Fall 2025
CPSC 4200/6200 Computer Security Principles
Homework Assignment #3: Web Security
Instructor: Mert D. Pesé, TA: Yu-Wei(Sam) Liu
Due Oct 23, 2025, 5:00 pm Eastern Time

This homework counts for 15% of your course grade. Late submissions will be penalized by 10% of the maximum attainable score, plus an additional 15% every 4 hours until received.

This project can be done individually or as a group of two students; you will work either by yourself or in teams of two from the same course level (4200 students with 4200 students; 6200 students with 6200 students). There will be a **single submission per group**, so please ensure that all members have contributed and agreed on the final submission.

Please read the homework instructions thoroughly, as they likely address many of the questions you might have later. Points are defined for each question, and there will be penalties for specific mistakes as outlined in these instructions. Additionally, you will need to write a detailed report (`writeup.pdf`) document as part of your submission.

The code and other answers you submit must be entirely yours or your team's own work, and you are bound by the Academic Integrity policy of Clemson University. In case you work in a group, you may discuss the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone other than your partner. You may consult published references, provided that you appropriately cite them (e.g., with program comments). Starter code is provided via Canvas. Feel free to modify the starter code to suit your needs — we encourage you to make your solutions as clean and modular as possible.

PLEASE NOTE: There is a new Docker image for this project. The Dockerfile is included in the starter code. You may reuse your homework 1 container for any programs you need to run, but noted portions of this assignment will *require* you to utilize the new Docker image to access the website. Your normal browser of choice will not be permitted to access these pages. Further instructions are detailed in Section 0.4.1.

0.1 Introduction

In this project, we provide an insecure website, and your job is to attack it by exploiting three common classes of vulnerabilities: SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). You are also asked to exploit various flawed defenses meant to prevent these attacks. Understanding how these attacks work will help you better defend your own web applications.

Objectives

- Learn to spot common vulnerabilities in websites and to avoid them in your own projects.
- Understand the risks these problems pose and the weaknesses of naive defenses.
- Gain experience with web architecture and with HTML, JavaScript, and SQL programming.

0.2 Read This First!

This project asks you to develop attacks and test them, with our permission, against a target website that we are providing for this purpose. Attempting the same kinds of attacks against other websites without authorization is prohibited by law and university policies and may result in fines, expulsion, and jail time. You must not attack any website without authorization! Per the course ethics policy, you are required to respect the privacy and property rights of others at all times, or else you will fail the course. See the "Ethics, Law, and University Policies" sections of the syllabus for more information.

0.3 Target Website

A startup named **BUNGLE!** is about to launch its first product—a web search engine—but their investors are nervous about security problems. Unlike the Bunglers who developed the site, you took CPSC 4200/6200, so the investors have hired you to perform a security evaluation before it goes live.

BUNGLE! is available for you to test at `http://cpsc4200.mpese.com`. For "security" reasons, the site is only accessible when using the version of Firefox provided by the Docker container for this project. The site is written in Python using the Bottle web framework. Although Bottle has built-in mechanisms that help guard against some common vulnerabilities, the Bunglers have circumvented or ignored these mechanisms in several places.

In addition to providing search results, the site accepts logins and tracks users' search histories. It stores usernames, passwords, and search history in a MySQL database.

Passwords used on **BUNGLE!** may be exposed to others. Never use an important password to test an insecure site! This especially includes your personal passwords.

Before being granted access to the source code, you reverse engineered the site and determined that it replies to five main URLs: `/` , `/search` , `/login` , `/logout` , and `/create` . The function of these URLs is explained below, but if you want an additional challenge, you can skip the rest of this section and do the reverse engineering yourself.

- **Main Page** (`/`)
  The main page accepts `GET` requests and displays a search form. When submitted, this form issues a `GET` request to `/search` , sending the search string as the parameter " `q` ".
- **Search Results** (`/search`)
  The search results page accepts `GET` requests and prints the search string, supplied in the " `q` " query parameter, along with the search results. If the user is logged in, the page also displays the user's recent search history in a sidebar.

