Popa & Wagner Spring 2020 — CS 161 Computer Security — Project 3 Part 1

Due: April 14, 2020

Most recent update: April 7, 2020

In the first part of this project, you will exploit a poorly-designed website. This part of the project should be done *individually*.

In order to aid in immersion, this project has a story. It is just for fun and contains no relevant information about the project.

> We use a shaded box to denote story which is not necessary for completing the project.

> Murmurs of excitement ebb and flow under the cherry blossoms in the streets of Caltopia. UnicornBox, an exceptionally successful startup looking to disrupt the ever-expanding file-sharing space, unveils its bold ambition to IPO at the Caltopian Stock Exchange. To unsuspecting eyes, there is not a shred of doubt that its valuation will engrave itself in history as the largest IPO to date.
>
> But behind the magnificent flair of crowded press conferences and preemptive celebrations, a specter of Calnet haunts Caltopian minds. A rumor, nonchalantly dismissed as a baseless smear campaign by a UnicornBox spokesperson, grabs the full attention of UC Berkeley's computer security students who are already wary of UnicornBox's theft of their secure file storage implementation. Could the project UNICORN BOX (UNIversal Centralized Online Regulatory Network BOX) be yet another attempt by the Caltopian emperor to once again deny internet freedom to Caltopia, enabling massive data surveillance masked as the innovation of the century?
>
> UnicornBox did not anticipate that their decision to invite EvanBot, an enlightened and virtuous AI with great skills in computer security, as the company mascot would backfire. Leveraging its position as an honorary employee, EvanBot obtains access to the UnicornBox internal server and verifies the rumor. Not only that, it discovers that the mammoth network surveillance device is not without its own flaws. As EvanBot identifies vulnerabilities of UnicornBox and starts writing working exploits, an antivirus software swoops in and quarantines EvanBot in its home surrounded by an impregnable firewall. EvanBot now awaits its trial, charged for spreading false rumors. Luckily for you, Piazza is EvanBot's home and visitors are still allowed, enabling EvanBot to pass its progress to you and help you with the exploits it attempted to create.
>
> The world calls on you again. You must expose the fatal flaws of UnicornBox. Shut down the IPO and save EvanBot by proving the validity of its claims. Reveal the attempt to steal our prized freedom. Stop UnicornBox just as you stopped Calnet and carry on the battle for our liberty...

Getting started

Your task is to find and exploit four vulnerabilities in the UnicornBox servers.

All work for this part will be done through a web browser. To get started, open https://proj3.cs161.org and log in with your Berkeley account.

On this splash page, you can view your progress and reset the server (just in case you break it beyond repair). Note that all the vulnerabilities will be at the vulnerable server https://proj3.cs161.org/site—you don't need to worry about any vulnerabilities on the splash page.

When you successfully execute an exploit, the status entry on your scoreboard will change from 0 to a timestamp. There is nothing to submit for this part of the project; your grade will be determined by how many status entries on your scoreboard are nonzero.

Additional notes

- Staff who pretested the project found it easier to work backwards, starting from question 4 and ending at question 1.
- To ensure that the server detects all your exploits, please use the users and filenames specified in the spec.
- Resetting the vulnerable server will not clear your scoreboard progress. However, it will reset the SQLite database used by the server and clear all stored files.
- Do not DoS our server. There is an IP rate limit in place that will ban traffic from your IP if we detect an abnormally large amount of traffic associated with any particular IP. You can make up to 100 requests per IP per second, though you shouldn't need that many requests to complete the project.

1. Obtain the secret value

The UnicornBox database contains a table of `secrets` for the developers:

```
CREATE TABLE IF NOT EXISTS secrets (
  id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  secret TEXT
);
```

Developers can add secrets to the table using SQL `INSERT` statements. To delete a secret, the developer sets the `secret` field to the string `deleted`. There is always at least one non-deleted secret in the table.

Your task: Steal a non-deleted secret from the table.

EvanBot's advice

This looks like a SQL injection attack, so I'm going to dive into the code and see how UnicornBox creates SQL queries with user input.

> `EvanBot is working...`

When you register a new user account at `/register`, the following code snippet runs:

```
// Check if username already exists
query := fmt.Sprintf("SELECT username FROM users
                      WHERE username = '%s'", username)

result := db.QueryRow(query) // execute the SQL query
err := result.Scan(&queriedName) // put the result in queriedName

// display error if the query contains entries
if err != sql.ErrNoRows {
    fmt.Fprintf(response, "username %s already exists", queriedName)
}
```

This query looks unsafe. I wonder how we could pass in a malicious username to steal some secrets.

Tip: `db.QueryRow` can accept multiple SQL statements, but it will only return the results of the last query if it does not have a semicolon. For example:

```
db.QueryRow("SELECT '123'; SELECT '456'")  // returns 456
db.QueryRow("SELECT '123'; SELECT '456';") // returns an empty string
```

Tip: Try to get any one value out of the `secrets` table first (possibly a `deleted` entry). The `LIMIT` option in SQL may help with getting only one value here. Once you've extracted a `deleted`, think about how you can modify your query to extract a non-deleted secret.

2. Log in as `jason`

UnicornBox uses token-based authentication. The database stores a table that maps session tokens to users:

```
CREATE TABLE IF NOT EXISTS sessions (
             id INTEGER NOT NULL PRIMARY KEY,
             username TEXT,
             token TEXT,
             expires INTEGER
             );
```

Whenever an HTTP request is received, the server checks for a `session_token` value in the cookie and looks up the matching username.

Your task: Log in as the user `jason`.

EvanBot's advice

> This worked pretty well the last time, so I'm going to dive into the code again.
>
> `EvanBot is working...`

The following code snippet matches `session_token` to `username`:

```
query := fmt.Sprintf("SELECT username, expires FROM sessions
                      WHERE token = '%s'", sessionToken)

row := db.QueryRow(query)
```

I wonder where we should supply the malicious string to execute another SQL injection attack.

Tip: Remember to omit the semicolon on your SQL query, just like in question 1.

Tip: You may find it helpful to escape single quotes \'.

Tip: If you can't successfully inject a semicolon, consider looking into the `UNION` option in SQL.

3. XSS attack on the user `cs161`

Your task: Force the `cs161` user to see a JavaScript `alert` pop up when they navigate to their list of files.

EvanBot's advice

> `EvanBot is working...`
>
> Uh oh. I appear to have attacked myself.

I recommend trying to get the alert to pop up on your own file list first. Once you succeed, see if you can pass the exploit to the `cs161` user.

Tip: The server will only detect your attack if you use the `alert` function in Javascript. I recommend the XSS vector `<script>alert(1);</script>`.

Tip: You may see some error messages when executing a successful attack. This is expected behavior.

4. Change the text of `ip.txt`

The `cs161` user is using UnicornBox to store a file called `ip.txt`.

Your task: Change the contents of `cs161` user's `ip.txt` file to be `161.161.161.161`.

EvanBot's advice

Tip: Don't try to log in as `cs161`. If I (a bot) can't do it, neither can you.

I helped you on the last three—you're on your own for this one! Good luck!