Secure Software development: **Project 4**

Points Possible: 100
Deadline: 11:59pm Tuesday November 26th 2024

Goals:

- Creating a network fuzzer, which is a program that will try out many different URL variants to see if any exist on a web server.
- Learn HTTP status codes and handling python arguments.

Description:

Given a base URL, such as http://server.com/FUZZ, and a word list containing many items, it will replace the string FUZZ with each of the words in the word list, and see if that URL exists.

You will need a number of files from this repository to work on this assignment:

project4.py	This is where you will write the code for this assignment, and the only file that will be submitted.
args.py	This is the supporting code for project4.py that does the command-line argument parsing. You should not modify this file!
common-reduced.txt	The word list that we will be using for testing. The format is one word per line, with no spaces in the words.
server.py	The code to run a mini web server on your machine for testing.

Calling the code:

The provided code in args.py already handles the command line argument parsing. Run "python3 project4.py -h" to see the various options. You are going to write the fuzz() function in project4.py – the parameter passed into that function contains the parse command line arguments – you can print it to the screen during development to see how they are stored (but be sure to remove that print statement – and any others beyond what is required – before submission).

Writing the code:

You have to write the fuzz() function in project4.py – remove the pass line that is there once you insert your own code. The intent is for you to use the urllib.request library; documentation on that can be found here.

For the basic fuzzing, you will likely need the urllib.request.urlopen() function and the status field of the object it returns.

Task 1: Basic fuzzing

The first task is to perform basic testing. Given a URL, such as `http://server.com/FUZZ`, you will replace `FUZZ` with each word in the word list file, and then try that URL to see if you get a response other than 404 (not found). You will do this by modifying the `fuzz()` function in the project4.py skeleton code. Try printing out the parameter to the function to see how the command-line arguments are passed into the function. You can see in the skeleton code how one might load a URL from a string.

When you are ready to test this, read the 'Testing' section, below, for easy ways to test it locally on your computer.

Output

The output for this assignment is very specific. Once you have completed the above task, any URL that does NOT return a 404 should display the output in the following EXACT format. (Note that you will modify what URLs to print later).

For example, if you were to call the fuzzer as:

`python3 project4.py -u http://ffuf.me/cd/basic/FUZZ -w common-reduced.txt`, then the output would be exactly:

```
200 http://ffuf.me/cd/basic/class
200 http://ffuf.me/cd/basic/development.log
```

Note that the output is the status number, a single space, and the URL. The order you print it out the lines does not matter, only the content on each line.

We are going to use this particular call the visible test when you submit your assignment (although we will use a smaller word list to save time, but it will contain both `class` and `development.log`). Other tests will be used to grade your assignment.

DO NOT HAVE ANY OTHER OUTPUT! We are going to test it by doing a file comparison, so if you have any other output it will report as not the same, and you will fail that test. Again, the order of the lines in your output does not matter.

Task 2: Advanced fuzzing

There are a number of command-line parameters that the fuzzer.py file will accept. You have to implement usage of the others.

You can find them all via `python3 project4.py -h`. The remaining ones to implement are as follows. Note that these are already parsed for you; you just have to handle when those values are in the `args` parameter to the `fuzz()` function.

- `-e EXTENSIONS` or `--extension EXTENSIONS`: One or more extensions to append (e.g. php, html, etc.). Multiple extensions may be provided. So if `-e php -e html` is provided, and the wordlist contains `hello` and `world`, then you should be replacing `FUZZ` with six different values: `hello`, `hello.php`, `hello.html`, `world`, `world.php`, and `world.html`.
  - The value to the `-e` parameter assumes that it will be prefixed with a period before being added to each word in the word list. So `-e html` means you will add `.html` to each word in the word list. However, note that the command line parameter inserts that period for you.
  - Note that adding any number of extensions still means you try the base word as well. So adding `-e html` means your program will try both `alert` and `alert.html`.
- `-mc MATCH_CODES`: Match HTTP response codes. May be specified multiple times. If left unspecified, defaults to the following response codes: [200, 301, 302, 401, 403]. Previously you printed out any URLs that did not return 404 (not found). That should now be modified to print out the URLs that return one of the response codes in this list (which is parsed for you and passed into the `fuzz()` function in the `args` parameter).
  - Specifying just one response code via -mc will replace the default list with just that one. So `-mc 200` will not check for any of the defaults other than 200. Note that the command line argument parsing does this for you.
  - **NOTE:** this can be specified *multiple* times, at which point you would then have to check for *multiple* match codes.

When we test your code, we will never give it invalid parameters, and there will always be a -u parameter and a -w parameter. So you do not have to error-check the parameters.

Testing

First test it locally, then test it remotely. Both are described below.

Local testing

Initially, as you are developing your code, you will want to test it out locally. One of the Python packages you installed is uvicorn, which is a small Python-based web server. This file has a `urls` list that is the set of URLs that it will respond with a 200 (OK) to; everything else it responds with a 304 or 404 (not found).

