Project 3: Web Security

This project is due on **Thursday, April 7** at **11:59pm**. This is a group project: you can work in teams of two and can share the same answers. If you are having trouble forming a teammate, please post to the #partner-search channel on Slack.

Your answers here must be entirely your own work and that of your partner. You may discuss the problems with other students outside your group, but you may not directly copy answers, or parts of answers.

Introduction

A startup named BUNGLE! is about to launch its first product, a web search engine, but they are nervous about security problems. Unlike the Bunglers who developed the site, you took CSCI 3403, so they have hired you to perform a security evaluation before it goes live.

Your job is to attack their insecure website 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 these problems with various flawed defenses in place. Understanding how these attacks work will help you better defend your own web applications.

Security and Legal Concerns

This website is intentionally vulnerable! Do not enter any sensitive information or important passwords, as they may be accessible to other students or the broader internet. Additionally, please avoid intentionally altering or disabling the target website. Feel free to experiment as you see fit to complete the assignment, but do not run any attacks with the intention of deleting data or permanently damaging the lab.

Finally, this project asks you to develop attacks and test them, *with our permission*, against a website that we are providing for this purpose. Attempting the same kinds of attacks against other websites without authorization is prohibited by law and may result in fines and jail time. **You must not attack any website without authorization!**

Target Website

BUNGLE! is available for you to test at https://csci3403.com/proj3. Note that all URLs you will attack in this project will start with /proj3, and are only on the server https://csci3403.com. The site is written in Python using the Flask web framework.

In addition to providing search results, the site accepts logins and tracks users' search histories. It stores usernames, passwords, and search history in a SQLite database. The site has 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 URL parameter "q".

If no user is logged in, the main page also displays a form that gives the user the option of logging in or creating an account. The form issues POST requests to /login or /create.

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 searching 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.

Defense Levels

The Bunglers have been experimenting with some naïve defenses, so you also 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.

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.

Resources

The built-in web developer tools for Chrome and Firefox 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.

Your solutions will involve manipulating SQL statements and writing web code using HTML and JavaScript. You should search the web for answers to basic how-to questions. There are many fine online resources for learning these tools.

Although these purpose tools are permitted, you are not allowed to use tools that are designed to automatically test for vulnerabilities.

Part 1. SQL Injection

The login form uses this unsafe code to validate new user logins:

```
def validateUser(username, password):
    with connection() as conn:
        query = "SELECT username FROM users WHERE username='{}' AND
                 password='{}'".format(username, password)
        return conn.execute(query).fetchone()
```

For each of the following exercises, find a way of exploiting this query to achieve the stated goal. Your submission should be the text you entered which contains your SQL injection.

1.0 Login as Admin

Goal: Log in with the username “admin” without using their password.
Submission: sql_0.txt

1.1 Steal the Admin password

Goal: Submit a query that displays the password for the username “admin” somewhere on the webpage. You may find this guide to SQL UNION attacks helpful: https://portswigger.net/web-security/sql-injection/union-attacks.
Submission: sql_1.txt

Part 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 logged-in victim’s browser, correctly executes a malicious payload.

We recommend that you begin by testing with a simple payload (e.g. alert()), then move on to the full payload. Note that you should only need to implement the payload once, then use different means of encoding it to bypass the different defenses.

There are five levels of defense. You can select the level of defense used from the top of the website. The Python code that implements each one is shown below. 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. Your submission for each level of defense will be a text file with the specified filename that contains a single line consisting of a URL. When this URL is loaded in a victim’s browser, it should execute the specified payload against the specified target.

Payload

The payload will be to steal the cookies of the current user, and send them to a remote server. An attacker could use this kind of attack to steal session tokens, which can be used to log in as other users. The stolen cookies would normally be sent to another public web server, but for the purposes of grading we will send the cookies to a server on the local machine. The payload should read the document’s cookie string, and send it to the following URL: http://localhost:31337/stolen?cookies=[cookies]. On Canvas we have provided a sample Python server for this purpose called `test_server.py`. You can run it with `python3 test_server.py`.

2.0 No defenses

Defense: None
Submission: xss_0.txt

2.1 Remove “script”

Defense: filtered = re.sub("script", "", input, flags=re.IGNORECASE)
Submission: xss_1.txt

2.2 Remove several tags

Defense: filtered = re.sub("script|<img|<body|<style|<meta|<embed|<object", "", input, flags=re.IGNORECASE)
Submission: xss_2.txt

Payload

Finally, submit a human-readable version of your payload code (as opposed to the form encoded version in the URLs).
Submission: xss_payload.html

Part 3. Cross-site Request Forgery (CSRF)

Your final goal is to demonstrate CSRF vulnerabilities against the login and logout forms. You will need to construct an HTML file that, when opened by a victim, performs the specified goal. The appearance and behavior of the HTML file after it has been opened does not matter, as long as it accomplishes the specified goal. You will submit this HTML file. The HTML file can embed inline JavaScript and load libraries over the internet, but should be entirely self-contained in one file.

3.0 Log out

Goal: Log the victim out of their account.
Submission: csrf_0.html

Note: Naturally, this will have no effect on logged-out users.

3.1 Login as Attacker

Goal: Cause the victim to log in to an account you control, thus allowing you to monitor their search queries by viewing the search history for this account. This should log them into the account “attacker” and password “l33th4x”.
Submission: csrf_1.html

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.

Submission Instructions

Please upload the following answers to Canvas in a single .zip file. The name of the zip file should include your identikey and your partner’s identikey (if you have a partner). **Each member of your team should upload the .zip file to get graded**. Your submission should

contain the following files. **Make sure you follow the naming conventions described below**, since portions of the assignment will be autograded!

Part 1: SQL Injection

Submit a text files for each section with your username/password combinations in the following format. It is fine if one of the fields does not affect your attack.

```
username: <username>
password: <password>
```

1.0 Admin Login: **sql_0.txt**
1.1 Admin Password: **sql_1.txt**

Part 2: XSS

Submit a text file for each section containing a URL that, when loaded in a browser, immediately carries out the specified XSS attack. Additionally, submit the human-readable (non–URL-encoded) payload.

2.0 No defenses: **xss_0.txt**
2.1 Remove “script”: **xss_1.txt**
2.2 Remove several tags: **xss_2.txt**
Payload: **xss_payload.html**

Part 3: CSRF

Submit an HTML file for each section which, when loaded in a browser, immediately carries out the specified CSRF attack.

3.0 Log out: **csrf_0.html**
3.1 Login as Attacker: **csrf_1.html**