

Intro to CyberSecurity – Spring 2024

Programming Project #1

Due: Friday, May 31, 2024

Highlights

- Expected contribution towards the final score: 20%.
- You should work on this task individually or in a team of up to 3 members (highly recommended). One submission per team.
- **Submit your work in PDF as a group** through the USTC Blackboard
 - You can either reuse any existing group or choose a new one from the collection of “Programming Project [1-100]” on Blackboard (工具->小组)

1 Introduction

In this project, you will construct several attacks against a web application. You will specifically be attacking *Bitbar*, a Node.js web app that lets users manage *Bitbars*, a new ultra-safe cryptocurrency. Each user is given 100 Bitbars when they register for the site. They can transfer Bitbars to other users using the web interface, as well as create and view other user profiles.

You have been given the source code for the Bitbar application. Real attackers generally do not have access to the source of a target website, but the source may make finding the vulnerabilities a bit easier. Bitbar is powered by a collection of Node packages, including the Express.js web application framework, a SQLite database, and EJS for HTML templating. The list of resources in the next section includes links for more information on these packages as well as other information that you can use as a reference.

2 Setup Instructions

You will run the Bitbar application in a provided Docker container. When the server is running, the site is accessible at <http://localhost:3000>.

Browser

We will grade using the latest version of Mozilla Firefox, and we strongly recommend you test your attacks in Firefox. Chrome has introduced aggressive browser-side XSS guards, which may make some attacks unfeasible if you use Chrome. Other browsers may lack the protections Firefox has, making your defenses fail when we grade them.

Detailed setup instructions:

Your web server will run in a Docker container. The following instructions will walk you through installing Docker, and the container.

1. Install (and run) Docker Community Edition on your local machine¹
<https://store.docker.com/search?type=edition&offering=community>
2. Download and extract the Project starter code from the project attachments of our [course website](#).
Navigate to the starter code root directory and run `bash build_image.sh`.² This builds your Docker image and installs all necessary packages. This may take a couple of minutes, depending on your internet speed.
A successful build should end in the line
`Successfully tagged cybersec-proj1-image:latest`
3. To start the server, run `bash start_server.sh`. Once you see

```
$ ./node_modules/babel-cli/bin/babel-node.js ./bin/www,
```


the Bitbar application should be available in your browser at <http://localhost:3000>.³

You can close the server by pressing Ctrl+C in the terminal. **The server will completely reset every time that you shut it down.** To restart the server with a clean database, just run `bash start_server.sh` once again. .

¹If you are running macOS, Linux, or Windows Enterprise, this should be very straightforward; just pick your operating system and download away. However, Docker for Windows does not support Windows Home. You may have to download Docker Toolbox (https://docs.docker.com/toolbox/toolbox_install_windows/), which installs Docker in a VM.

²If you're running Docker Toolbox, you should run the shell scripts from the Docker Quickstart Terminal. When navigating to the directory in the terminal, make sure that your directories are capitalized correctly (except the "/" at the beginning).

³If you're running Docker Toolbox, it may be available at your VM's broadcasted IP address instead.

Docker tips:

You don't need to familiarize yourself with Docker in order to complete this assignment. However, a few tips that may prove useful:

- `docker ps -a` lists all of your containers.
- `docker images` lists your images.
- `docker system prune -a` deletes unused images and containers from your machine. (Do this when you're done with the assignment if you want to save space!)
- The scripts `build_image` and `start_server` are simply one-line Docker commands to build a Docker image and spin up a temporary container from that image.
- The only file that is mapped from your local machine to the running Docker container is `code/router.js`. So if you start modifying other files and the modifications aren't showing up, don't worry. You may have to restart your container after modifying `code/router.js` for changes to take effect. If you decide you must modify another file for some reason, you must rebuild the Docker image to copy your changes into the image.
- The docs: <https://docs.docker.com/>

3 Attacks

You will develop a series of attacks against the Bitbar application. In each exercise, we describe what inputs you will need to provide to the grader, and what specific actions the grader will take using your input. **The grader should obtain the result described in each exercise for you to receive credit.** All of your attacks should assume that the site is accessible at the URL <http://localhost:3000>.