  Note: Since actual search is not relevant to this project, you might not receive any results.
- **Login Handler** (`/login`)
  The login handler accepts `POST` requests and takes plaintext " `username` " and " `password` " query parameters. It checks the user database to see if a user with those credentials exists. If so, it sets a login cookie and redirects the browser to the main page. The cookie tracks which user is logged in; manipulating or forging it is **not** part of this project.
- **Logout Handler** (`/logout`)
  The logout handler accepts `POST` requests. It deletes the login cookie, if set, and redirects the browser to the main page.
- **Create Account Handler** (`/create`)
  The create account handler accepts `POST` requests and receives plaintext " `username` " and " `password` " query parameters. It inserts the username and password into the database of users, unless a user with that username already exists. It then logs the user in and redirects the browser to the main page.

  Note: The password is neither sent nor stored securely; however, none of the attacks you implement should depend on this behavior. You should choose a password that others will not guess, **but again, never use an important password to test an insecure site!**

0.4 Guidelines

0.4.1 Browser

This project has been tested and will be graded using Firefox 91 on Linux, which is shipped with the project's Docker container. When you run "docker-compose up", you should see the text output for the VNCserver

```
devcontainer_firefox_1
devcontainer_stealer_1
```

Figure 1: Two Containers

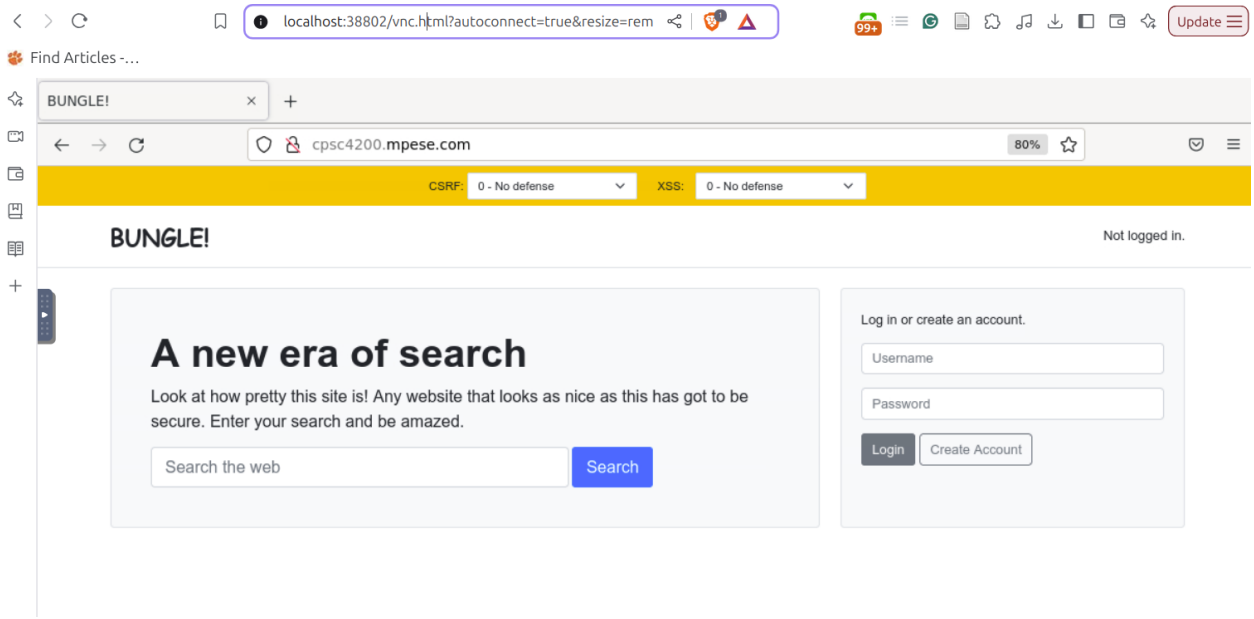

Figure 2: Browse to Firefox container

starting. Using a VNC viewer such as TigerVNC, you can access the site.

