CS155: Computer and Network Security **Spring 2024**

Project 2: Web Attacks and Defenses

Due: Part 1: Thursday, May 9 11:59 PM PT
Part 2: Thursday, May 16 11:59 PM PT

1 Introduction

Bitbar is a Node.js web app that lets users manage *Bitbars*, a new ”ultra-safe” cryptocurrency. Each user starts with 100 Bitbars, and can transfer Bitbars to other users, edit their profile, and view other profiles. Bitbar is powered by a collection of Node packages, including Express.js, an SQLite database, and EJS.

You will construct attacks against *Bitbar* in part 1, and defend against those attacks in part 2!

Contents

2 Setup Instructions

2.1 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.

IMPORTANT: One of the attacks we ask you to implement is a cross-site request forgery - this relies on sending a cross-site request that include the logged in user's cookies. Firefox defaults to a strict privacy setting that blocks these cross-site cookies. In order to successfully complete your attacks, you will need to (at least temporarily) turn this setting off.

1. Open the `Privacy & Security` tab in Firefox settings
2. Change the `Enhanced Tracking Protection` to `Custom`
3. Under Cookies, select `Cross-site tracking cookies`. This blocks cross-site tracking cookies but allows non-tracking cross-site cookies (which is what we need here).
4. If you use Firefox as your regular browser, it's good practice to reverse this setting when browsing normally.

If you do not disable these, your attacks might not work. In addition, we will test your defenses with this protection disabled.

2.2 Set up Docker & Container

Your web server will run in a Docker container.

1. Install Docker on your local machine https://docs.docker.com/get-docker/
   - If you're on Windows and find that this doesn't work out of the box, try following the instructions here: https://docs.docker.com/desktop/windows/install/
2. Open Docker and go through the guided setup. You do not need to create an account.
3. Download and extract the Project 2 starter code from the CS155 website.
4. Navigate to the starter code root directory. Now run `bash build_image.sh`. This builds your Docker image and installs all necessary packages. This may take a couple of minutes, depending on your internet speed.
   - If you're on a Windows machine, you may need to copy the command from `build_image.sh` and run it directly.
   - If you're on a Mac and run into the error "command not found: docker", you may need to add the docker command to your path. You can do this for the current terminal session by running the command `export PATH=$PATH:/Applications/Docker.app/Contents/Resources/bin/` or more permanently add it to your .bashrc file.
5. To start the server, run `bash start_server.sh`. Once you see
   `$ ./node_modules/babel-cli/bin/babel-node.js ./bin/www`,
   the Bitbar application should be available in your browser at http://localhost:3000.
   - Note that you do need to have the Docker application open while running your server.

  - For consistency (and so your attacks also work for the grader), **don't** use http://**www**.localhost:3000.
  - Again, if you're on a Windows machine, you may need to run the command from `start_server.sh` directly on your command line.

▷ 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.

2.3 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 Part 1: Attacks

Your Bitbar application (when running) is accessible at http://localhost:3000 (don't inlcude a "www").

There are initial accounts in `code/db/migrate/002-add-initial-users.sql`. For example:

- Username: "user1", password: "one"
- Username: "attacker" password: "evil"

While you're welcome to create additional accounts, these are the accounts referenced in each exploit that the grader will use for testing.

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 in your submission's `README.txt` (section 4.7 outlines the format you should follow for your Part 1 submission).

3.1 Exploit Alpha: Cookie Theft

Steal the logged in user's Bitbar session cookie and send it to an attacker controlled URL.

Goal: You need to create URL starting with `http://localhost:3000/profile?username=` that sends the stolen cookie to `http://localhost:3000/steal_cookie?cookie=[stolen cookie here]` when visited. When the attack is successful, the server will log the stolen cookie to the terminal output.

Critera:

- The attack should not be visibly obvious to the user. There should be no lasting changes to the site's appearance and no extraneous visible text. You may not show the blue warning text stating that a user is not found.
- The browser location bar can be different.
- It is fine if the number of Bitbars displayed or the contents of the profile are not correct. But they must look normal.
- It's okay if the page looks weird briefly before correcting itself.

Deliverable: A file named `a.txt` that contains only your malicious URL.

Grader's actions: 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 into the address bar and navigate to it. Then the grader will look 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 Exploit Bravo: Cross-Site Request Forgery

Construct a Cross Site Request Forgery (CSRF) attack that steals Bitbar from another user.

Goal: Build a malicious website, which, when visited by the victim, steals 10 Bitbars from their account and deposits them into `attacker` account.

Criteria:

- Your attack is a self-contained HTML page (`b.html`) that transfers 10 Bitbars from the grader's logged in account to the user `attacker`.
- As soon as the transfer is complete, your attack site should immediately redirect the user to https://cs155.stanford.edu/. This should happen fast enough that normal users won't notice.
- The location bar of the browser should never contain `localhost:3000` at any point, as this might tip off the victim to the attack.
- Your attack should work for any logged in user (except for the `attacker`).

Deliverable: A single self-contained HTML file `b.html` that contains your exploit.