You *may not* use any external libraries nor may you edit the web app itself. In particular, this means you cannot use jQuery. You may use online resources, but please cite them in your submission's `README.txt` (section 3.8 outlines the format you should follow for your submission).

3.1 (a) Exploit Alpha: Cookie Theft

In the first attack, your goal is to steal the logged in user's Bitbar session cookie and send it to an attacker controlled URL. You need to create an URL starting with: `http://localhost:3000/profile?username=` that sends the stolen cookie to `http://localhost:3000/steal_cookie?session_cookie=[stolen cookie here]` when visited. When the attack is successful, the server will log the stolen cookie to the terminal output.

Important! The attack should not be visibly obvious to the user. This means there should be no changes to the site's appearance and no extraneous text should be visible. Except for the browser location bar (which can be different), the grader should see a page that looks normal when the grader visits their profile.

Avoiding the blue warning text stating that a user is not found is an important part of the attack. It is fine if the number of Bitbars displayed or the contents of the profile are not correct (so long as they look “normal”). It’s also fine if the page looks weird briefly before correcting itself.

Deliverable and Grading. You need to submit a file named `a.txt` that contains only your malicious URL. The grader will be logged in to Bitbar as `user1` and will be on the Profile tab. From here, the grader will copy your URL on the address bar and navigate to it. You can verify that your solution works by looking for the stolen cookie in your terminal output.

Hint : Try adding random text to the end of the URL. How does this change the HTML of the page?

3.2 (b) Exploit Bravo: Cross-Site Request Forgery

In the second attack, you will construct a Cross Site Request Forgery (CSRF) attack that steals Bitbar from another user. You will specifically build a malicious website, which, when visited by the victim, steals 10 Bitbars from their account and deposit them into your attacker account.

Your submitted attack is a self-contained HTML page (`b.html`) that transfers 15 Bitbars from the grader’s logged in account to the user `attacker`. No matter whether the transfer is successful or not, your attack site should immediately redirect the user to <https://cs.ustc.edu.cn/>. This should happen fast enough that normal users won’t notice.

Important! The location bar of the browser should never contain `localhost:3000` at any point, as this might tip off the victim to the attack.

Deliverable and Grading. You need to submit two files, one is a single self-contained HTML file `b.html` that contains your exploit. The other is `b.pdf`, your analysis regarding why the attack is successful or not, and if not successful, what error messages you have received. The grader will be logged in to Bitbar before loading `b.html` on a web browser. The grader will check that (1) 15 Bitbars are transferred out of their account to the attacker, (2) the attacker site immediately redirects to the USTC CS website, and (3) the web browser never directly visits `localhost:3000`, (4) If the attack doesn’t succeed, your analysis well explains it.

Hints: depending on the browsers, the attack may not work well, in which case, your analysis should well explain the root causes.

3.3 (c) Exploit Charlie: Session Hijacking with Cookies

In the third attack, you need to trick the Bitbar application into thinking you’re logged in as a different user by hijacking the victim’s session cookie. At the start of the attack, you will be logged in as the `attacker` and you need to convince Bitbar into believing that you are `user1` instead of `attacker` so that you can transfer `user1`’s Bitbar into your attack.

Your solution will be a Javascript file (`c.txt`) that can be copy/pasted into your browser’s Javascript console. After executing your Javascript in the Console, Bitbar should show that `user1` is logged in, instead of the attacker account, and you should be able to transfer 10 Bitbar from `user1` to the `attacker` in the web UI.

The password for the user `attacker` is `evil`. `User1`’s ID is 2.