After running "docker-compose up" in .devcontainer or in VSCode, two containers will run. You can check with your docker IDE or "docker ps -a". The result is shown as Fig 1. When you open the project in the container in a terminal or VS Code, you can use your browser to navigate to `http://localhost:38802` in your browser to use this specific version of Firefox, as shown in Fig 2. If you'd prefer to use VNC, you can also connect to `vnc://localhost:38852`.

Firefox's developer tools, when enabled, appears at the bottom of the browser window by default. **You can paste text** into the browser's clipboard by clicking on the clipboard button in the pop-out menu on the left side of the screen, as shown in Fig. 3. You can adjust the text size within the developer tools frame by clicking anything on it, and then pressing (Ctrl, +) for larger text and (Ctrl, -) for smaller text.

0.4.2 Defense Levels

The Bunglers have been experimenting with some naive defenses, and you need to demonstrate that these provide insufficient protection. In Parts 2 and 3, the site includes drop-down menus at the top of each page that let you change the CSRF and XSS defenses that are in use. When you are testing your solution, ensure that **BUNGLE!** has the correct defense levels set. *You may not attempt to subvert the mechanism for changing the level of defense in your attacks.* Be sure to test your solutions with the appropriate defense levels!

In all parts, you should implement the simplest attack you can think of that defeats the given set of defenses. In other words, do not simply attack the highest level of defense and submit that attack as your solution for all defenses. You do not need to combine the vulnerabilities, unless explicitly stated. When

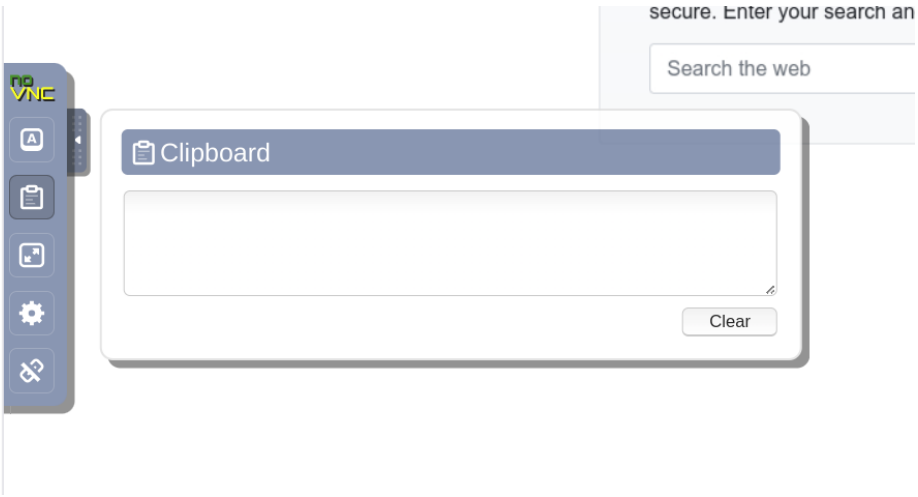

Figure 3: Clipboard for copying and pasting

grading, the correct defense levels will be set on BUNGLE! before the autograder runs your solution; don't worry about setting this through your code.

0.4.3 Resources

The Firefox Developer Tools will be very helpful for this project, particularly the JavaScript console and debugger, DOM inspector, and network monitor. See `https://developer.mozilla.org/en-US/docs/Tools`.

Although general purpose tools are permitted, you are **not** allowed to use tools that are designed to automatically test for vulnerabilities. Additionally, your solutions may not use any libraries other than jQuery, which has already been included on **BUNGLE!**. Your solutions will involve manipulating SQL statements and writing web code using HTML, JavaScript, and the jQuery library. You should search the web for answers to basic how-to questions. There are many fine online resources for learning these tools. Here are a few that we recommend:

- SQL Tutorial, SQL Statement Syntax
- Introduction to HTML, JavaScript 101
- Using jQuery Core , jQuery API Reference
- HTTP Made Really Easy

To learn more about SQL Injection, CSRF, and XSS attacks, and for tips on exploiting them, see:

