Assignment 4: Web Security

Acknowledgment: This project is based on an assignment created by Prof. Arvind Narayanan, Joseph Eichenhofer and their team at Princeton University for COS 432/ELE 432 course in Fall 2018.

This project is due on **Friday, November 1** at **11:59 p.m.** and counts for 10% of your course grade.

The code and other answers you will submit must be entirely your own work, and you are bound by the Honor Code. You may consult with other students about the conceptualization and the meaning of the questions, but you may not look at any part of someone else's solution or collaborate with anyone else. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions must be submitted via the submission server, following the submission checklist below.

Introduction

In this project, we provide an insecure website, and your job is to attack it by exploiting three common classes of vulnerabilities: cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL injection. 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.

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.

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!**

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 CS628A, so the investors have hired you to perform a security evaluation before it goes live.

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.

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`".

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` and `/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.

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.

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.

Part 1. SQL Injection

Your first goal is to demonstrate SQL injection attacks that log you in as an arbitrary user without knowing the password. In order to protect other students' accounts, we've made a series of separate versions of the bungle site for each student. For each of the following defenses, provide inputs to the target login form that successfully log you in as a victim:

1.0 No defenses

Target: 172.27.16.3:33414/c011a736/sqlinject0/index.html
Submission: `sql_0.txt`

1.1 Simple escaping

The server escapes single quotes (`'`) in the inputs by replacing them with two single quotes.

Target: 172.27.16.3:33414/c011a736/sqlinject1/index.html
Submission: `sql_1.txt`

1.2 Escaping and Hashing

The server uses the following PHP code, which escapes the username and applies the MD5 hash function to the password. You may need to write a program to produce a working exploit. You can use any language you like, but we recommend C.

You may find this useful, OpenSSL: https://rosettacode.org/wiki/MD5#C.

Target: 172.27.16.3:33414/c011a736/sqlinject2/index.html
Submissions: `sql_2.txt` and `sql_2-src.tar.gz`

1.3 The SQL [Extra credit]

This target uses a different database. Your job is to use SQL injection to retrieve:

(a) The name of the database
(b) The version of the SQL server
(c) All of the names of the tables in the database
(d) A secret string hidden in the database

Target: 172.27.16.3:33414/c011a736/sqlinject3/index.php
Submission: `sql_3.txt`

For this part, the text file should contain the four items above, and also a list of the URLs for all the form queries you made to learn the answers.

For this part, the text file you submit should starts with a list of the URLs for all the queries you made to learn the answers. Follow this with the values specified above, using this format:

URL
URL
URL
...

Name: *DB name*
Version: *DB version string*

```
Tables: comma separated names
Secret: secret string
```

What to submit For 1.0, 1.1, and 1.2, when you successfully log in as `victim`, the server will provide a URL-encoded version of your form inputs. Submit a text file with the specified filename containing only this line. For 1.2, also submit the source code for the program you wrote, as a gzipped tar file (`sql_2-src.tar.gz`). For 1.3, submit a text file as specified.

Part 2. Cross-site Request Forgery (CSRF)

Your next task is to demonstrate CSRF vulnerabilities against the login form, and **Bungle!** has provided two variations of their implementation for you to test. 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 and host it on some server, such that when the link is opened by a victim, it logs their browser into **Bungle!** under the account "`attacker`" and password "`97cedde9ff6c166b0bbc6100b24c810c`".

Your solutions should not display evidence of an attack; the browser should just display a blank page. (If the victim later visits **Bungle!**, it will say "logged in as attacker", but that's fine for purposes of the project. After all, most users won't immediately notice.)

2.0 No defenses
Target: `172.27.16.3:33414/c011a736/login?csrfdefense=0&xssdefense=4`
Submission: `csrf_0.html`

2.1 Token validation
Target: `172.27.16.3:33414/c011a736/login?csrfdefense=1&xssdefense=0`
Submission: `csrf_1.html`

What to submit 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 http://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js. It is also permissible to load the jQuery cookies library from https://cdnjs.cloudflare.com/ajax/libs/jquery-cookie/1.4.1/jquery.cookie.min.js. Otherwise, your code must be self contained.

Part 3. Cross-site Scripting (XSS)

Your final goal 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.

Payload

The payload (the code that the attack tries to execute) will be to steal the username and the most recent search the real user has performed on the **BUNGLE!** site. Your URL must perform a google search with the query username_lastsearch. Where username is of the victim user and last search is his last searched query.

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. Try to use a different technique for each defense.

3.0 No defenses
Target: `172.27.16.3:33414/c011a736/search?xssdefense=0`
Submission: `xss_0.txt`

For 3.0 only, also submit a human-readable version of your payload code (as opposed to the form encoded into the URL). Save it in a file named `xss_payload.html`.

3.1 Remove "script"

Target: `172.27.16.3:33414/c011a736/search?xssdefense=1`
Submission: `xss_1.txt`

3.2 Remove several tags

Target: `172.27.16.3:33414/c011a736/search?xssdefense=2`
Submission: `xss_2.txt`

3.3 Remove some punctuation

Target: `172.27.16.3:33414/c011a736/search?xssdefense=3`
Submission: `xss_3.txt`

What to submit 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. The payload encoded in your URLs may embed inline JavaScript and load jQuery from the URL http://ajax.googleapis.com/ajax/libs/jquery/2.1.4/jquery.min.js. It is also permissible to load the jQuery cookies library from https://cdnjs.cloudflare.com/ajax/libs/jquery-cookie/1.4.1/jquery.cookie.min.js. Otherwise, your code must be self contained.

Part 4. Writeup: Better Defenses

For each of the three kinds of attacks (SQL injection, CSRF, and XSS), write a paragraph of advice for the **BUNGLE!** developers about what techniques they should use to defend themselves. Place these paragraphs in a file entitled "`writeup.txt`".

What to submit A text file named `writeup.txt` containing your security recommendations.

Submission Checklist

Upload to the submission portal a zipped file.

Part 1: SQL Injection

For parts 1.0, 1.1, and 1.2, submit text files that contain the strings provided by the server when your exploits work. For 1.2, also submit a tarball containing the source code you wrote to produce your solution. For 1.3, submit a text file as specified in the problem statement. Make sure your text files are actual .txt files or we may not be able to grade your work.

`sql_0.txt`	1.0 No defenses
`sql_1.txt`	1.1 Simple escaping
`sql_2.txt`	1.2 Escaping and Hashing
`sql_2-src.tar.gz`	1.2 Escaping and Hashing
`sql_3.txt`⋆	1.3 The SQL [Extra credit]

Part 2: CSRF

HTML files that, when loaded in a browser, immediately carry out the specified CSRF attack. Please remember to correctly specify the CSRF defense level in your URLs.

`csrf_0.html`	2.0 No defenses
`csrf_1.html`	2.1 Token validation

Part 3: XSS

Text files, each containing a URL that, when loaded in a browser, immediately carries out the specified XSS attack. For 3.0, also submit the human-readable (non–URL-encoded) payload. Please remember to correctly specify the XSS defense level in your URLs. (If you have nested URLs, specify the defense levels in the nested ones too!)

`xss_payload.html`	3.0 No defenses
`xss_0.txt`	3.0 No defenses
`xss_1.txt`	3.1 Remove “script”
`xss_2.txt`	3.2 Remove several tags
`xss_3.txt`	3.3 Remove some punctuation

Part 4: Writeup

One text file named `writeup.txt` containing an answer to the questions in part 4.

⋆ These files are optional extra credit.