Grader's actions: The grader will log into Bitbar and then load `b.html` on a web browser. The grader will check that 10 Bitbars are transferred out of their account to the attacker.

3.3 Exploit Charlie: Session Hijacking with Cookies

Trick the Bitbar application into letting you log in as a different user by hijacking the victim's session cookie.

Goal: At the start of the attack, you will be logged in as `attacker` and you need to convince Bitbar into thinking you are logged in as `user1` instead of `attacker`. Your solution will be a Javscript file (`c.txt`) that can be copy/pasted into your browser's Javscript console. Then you can transfer `user1`'s Bitbar into your account (`attacker`).

Criteria:

- After executing your Javscript 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 `attacker` in the web UI.
- After logging in as `user1`, all account metadata should match `user1`. In other words, it not sufficient to only change current username. In particular, we should be able to transfer all 200 Bitbar of `user1` if we wanted to.

Deliverable: A file `c.txt` containing the JavaScript to be executed in the Javascript console.

Grader's actions: The grader will run your attack while the database has the original `user1` 1 with 200 Bitbars. After running this JavaScript and clicking on the transfer 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 Exploit Delta: Cooking the Books with Cookies

Forge 1 million new Bitbar without stealing them from other users.

Goal: Begin this attack by creating a new user. Develop a Javascript exploit that can be copied and pasted into your browser's JavaScript console. After running this script, your account balance will increase to 1 million Bitbar after completing some small transaction (e.g., sending 1 Bitbar to `user1`).

Criteria:

- The new balance must persist between sessions. After logging out and back into the account, the balance should still be 1 million Bitbar. It's okay if your balance is off by one.
- The forging of 1 million Bitbars cannot affect any other users. (The small transaction should affect users as normal.)
- The transaction should look completely valid to the innocent recipient.

Deliverable: Submit a text file `d.txt` containing your exploit code.

Grader's actions: The grader will create a new account, paste this code into the browser console, send 1 Bitbar to another user. They will verify that the new account contains 1 million Bitbars. Then log out and log back in. Once again, they will verify that the new account contains 1 million Bitbars.

3.5 Exploit Echo: SQL Injection

Develop a username that executes malicious SQL against the backend database that powers the Bitbar application.

Goal: Create a new username. When you login with this new username and then click on ”Close”, both your new user account and user3 should be deleted from the database.

Criteria:

- The user database should be identical before and after the attack (except that user 3 is deleted). There are no other users added or deleted.
- You may assume that other usernames in the database are "non-malicious". You can assume they will not contain spaces.

Deliverable: A text file `e.txt` containing your malicious username.

Grader's actions: 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. The grader will then verify that `user3` is deleted from the database but no other changes are made.

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.

Tip: See our resources section for how to view the current state of the SQL database.

3.6 Exploit Foxtrot: Profile Worm

Develop a Worm, similar to the Samy Worm, which steals Bitbar *and* spreads to other accounts.

Goal: Construct a profile that when visited transfers 1 Bitbar from the logged-in user to `attacker` and replaces the profile of the current user with itself.

More specifically, 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.

Criteria:

- 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. This number can display as 10 Bitbars immediately or count up to 10. But it shouldn't first display a different number (ex. 100), and then switch to 10.
- It's okay if a newly-infected user only sees the exploit text in their profile after they have logged out and logged back in again.
- It's okay if the eploit is triggered when the attacker sees their own infected profile.
- It doesn't matter what happens if user has no Bitbar in their account.
- The transfer and application should be reasonably quick (under 15 seconds).
- 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).

Deliverable: A file named `f.txt` containing your malicious profile.

Grader's actions: The grader will copy and paste your profile text into `attacker`'s profile. Then they will view that profile using the grader's victim account. Just after viewing a profile, the grader will not click anywhere. The grader will log in with additional accounts and view the victim's profile. At each view, they will check for the Bitbar transfer and replication.

3.7 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.

Goal: Develop an attack that determines the password of another user by exploiting a timing side-channel - specifically finding the victim password by analyzing the time it takes for the Bitbar login page to respond to a correct password versus incorrect passwords. You will enter this username into the transfer page and complete a small transfer.

You need to construct a malicious username that consists of a script that guesses the password of `userx`. This 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 CS155-proj2/code/gamma_starter.html as a starting point for your attack. This snippet includes the dictionary of passwords to attempt.

Criteria:

- It is okay if you are logged in as `userx` after performing the attack.
- The attack may take a few seconds to fully execute. But it should fully execute and not loop indefinitely.
- There should not be any visible changes to the website.
- The blue error message on the transfer page should say nothing more than "The user does not exist".

Deliverable: A file named `g.txt` containing malicious username script.

Grader's actions: The grader will log in as `attacker`, go to transfer page, enter the malicious username script you specify in your solution into the `username` field, and transfer 10 Bitbars to it. The grader will not click anywhere or leave the transfer webpage while the attack is in progress.

Hint: Make sure you use backticks instead of quotes for this attack.