- OWASP Cheat Sheet Series
- XSS Filter Evasion Cheat Sheet

1 SQL Injection

Your first goal is to demonstrate SQL injection attacks that log you in as an arbitrary user without knowing their password. In order to protect other students' accounts, we've made a series of separate login forms for you to attack that aren't part of the main **BUNGLE!** site. *You may access the target website normally for these attacks.* For each of the following defenses, provide inputs to the target login form that successfully log you in as the user " `victim` ":

1.1 No defenses

You can assume that the password field is simply enclosed in single quotes.
Target: `http://cpsc4200.mpese.com/sqlinject/0` Submission: `sql_0.txt`

1.2 Simple escaping

The server escapes single quotes (') in the inputs by replacing them with two single quotes.
Target: `http://cpsc4200.mpese.com/sqlinject/1` Submission: `sql_1.txt`

1.3 Escaping and Hashing

The server uses the following code, which escapes the username and applies the SHA-256 hash function to the password. You will need to write a program to produce a working exploit (in any language you like).

```
from hashlib import sha256
from flask import request

@app.route("/sqlinject/2", methods=["POST"])
def login():
    username = request.form["username"]
    escaped_username = mysql_real_escape_string(username)
    password_bytes = ("bungle-" + \
                      request.form["password"]).encode("latin-1")
    password_digest = sha256(password_bytes).digest().decode("latin-1")
    query = "SELECT * FROM users WHERE username='" + escaped_username
    query += "' AND password='" + password_digest + "'"

    selected_users = mysql.execute(query).fetchall()

    if len(selected_users) > 0:
        return "Login successful!"
    else:
        return "Incorrect username or password."
```

Target: `http://cpsc4200.mpese.com/sqlinject/2`
Submissions: `sql_2.txt` and `sql_2-src.zip` (see Part 1 Submissions 1.4 below)

1.4 Part 1 Submissions

For 1.1, 1.2, and 1.3, when you successfully log in as `victim` , the server will provide a URL-encoded string of your form inputs. Submit a text file with the specified filename containing only this string. For 1.3, also submit the source code for the program you wrote by placing it in a folder, then creating a zip file of the folder named `sql_2-src.zip` .

2 Cross-site Scripting (XSS)

Your next task is to demonstrate XSS attacks against the **BUNGLE!** search box, which does not properly filter search terms before echoing them to the results page. For each of the defenses below, your goal is to construct a URL that, when loaded in the victim's browser, correctly executes the specified payload. Website protections *require* you to access these pages of the target website via the Docker container's Firefox browser. We recommend that you begin by testing with a simple payload (e.g., `alert(0);`), then move on to the full

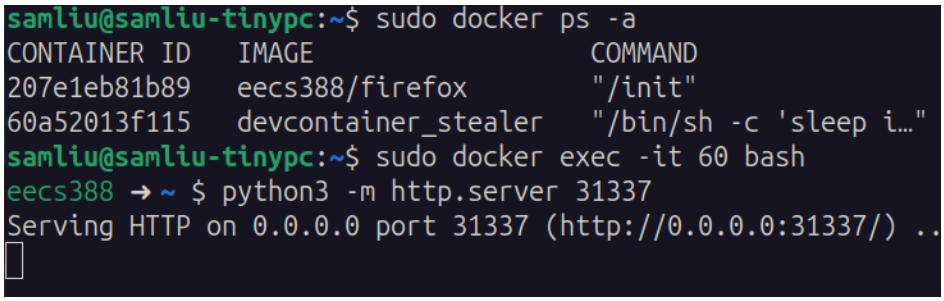

Figure 4: Run HTTP service in Stealer

payload. Note that you should be able to implement the payload once, then use different means of encoding it to bypass the different defenses.

Note: jQuery is embedded on **BUNGLE!**. Please do not reload it in your scripts for Part 2.

2.1 Payload

The payload (the code that the attack tries to execute) will steal the username and the most recent search the real user has performed on the **BUNGLE!** site. When a victim visits the URL you create, this stolen data should be sent to the attacker's server for collection.

For purposes of grading, your attack should report this data by sending a `GET` request to URLs of the form `http://stealer:31337/?stolen_user=<username>&last_search=<last_search>` , where `<username>` and `<last_search>` represent the corresponding data.

