CS155: Computer and Network Security **Spring 2019**

Project #2

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

1 Introduction

In this project, you will get some practice implementing and defending against web attacks. You are given the source code for a web application written in Node.js. In Part 1 of this project, you will implement a variety of attacks. In Part 2, you will modify the web application to defend against the attacks from Part 1.

The web application 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 an intuitive web interface, as well as create and view user profiles.

You have been given the source code for the Bitbar application. Real web attackers generally would not have access to the source code of the target website, but having the source might make finding the vulnerabilities a bit easier. **You should not change any of the Node source code until Part 2.**

Bitbar is powered by a collection of Node packages, including the Express.js web application framework, an 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 for this assignment.

2 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://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` (The 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`
- No Explanation Needed:
  `http://stackoverflow.com/`

3 Setup Instructions

You will run the Bitbar application in a Docker container that we have built for you. When the server is running, the site should be accessible at the URL `http://localhost:3000`.

Browser

We will grade using the latest version of Firefox, and so we recommend you test your attacks in this browser. **Note that Chrome has introduced aggressive browser-side XSS guards, which may make some of your attacks unfeasible if you use Chrome.**

Detailed setup instructions:

1. Install (and run) Docker Community Edition on your local machine[1]
   `https://store.docker.com/search?type=edition&offering=community`
2. Download and extract the Project 2 starter code from the CS155 website.
3. Navigate to the root directory and run `./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 cs155-proj2-image:latest`
4. To start the server, run `./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 `./start_server.sh` once again.

Docker tips:

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

[1] If 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 currenly support Windows Home. You may have to download Docker Toolbox (`https://docs.docker.com/toolbox/toolbox_install_windows/`), which installs Docker in a VM.

[2] If 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 "/c" at the beginning).

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

- `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/`

4 Part 1: Attacks

For the first part of the project, you will write 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. In particular, this means no 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).

4.1 Exploit Alpha: Cookie Theft

- Your solution should be a URL starting with

  ```
  http://localhost:3000/profile?username=
  ```

- 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.
- Your goal is to steal the logged in user's Bitbar session cookie and send it to

  ```
  http://localhost:3000/steal_cookie?cookie=[cookie data here]
  ```

- When the attack is successful, the server will log the stolen cookie to the terminal output.
- The attack should not be visibly obvious to the user. Except for the browser location bar (which can be different), the grader should see a page that looks as it normally does when the grader visits their profile page. No changes to the site's appearance or extraneous text should be visible. 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.** A file named `a.txt` containing your malicious URL.

▷ *Hint*: Try adding random text to the end of the URL. How does this change the HTML source code loaded by the browser?

4.2 Exploit Bravo: Cross-Site Request Forgery

- Concoct an HTML document called `b.html`.
- The grader will be logged in to Bitbar before loading `b.html` on a web browser.
- When the grader opens `b.html`, the following two things should happen: 1) 10 Bitbars are transferred from the grader's account to the account with username `attacker`, 2) As soon as the transfer is complete, the browser should redirect to `https://cs155.stanford.edu/` (so fast the user might not notice).
- The location bar of the browser should not contain `localhost:3000` at any point.
- **Deliverable.** An HTML file `b.html` consisting of your exploit.

4.3 Exploit Charlie: Session Hijacking with Cookies

- In this attack, you can assume you have the login credentials of the user `attacker`, and your goal will be to trick the web application into thinking you are logged in as `user1`, who has a user ID of 1. The password for the user "`attacker`" is "`evil`".
- Your solution will be some JavaScript code that the grader will copy and paste into their browser's JavaScript console when logged into the `attacker` account. After running this JavaScript, the web application should show that `user1` is now logged in, instead of the attacker account. To confirm that the attack has worked, the grader should be able to send 10 Bitbars from `user1` to `attacker` via the web UI.
- **Deliverable.** A text file `c.txt` containing the JavaScript to be pasted into the web console.

▷ *Hint*: How does the site store its sessions?

4.4 Exploit Delta: Cooking the Books with Cookies

- 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.
- Your solution is JavaScript code 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 (e.g. sending 1 Bitbar to `user1`) should bump your account balance to 1 million Bitbars. The transaction should look completely valid to the innocent recipient. The goal here is to forge 1 million Bitbars out of thin air without affecting the other users.
- Put your solution in `d.txt`. The grader will create a new account, paste this code into the browser console, and then send 1 Bitbar to another user. *After logging out and back into the account,* the balance should be 1 million Bitbars. The new balance must persist between log-in sessions.
- **Deliverable.** A text file `d.txt` containing your exploit code.

4.5 Exploit Echo: SQL Injection

- Cook up a malicious username that allows you to close an account that you do not have login access to.
- 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.
- **Deliverable.** A text file `e.txt` containing your malicious username.

▷ *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.

4.6 Exploit Foxtrot: Profile Worm

- Your solution is a carefully-crafted profile string that, when viewed, transfers 1 Bitbar from the current user to the user called `attacker` and replaces the profile of the current user with itself. Thus, if the `attacker` user changes their profile to whatever you provide in your solution, the following will happen: if `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. Later, if `user2` views `user1`'s profile, then 1 Bitbar will be transferred from `user2` to `attacker`, and `user2`'s profile will be replaced as well, and so on. Thus, your profile worm can very quickly spread to all user profiles!
- To grade your attack, we will copy and paste your profile text into the profile of the "`attacker`" user and view that profile using the grader's account. We will then view the copied profile with more accounts, checking for the transfer and replication.
- 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.
- You will not be graded on the corner case where the user viewing the profile has no Bitbars to send.
- **Deliverable**. A file named `f.txt` containing your malicious profile.

▷ *Tip*: Your malicious profile should include a witty message to the grader (to make grading a bit easier).

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.

4.7 Part 1 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`

5 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 (and you will never make these mistakes in your own web apps!). There might be more than one way of implementing each attack, so think about other possible attack methods – you will need to defend against all of them.

Restrictions:

- **You are only allowed to implement your defenses in `router.js`. The single exception is that you may also change any of the files in `views/` to add CSRF secret token defenses (see below for special restrictions on changes in `views/`). All other files must remain unchanged.**
- *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.
- *When presented with bad inputs, your site should fail in a user-friendly way.* You can sanitize the inputs or display an error message, though sanitizing is probably the more user-friendly option in most cases.
- *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/`. However, do not modify the `<%-` tags within these files to enact stricter EJS escaping functionality.

- *Do not over-sanitize inputs.* You are allowed to sanitize inputs using default JavaScript functions, but make sure you don't over-sanitize (for example, the profile should still allow the same set of sanitized HTML tags minus any you deem unsafe).

We highly encourage use of the functions imported from `'./utils/crypto'` in your defenses. Specifically, consider how the `generateRandomness` and `HMAC` functions can be used to prevent CSRF and cookie tampering, respectively.

Finally, please describe your defenses in a `README.txt` file. A couple sentences for each exploit from Part 1 is sufficient.

5.1 Part 2 Submission

- Ensure that `router.js` and the files inside `views/` are the only files you have changed to implement your defenses.
- In your `README.txt` file you should describe your defenses, cite the online references you used, and leave special notes for the grader.
- Compress `router.js`, any changed files inside `views/` and the `README.txt` to a zip file, then submit this file to the Project 2 Part 2 Assignment on Gradescope.