You will run your uvicorn server as follows:

```
uvicorn server:app --reload --port 5000
```

On some systems (Windows without WSL, in particular), you have to put `python -m` at the start of that command.

This runs the server on port 5000 on your computer. To test against this, you will run your project4 as follows:

```
python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt
```

Note: it may be that your computer needs to use `http://127.0.0.1:5000/FUZZ` instead.

Also note: sometimes Macs have issues with port 5000. If that is the case, then try any other port (5001, 5002, etc.).

Following the output specifications described above, the output should be exactly as follows. Note that the order of the lines does not matter, as long as the format of each line is exact.

```
200 http://localhost:5000/.gitignore
200 http://localhost:5000/employers
200 http://localhost:5000/~admin
```

Remote testing

As a remote test, you can try it with the URL of http://ffuf.me/cd/basic:

```
python3 project4.py -u http://ffuf.me/cd/basic/FUZZ -mc 200 -w common-reduced.txt
```

The output would be:

```
200 http://ffuf.me/cd/basic/class
200 http://ffuf.me/cd/basic/development.log
```

Please do NOT try it out against any other machines or other websites.

Credit: this homework is based heavily on ICS: Programming HW: Fuzzing. Note that do not use the server.py from their website. That would cause your fuzz result to be incorrect from this assignment.

Programming Environment:

Python 3.
Python Package to install:
uvicorn

These are the following test cases that you should pass:

Test cases	Command	Result
1	python3 project4.py -u http://ffuf.me/cd/basic/FUZZ -w common-reduced.txt	200 http://ffuf.me/cd/basic/class 200 http://ffuf.me/cd/basic/development.log
2.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt	200 http://localhost:5000/~admin 200 http://localhost:5000/employers 200 http://localhost:5000/.gitignore
3.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -mc 304	200 http://localhost:5000/~admin 304 http://localhost:5000/award 200 http://localhost:5000/employers 200 http://localhost:5000/.gitignore 304 http://localhost:5000/research
4.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -e log	200 http://localhost:5000/~admin 200 http://localhost:5000/administrator.log 200 http://localhost:5000/employers 200 http://localhost:5000/.gitignore

5.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -e html	200 http://localhost:5000/~admin 200 http://localhost:5000/downloads.html 200 http://localhost:5000/employers 200 http://localhost:5000/employers.html 200 http://localhost:5000/.gitignore
6.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -e html -e log	200 http://localhost:5000/~admin 200 http://localhost:5000/administrator.log 200 http://localhost:5000/downloads.html 200 http://localhost:5000/employers 200 http://localhost:5000/employers.html 200 http://localhost:5000/.gitignore
7.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 304	304 http://localhost:5000/award 304 http://localhost:5000/research
8.	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 304 -e log	304 http://localhost:5000/award 304 http://localhost:5000/research 304 http://localhost:5000/research.log
9	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -mc 304 -e log	200 http://localhost:5000/~admin 200 http://localhost:5000/administrator.log 304 http://localhost:5000/award 200 http://localhost:5000/employers 200 http://localhost:5000/.gitignore 304 http://localhost:5000/research 304 http://localhost:5000/research.log
10	python3 project4.py -u http://localhost:5000/FUZZ -w common-reduced.txt -mc 200 -mc 304 -e log -e html	200 http://localhost:5000/~admin 200 http://localhost:5000/administrator.log 304 http://localhost:5000/award 200 http://localhost:5000/downloads.html 200 http://localhost:5000/employers 200 http://localhost:5000/employers.html 200 http://localhost:5000/.gitignore 304 http://localhost:5000/research 304 http://localhost:5000/research.log
11	python3 project4.py -u http://ffuf.me/cd/ext/logs/FUZZ -w common-reduced.txt -mc 200 -mc 304 -e log	200 http://ffuf.me/cd/ext/logs/users.log

Requirements:

1. (5 points) **Use comments to provide a heading at the top of your code** containing your name, NCU StudentID, and e-mail.
2. (5 points) Your source code file should be named as "**project4_LastName_ StudentID.py**". (e.g. project4_Harn_982001510.py)
3. (70 points) Test case: 1 to 8, Test case: 11
4. (10 points) Test case: 9 and Test case: 10
5. (10 points) Usability of your program (Good Comment Practices).

You will **lose points** if you: do not use the specific program file name, or do not have a comment block on **EVERY** program you hand in. You will lose **at least 40 points** if there are compilation errors or warning messages when we compile your source code.

Deliverables:

- A heading at the top of your code contains your name, NCU UserID, and e-mail., and how to compile your code.
- Submit your **source code file** named as "project4_LastName_UserID.py" through the **eeClass** system.

Late Submission Penalty:

- After the 11:59pm on the due day, you can't submit your assignment anymore. Late assignments through e-mail will receive a 20% deduction per day penalty.

Rebuttal period:

- You will be given a period of 2 business days to read and respond to the comments and grades of your homework or project assignments. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.