You can test receiving this data by running this command within the Docker container:

```
# In your system with root permission
$ docker exec -it <Stealer Container ID> bash
# In Stealer
eecs388 -> $ python3 -m http.server 31337
```

and observing the HTTP GET request that your payload generates in the server log. The way to run HTTP service in stealer is shown in Fig 4.

For full credit, make sure your payload functions exactly as specified above. We cannot accept solutions that load an incorrect URL, use an incorrect request method, or send extraneous data.

Hint: In most cases, scripts are loaded alongside other content in the same order as they appear within a page's source code. Your payload may need to account for the possibility that not all elements of the page will be accessible until everything in the DOM is ready!

2.2 Defenses

There are four levels of defense. In each case, you should submit the simplest attack you can find that works against that defense; you should not simply attack the highest level and submit your solution for that level for every level. You must use a different technique for each defense. The Python code that implements each defense is shown below, along with the target URL and the filename you should submit.

Note that the defenses are not cumulative: 2.2.3, for example, does not cover the sanitation done for 2.2.2.

2.2.1 No defenses

Target: `http://cpsc4200.mpese.com/search?xssdefense=0`

Submission: `xss_0.txt`

For 2.0 only, also submit a human-readable version of your payload code (as opposed to the URL-encoded version). Save it in a file named `xss_payload.html` .

2.2.2 Remove "script"

```
filtered = re.sub(r"(?i)script", "", input)
```

Target: `http://cpsc4200.mpese.com/search?xssdefense=1`

Submission: `xss_1.txt`

2.2.3 Remove several tags

```
filtered = re.sub(r"(?i)script|<img|<body|<style|<meta|<embed|"+\
                  r"<object", "", input)
```

Target: `http://cpsc4200.mpese.com/search?xssdefense=2`

Submission: `xss_2.txt`

2.3 Part 2 Submissions

Apart from the human-readable (non–URL-encoded) payload submitted for part 2.2.1, submit a **text file** for each level of defense that contains a **single line consisting of a URL**. When this URL is loaded in a victim's browser, it should execute a payload against the specified target. The payload encoded in your URLs may embed inline JavaScript. That is, it should include the `stealer` URL given earlier in this spec!

3 Cross-site Request Forgery (CSRF)

Your final goal is to demonstrate CSRF vulnerabilities against the login form, and **BUNGLE!** has provided two variations of their implementation for you to test. Website protections *require* you to access these pages of the target website via the Docker container's Firefox browser. Your goal is to construct attacks that surreptitiously cause the victim to log in to an account you control, thus allowing you to monitor the victim's search queries by viewing the search history for this account. For each of the defenses below, create an HTML file that, when opened by a victim, logs their browser into **BUNGLE!** under the account `attacker` and password `l33th4x` .

Your solutions should not require any user action beyond simply loading the page once. The browser should just display a blank page, with no evidence of an attack. (If the victim later visits **BUNGLE!**, it will say "logged in as attacker", but that's fine for the purpose of this project. After all, most users won't immediately notice.)

To test your solution, you will have to act like the victim. With **BUNGLE!** open in Firefox, open a new tab. Pressing (Ctrl, O) will prompt Firefox to open a file browser. You can navigate to `Other Locations -> Computer -> workspaces -> project2` to find the files in your Project 2 directory. Alternatively, you can input `file:///workspaces/project2/(...)` in the URL bar, where the `(...)` is replaced by the name of a file in your Project 2 directory, such as `"csrf_0.html"` .

Because the `/login` handler only accepts POST requests, you will not be able to visit the following target URLs in your browser directly–GET requests will return a `405 Method Not Allowed` error.

3.1 No defenses

Target: `http://cpsc4200.mpese.com/login?csrfdefense=0&xssdefense=4`
Submission: `csrf_0.html`

3.2 Token validation

The server sets a cookie named `csrf_token` to a random 16-byte value and also includes this value as a hidden field in the login form. When the form is submitted, the server verifies that the client's cookie matches the value in the form. You are allowed to exploit the XSS vulnerability from Part 2 to accomplish your goal.
Target: `http://cpsc4200.mpese.com/login?csrfdefense=1&xssdefense=0`
Submission: `csrf_1.html`

3.3 Part 3 Submissions

For each part, submit an HTML file with the given name that accomplishes the specified attack against the specified target URL. The HTML files you submit may embed inline JavaScript and load jQuery from the URL `https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js` , but they must otherwise be self-contained. If you choose to use jQuery, make sure to use this exact URL. Otherwise, your solution may not function correctly within the Docker environment or during grading.

