionary. The key is the ID of the bug. The value is the bug triggering request. If no bug is found, this file will not be created.

You are not allowed to change **server.py**, **luhn.py**, or **test_1.py**.

Exercise 12-2: Quiz (2 Points)

In this exercise we will recap the chapter on *Testing Graphical User Interfaces*.

Provide the BEST answers to the following questions in `exercise_2.py` by assigning to each variable `Q1, ..., Q4` the values `1` to `4`.

For instance, if you think the first answer is the BEST answer to Q1, set `Q1=1` in `exercise_2.py`.

There is only **one BEST answer** to each question.

Questions

Q1: What is the model classically used to encode user interface states and transitions for automated testing?

1. A finite state machine.
2. A non-deterministic pushdown automaton.
3. A context-sensitive grammar.
4. A Turing-complete programming language.

Q2: What is an advantage of encoding classical GUI models into a grammar?

1. The grammar can be used to represent an infinite number of states.
2. The grammar can be used to represent an infinite number of actions.
3. Actions and states can be represented in a single unified model.
4. It renders the model stateless thus simplifying the fuzzing process.

Q3:. Which of the following is an important assumption of the techniques for GUI model mining presented in the lecture?

1. There cannot be any loop between the states of the interface.
2. Transitions between states must be deterministic.
3. For each and every state there can exist only a single next state.
4. Each transition can only feature a single action (fill formular, submit, click, ...).

Q4:. Which of the following statements is correct?

1. GUI testing can be carried out even when the source code of the program under test is not available.
2. GUI testing is usually faster than fuzzing at the API level.
3. GUI testing cannot be used to test remote programs.
4. GUI testing always requires an input grammar.