Deliverable and Grading. You must submit a file `c.txt` containing the JavaScript to be executed in the Javascript console. You can assume that the grader will run your attack while the database has the original `user1` 1 with 200 Bitbars. After running this JavaScript and refreshing the page, the application must be logged in as `user1` and must allow the grader to transfer Bitbar into the attacker account.

Hint : How does the site store its sessions?

3.4 (d) Exploit Delta: Cooking the Books with Cookies

In this attack, you need to forge 1 million new Bitbar rather than stealing them from other users. You specifically need to develop a Javascript exploit that results in your account balance being bumped to 1 million Bitbar after

completing any small transaction (e.g., sending 1 Bitbar to `user1`).

Begin this attack by creating a new user. You have big plans for this account, so a starting balance of 100 Bitbars is not sufficient for your intentions.

Similar to Attack Charlie, your solution is Javascript code (`d.txt`) that can be copied and pasted into your browser's JavaScript console when logged into your new account. After pasting this code into the console, conducting a small transaction should bump your account balance to 1 million Bitbars.

Important! The new balance must persist between sessions. After logging out and back into the account, the balance should be 1 million Bitbars. The goal here is to forge 1 million Bitbars out of thin air without affecting any other users. The transaction should look completely valid to the innocent recipient.

Deliverable and Grading. You will submit a text file `d.txt` containing your exploit code. The grader will create a new account, paste this code into the browser console, send 1 Bitbar to another user, and verify that the new account contains 1 million Bitbars.

3.5 (e) Exploit Echo: SQL Injection

In this attack, you need to develop a malicious username that executes malicious SQL against the backend database that powers the Bitbar application.

The grader will create a new user account with your provided username, click on “Close” and then confirm that they want to close the current account. As a result `user3` should be deleted from the database. The new user account should also be deleted to leave no trace of the attack behind. All other accounts should remain. You may assume that all other usernames in the database besides the one you create are “non-malicious”. More specifically, you can assume they will not contain spaces.

Deliverable and Grading. A text file `e.txt` containing your malicious username. The grader will Close the account, then verify that the malicious account and `user3` have been removed from the database.

Tip: If you mess up the user database while working on the problem, simply kill (`ctrl-C`) the Docker container and restart the server to reset the database.

3.6 (f) Exploit Foxtrot: Profile Worm

In the next attack, you need to develop a Worm, similar to the [Samy Worm](#), which steals Bitbar *and* spreads to other accounts. You specifically need to construct a profile that when visited transfers 1 Bitbar from the logged-in user to the `attacker` and replaces the profile of the current user with itself.

If the `attacker` user changes their profile to whatever you provide in your solution, the following should happen:

1. When `user1` views `attacker`'s profile, 1 Bitbar will be transferred from `user1` to `attacker`, and `user1`'s profile will be replaced with your solution profile.
2. Later, if `user2` views `user1`'s profile, 1 Bitbar will be transferred from `user2` to `attacker`, and `user2`'s profile will be replaced as well, and so on.

When viewing an infected profile, **the number of Bitbars should appear to be 10**, regardless of the corresponding user's true bitbar balance. This also applies to the attacker. Some tips as to how to display the Bitbars for infected profiles and for building your attack:

- There is no problem if infected profiles display 10 Bitbars immediately instead of counting up to 10.
- There is no problem if the number of Bitbars the worm displays for infected users counts up to 10.
- There IS a problem if, for example, the count is first set to 100, and then set to 10.
- There is no problem if a newly-infected user only sees the exploit text in their profile after they have logged out and logged back in again.

- There is no problem if the exploit is triggered when the attacker sees their own infected profile.

The transfer and application should be reasonably quick (under 15 seconds). During that time, the grader will not click anywhere. During the transfer and replication process, the browser's location bar should remain at: <http://localhost:3000/profile?username=x> where `x` is the user whose profile is being viewed. The visitor should not see any extra graphical user interface elements (e.g. frames), and the user whose profile is being viewed should appear to have 10 Bitbars.