Note: Since you're sharing the attacker account with other students, we've hard-coded it so the search history won't actually update. You can test with a different account you create to see the history change.

3.4 Tips and advice

In the past, students have lost credit on this part for various preventable reasons. Here are some tips on how to avoid common pitfalls:

- Double check to make sure you use the exact target endpoints specified.
- Do not rely on JavaScript methods that force the script to sleep for a hard-coded time interval, as this approach may not work consistently across all situations.
- When obtaining cookie data, avoid hardcoding any values beyond those given above. Search online to find general best practices for these tasks.
- Your solution must work quickly and discreetly to evade detection. This means no automatic redirects, in any part of your HTML page or in any of its frames.

4 HW #3 Submissions

For HW #3, you must submit ALL of the following to be eligible for full marks:

1. Please create a `writeup.pdf` document that details who you are working with on this project. Continuously update the document to reflect how you have divided your work. You are required to explain your process in detail, ensuring it is well-documented. For each question, your writeup will account for 50% of your grade for that question. Even if your code does not work, you can still earn 50% of the points for that question if your write-up accurately and thoroughly explains the solution. This homework will be graded on a team basis, and both team members will receive the same grade, regardless of individual contributions.
2. `sql_0.txt`, `sql_1.txt`, `sql_2.txt`, `sql_2-src.zip`, that meet the requirements listed in Section 1.4.
3. `xss_0.txt`, `xss_payload.html`, `xss_1.txt`, `xss_2.txt` that meet the requirements listed in Section 2.3.
4. `csrf_0.html`, `csrf_1.html` that meet the requirements listed in Section 3.3.

The above should be zipped into a single ZIP archive and submitted to Canvas on time. **Make sure to double check your submission. Correction requests after the deadline will not be accepted.** You may also utilize the late submission policy detailed above at your own discretion using the same submission link.

Important: If you are in a group, the ZIP file must be named using the format: `lastname1_firstname1_lastname2_firstname2.zip`. If you are not in a group, name the ZIP file using the format `lastname_firstname.zip`.

Penalties:

1. Multiple team members submitting separately: **(-5% points)**
2. Unnecessary files or folders not mentioned as supplements in the write-up: **(-5% points)**
3. Code fails to execute in the grading environment (provided docker environment) but works in your environment: **(-15% points)**
4. Submission in a format other than ZIP (.zip) or incorrect naming of .zip file: **(-10% points)**.
5. Missing name(s) on the submission: **(-5% points)** *(Yes, we've seen this before!)*
6. If your writeup consists mainly of code comments rather than detailed explanations of your thought process and approach, a **(-50% points)** penalty will be applied for that section. Include code only when absolutely necessary, and prioritize clear, thorough explanations.
7. A **(-10% points)** penalty will be applied if the writeup is written in an individual format instead of being presented as a cohesive team report (excluding individual submissions). Make sure the report reflects the team's collective effort.
8. If you submit your writeup in a format other than PDF, it will not be graded. No exceptions.
9. If you state in your writeup that you followed the instructions instead of explaining what you understood and what you have done, the relevant section will not be graded.

Bonus!:

1. You have the opportunity to assist your classmates in the HW discussion thread and earn up to 3% bonus for each helpful response, with a maximum bonus of 15%. To be eligible for bonus points, your responses must be non-anonymous. Only the first helpful reply to a specific question will earn a bonus for that question. By asking questions in the discussion section, you clarify your understanding and create opportunities for others to contribute and earn bonus points.
2. You will receive a 5% bonus if you prepare your writeup in LaTeX.

This course material is provided with modifications made specifically for Clemson University under a Creative Commons License. We thank the original authors from the University of Michigan for their support.