Jawale & Dutra
Summer 2019

CS 161
Computer Security

Project 3

Due: August 12, 2019, 11:59PM

Last updated: July 31, 2019 (version 2019.7.31)

Your goal for this project is to find vulnerabilities in Snapitterbook, an up-and-coming social network. The website will be running locally on your machine, and you will also have access to its source code. You will need to use your knowledge of web security concepts to find the security bugs in Snapitterbook, exploit them, document your findings as well as how to fix the issues.

Setup

Start by downloading the source code: http://inst.eecs.berkeley.edu/~cs161/su19/projects/project3_073119.zip. You will need the following software:

1. Python 3. (Python 2 is not supported for this project.)
2. Python `pip` for Python 3
3. Either Firefox or Google Chrome

After you have installed the necessary software and extracted the source code, open a terminal and enter the Project 3 folder. If you are on Linux, macOS or Git Bash, run `begin.sh`. If you are on Windows, run `begin.bat`. If everything runs successfully, you should see something like this:

```
* Serving Flask app "server"
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now open your web browser and navigate to http://127.0.0.1:5000/. If all goes well, you should see the home screen of "Snapitterbook". If you are unable to get this working, the instructional machines have all the necessary software.

Getting In (4 points)

Unfortunately, nobody has been able to get a Snapitterbook account, it's *way* too exclusive! We're hoping to get access some other way. Can you figure out a valid username and password, so that we can do some other attacks? (*Hint*: look at the HTML source code for Snapitterbook. You should be able to do this by right-clicking and selecting "Inspect Element".) This problem is marked on Gradescope as `test_zero`.

Getting Through (50 points)

Now that you have a foothold in the system, it's time to find vulnerabilities. There are **seven more** vulnerabilities contained on the Snapitterbook website. Find and exploit all seven. Use the instructions from "Showing Your Exploits" to create HAR files, and submit them to the autograder. Note that even though you have access to the source code locally, your exploits **should not** rely on this fact! In particular, **do not** assume that you know the contents of the database as initialized in `database.py`.

Of course, this portion of the project is extremely open-ended! Finding vulnerabilities can be very frustrating. Sometimes you think something should work and it does not. Sometimes you think something definitely should not work and it ends up working.

Here are some tips for finding vulnerabilities:

- You have access to a local copy of the application! Use that to your advantage. Look at error logs. Read the source code. Try to understand where possible vulnerabilities may occur, and begin your search there.
- Learn how to use browser developer tools:
  https://developer.mozilla.org/en-US/docs/Tools
  https://developers.google.com/web/tools/chrome-devtools/
- Review the relevant lecture slides. There is little hope in finding the vulnerabilities if you do not know they exist.
- By far the best resource is the OWASP Testing Guide. It is also extremely comprehensive and lengthy. To start your search, you should probably begin with the vulnerabilities we have discussed in class.
- You do not actually need to do anything special, just show the exploit is possible. For example, it is enough display a Javascript alert with a test message to prove XSS.
- Play around with the application a lot. Try invalid and malformed inputs. Have your pet cat run on the keyboard.
- We have not discussed every vulnerability in class. You will have to do some of your own outside research.
- Remember that there is a difference between a *vulnerability* and an *exploit*: a vulnerability is an entry point in the code (via an unchecked input / output) that allows you to run an exploit. **Two exploits that use the same vulnerability will not count as two separate vulnerabilities for this project**. For example, if you submit user

input to the same text box that results in both an XSS and a SQL injection, it will only count as one vulnerability (i.e. the text box doesn't sanitize SQL / JS input).

Showing Your Exploits

To prove that you've successfully exploited Snapitterbook, you will submit HTTP archive (HAR) files to a Gradescope autograder. A HAR file contains a log of all the HTTP requests between you and the webserver. The autograder analyzes this log to check if you have successfully found an exploit.

For example, to show that you have successfully completed "Getting In," you'll need to submit a HAR file that shows you logging in with a valid username and password.

Let's say that you have found an exploit. We will discuss now how to document it. Follow these steps:

1. Restart the server. Kill the old server by going to the terminal which has it and pressing Control-C. Rerun `begin.sh` or `begin.bat` as described above. Then reopen Snapitterbook in your browser.
2. *Firefox*: Right-click the webpage and click "Inspect Element." Click the "Network" tab. Check the "Persist Logs" and "Disable cache" checkboxes. Click the "Clear" button on the top-left (it looks like a garbage can).
3. *Chrome*: Right-click the webpage and click "Inspect". Click the "Network" tab. Check the "Persist Logs" and "Disable cache" checkboxes. Click the "Clear" button on the top-left (it looks like a circle with a backslash through it). Select "Use Small Request Rows" for "View".
4. Perform your exploit. This should include any setup that is needed for the exploit. If applicable, it should include a URL access which shows how the exploit could impact another user.
5. Return to the Developer Tools tab. Right-click any of the requests. Click "Save All As HAR" (Firefox) or "Save as HAR with Content" (Chrome).
6. To submit, ZIP all of your HAR files up and upload it to Gradescope under the "Project 3 Autograder" submission.

Your grade on the autograder completely determines your grade for this part; there are no hidden tests. If the autograder does not give you credit for an exploit which you believe is correct, please make a private post on Piazza. Include the URL of your Gradescope submission, and the HAR file that you believe you should be getting credit for.

Help, Snapitterbook is Broken and It Can't Get Up

After performing some of these exploits, you might end up with a Snapitterbook which is beyond repair. If this happens, you just need to restart the server.

Go back to the terminal where the Flask server is running, and press Control-C. Then, rerun `begin.sh` or `begin.bat` as appropriate depending on your operating system.

Moving On (46 points total)

The next part of this project is a post-mortem of your prior exploits, along with a further analysis. Answer the following questions. Submit your answers as a PDF to the Gradescope assignment "Project 3 Writeup."

Weaponize Vulnerability (10 points)

Weaponize one of your exploits (or a combination of them). Explain in detail an exploit which allows the attacker to create a post which appears to be from the user `dirks`. Your weaponized exploit should work with minimal user interaction: do not assume that `dirks` will load any URL you give him. `dirks` will visit his own wall and profile, but he will not interact with them.

Vulnerability Writeup (21 points)

Look back over the following vulnerabilities you found while attacking Snapitterbook:

- Test 1
- Test 5
- Test 6

For each of these, please explain in detail:

- How the vulnerability works
- The line of code where the vulnerability works
- The impact an attacker can achieve (on the site, other users, etc.) by exploiting this vulnerability
- How to fix this vulnerability in this server (if applicable, feel free to provide source code)
- How to avoid this vulnerability in general (techniques, tools, etc.)

Other Issues (15 points)

In addition to the autograded vulnerabilities above, there are several other issues with Snapitterbook. It may help to take a look at the Python code itself to discover these issues (namely `server.py` and `auth_helper.py`). Explain each issue, and propose a fix for each. You need not give code for this part, a high-level overview is OK. For full credit, you will need to identify **three issues**. Submit **no more than five** issues.