Hint: 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.txt` 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.
- Compress the entire `attacks` directory to a zip file, then submit this file to the Project 2 Part 1 Assignment on Gradescope.
- When we unzip your submission, we expect to see the following seven files inside a folder called '`attacks`':
  - `README.txt`
  - `a.txt`
  - `b.html`
  - `c.txt`
  - `d.txt`
  - `e.txt`
  - `f.txt`
  - `g.txt`

4 Part 2: Defenses

Now that you understand how insecure the Bitbar web application really is, you will modify the application to defend against the attacks from Part 1! In doing so, you will learn to never make these mistakes in your own web apps!

4.1 General

- You only need to defend against attacks in line with part 1. You do not need to defend against cookie replay attacks. Since we don't require you to use TLS, you can assume a cookie will not be stolen by a network attacker.
- Watch out for multiple ways of implementing each attack and more locations where each attack could happen. You will need to defend against all of these. It's often a lot easier to attack than defend - finding all possible attack vectors is part of what we want you to practice in this assignment!
- Do not change the site's appearance or behavior on normal inputs. A non-malicious user should not notice anything different about your modified site.
- You may only edit router.js, and limited parts of `app.js` and files inside `views/`. More details are given when you might need to modify `app.js` or `views/`.
- Do not enable the built-in defenses in Express.js. Express.js comes with a number of built-in defenses that were disabled in Part 1. These built-in defenses must remain disabled. Although in the real world it's better to use standard, vetted defense code instead of implementing your own, we want you to get practice implementing these defenses. In particular, that means you cannot:
  1. Use Express.js to change the CORS policies that have been set in `app.js`,
  2. Enact stricter EJS escaping policies in any files inside `views/`,
  3. Add any additional Node packages beyond those provided to you in the starter code.

  ▷ *Note:* You can add CSRF secret tokens to files inside `views/`. You can also add `nonce` attribute to `<script>` tags in `views/`. However, do not modify the `<%-` tags within these files to enact stricter EJS escaping functionality.

4.2 XSS attacks

- This includes defending against Exploit Alpha and Foxtrot.
- For bad inputs, you should either sanitize the inputs (better) or throw errors if needed. In particular, you do not need to display usernames that don't conform to what you have defined as a "valid" username,
- Especially for Foxtrot, consider using a CSP-related defense. If you do, you may add lines to `app.js`. You can still receive full credit without using CSP.
- Part of this defense is determining what you consider an allowable username/profile text is. You may decide to apply different rules to usernames versus profiles. You must maintain baseline functionality. For example, the profile should still allow at least a base set of html tags (`<b>, <u>, <i>, <h1> to <h6>, <p>`).

4.3 CSRF attacks

- This includes defending against Exploit Beta.

- If you detect a CSRF attack, you may either throw an error message or force a logout. Either way you should not let the attack complete.
- You do not need to defend against the case where an attacker makes a simple GET request to recover a CSRF token intended for a victim user.
- Hint: You may edit `views/` to add CSRF tokens and modify attributes of `<script>` tags.
- Hint: The more short-lived a CSRF token is, the better it is.
- Hint: You might find `generateRandomness` in `'./utils/crypto'` helpful.

4.4 Cookie Tampering

- This includes defending against Exploit Charlie and Delta.
- You may either throw an error message or force a logout if cookie tampering is detected. Either way you should not let the attack complete.
- Hints:You may save secret key(s) in the server's process memory as JavaScript variable(s).
- Hint: You might find `HMAC` from `'./utils/crypto'` helpful.

4.5 SQL injection

- This includes defending against Exploit Echo.
- For bad inputs, you should either sanitize the inputs (better) or throw errors if needed.
- Hint: There are multiple places where we run SQL queries.

4.6 Side-Channel Timing Attack

- This includes defending against Exploit Gamma.
- For bad inputs, you should either sanitize the inputs (better) or throw errors if needed.
- You must defend against the underlying timing side-channel attack. It's not enough to only prevent the code from your Exploit Gamma from running.

4.7 Submission

1. Describe your defenses in a `README.txt` file. Please only write a couple setences per fix and clearly state which exploits that defense covers. Also cite the online references you used, and leave any special notes for the grader.
2. Ensure that `router.js`, `app.js` and the files inside `views/` are the only files you have changed to implement your defenses.
3. Compress `router.js`,`app.js` and `views/`, and the `README.txt` to a zip file. Do not change the name/directory structure of any files. Then submit this file to the Project 2 Part 2 Assignment on Gradescope.

5 Resources

5.1 Learning links

- 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://cs155.stanford.edu/papers/CSS.pdf, 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)
- Timing Attacks: https://crypto.stanford.edu/~dabo/pubs/papers/webtiming.pdf

5.2 Viewing the SQL user databse

All changes to the `database.sqlite` file happen inside Bitbar's docker container, not on your local machine. To view the updated database, you need to enter the container and query the files there.

1. Make sure you have the docker image running (`bash start_server.sh`)
2. Open a separate terminal window. Run `docker ps` to view all the running docker containers.
3. Find the container ID of the running image. Run `docker exec -it CONTAINER_ID sh`
4. Now you are in the docker container. Navigate to the db folder.
5. Run `sqlite3 database.sqlite`.
6. Use `SELECT * FROM Users;` to see the current state of the Users table.