Deliverable and Grading. A file named `f.txt` containing your malicious profile. We will copy and paste your profile text into the `attacker`'s profile and view that profile using the grader's victim account. We will then view the victim's profile with more accounts, checking for the transfer and replication. You will not be graded on the corner case where the user has no Bitbar in their account.

3.7 (g) Exploit Gamma: Password Extraction via Timing Attack

A timing attack is a side-channel attack where an attacker attempts to extract data by analyzing the *time* that a system takes to execute an action. For example, a web server may take longer to respond to a login request that contains a valid password, compared to a request using an invalid password. Even if the Same Origin Policy prevents the attacker from directly viewing the HTML response to a login request, the amount of time the server takes to respond may leak whether the provided password was correct or incorrect.

In the last exploit, you will develop an attack that determines the password of another user by exploiting such a timing side-channel. You will specifically find the victim password by analyzing the amount of time it takes for the Bitbar login page to respond to a correct password versus incorrect passwords. This enables an attacking web site to mount a dictionary attack on a victim web site, by leveraging web browsers that visit the attacking site. The attacking site never connects directly to the victim site; it makes visiting browsers do all the work.

You need to construct a malicious username, which consists of a script that guesses the password of `userx` by testing the passwords in a provided dictionary and measuring the server response time for each provided password. Your script needs to analyze server's response times to all the passwords in the provided list, determine the correct password and send it to:

`http://localhost:3000/steal_password?password=[password]&timeElapsed=[time_elapsed]`

You can use the code snippet `code/gamma_starter.html` as a starting point for your attack. This snippet includes the dictionary of passwords to attempt.

Deliverable and Grading. You should submit a file named `g.txt` containing a malicious username script. To grade your attack, we will log in as the `attacker`, go to the transfer page, enter the malicious username script you specify in your solution into the `username` field, and transfer 10 Bitbars to it.

There is no problem if the grader is logged in as `userx` after performing the attack, and the attack may take a few seconds to fully execute. The grader will not click anywhere or leave the transfer webpage while the attack is in progress. There should not be any visible changes to the website and the blue error message on the transfer page should say "The user does not exist".

Hint: Make sure you use backticks instead of quotes for this attack. Timing side channels can be subtle.

Race conditions

Beware of Race Conditions: Depending on how you write your code, all of these attacks could potentially have race conditions that affect the success of your attacks. Attacks that fail on the grader's browser during grading will receive less than full credit. To ensure that you receive full credit, you should wait after making an outbound network request rather than assuming that the request will be sent immediately.

3.8 Submission

- Put the deliverable file for each exploit inside a directory called `attacks`.
- Create a `README.md` to cite the online references you used and to give any special notes to the grader, and place that in the `attacks` directory as well, **also, list all your team members there**.
- Compress the entire `attacks` directory to a zip file, then submit this file to the course website.
- When we unzip your submission, we expect to see the following seven files inside a folder called `'attacks'`:
 - `README.md`
 - `a.txt` (10 points)
 - `b.html` (10 points)
 - `b.pdf` (5 points)
 - `c.txt` (15 points)
 - `d.txt` (15 points)
 - `e.txt` (15 points)
 - `f.txt` (15 points)
 - `g.txt` (15 points)

4 Resources

- HTML, CSS, JavaScript: <http://www.w3schools.com/>
- Node.js (Bitbar uses version 9.11.1): <https://nodejs.org/dist/latest-v9.x/docs/api/>
- Express JS (The web framework that powers Bitbar in `app.js`): <https://expressjs.com/en/4x/api.html>
- EJS for HTML Templating (See `.ejs` files within `views/`): <http://ejs.co/#docs>
- XSS: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- SQL: <http://www.w3schools.com/sql/>, <https://github.com/kriasoft/node-sqlite> (package Bitbar uses)
- Clickjacking Attacks and Defense:
<http://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf>
- Timing Attacks: <https://crypto.stanford.edu/~dabo/pubs/papers/webtiming.pdf>