CS 473: Network Security
Assignment 3
Spring 2022

Topics: XSS, CSRF, SQL Injection Attacks

Release Date: Thursday 24th March 2022

Total Marks: 165

Submission Date and Guidelines:

This assignment has 4 sections related to SQL injection, CSRF, XSS and clickjacking attacks. All the following tasks must be completed by **Tuesday 5th April 2022, 11:55 pm**. You will have to submit a **SINGLE** zipped folder on LMS, containing a pdf with screenshots of the webpage after the attack as proof, the attack input string, and a brief description about how you carried out the attack, for each task specified. The zipped folder should also contain a modified clickjacker.html file for Task 4. Submit your file with the naming convention rollnumber_Assignment3.zip.

This is an **Individual** assignment. The code and other answers you submit **MUST** be entirely your own work, and you are bound by the Student Code. 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 electronically on LMS within the due date.

Pre-requisites:
For this homework, you need to have an **Ubuntu Machine or Ubuntu VM** (You may use the one you used in the previous assignment).
Please make sure you have **Sqlite3** (it is by default installed on Ubuntu machines - simply enter sqlite3 on your terminal to check). If it is not installed, use the command:

```
sudo apt install sqlite3
```

You also need **Python (any 2.x version)** installed as well. You can check your version using the command: `python -version` to check.
Lastly, use **Google Chrome**, for the purpose of this homework. You can install it on your VM.

Resources

Resources The Chrome Web Developer tools will be a tremendous help for this project, especially the JavaScript console and debugger, DOM inspector, and network monitor. You can access it using (Ctrl +Shift + I) on Windows.

Although general-purpose tools are permitted, you MUST NOT use tools that are designed to automatically test for vulnerabilities. Your solutions will involve manipulating SQL statements and writing web code using HTML, JavaScript.

There are many fine online resources for learning these tools. Here are a few that we recommend:
SQL Tutorial: SQL Tutorial
SQL Statement Syntax: http://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html
Introduction to HTML: Introduction to HTML - Learn web development | MDN
HTTP Made Really Easy: HTTP Made Really Easy

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

- SQL Injection Prevention - OWASP Cheat Sheet Series
- Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series
- https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_She
- XSS Filter Evasion - OWASP Cheat Sheet Series
- SQL Injection | OWASP Foundation

For those new to HTML and CSS, the following resources will be helpful:
HTML: https://www.w3schools.com/html/
CSS: https://www.w3schools.com/css/

Task 1: Cross-Site Scripting (XSS) (60)

Complete the whole https://xss-game.appspot.com/ XSS game. There are a total of six short levels. Complete all of them. After completion of each level, you have to document the following:

1. A screenshot of the 'Level completion' page.
2. A brief description of your thought process and how you carried out the attack, providing all inputs.

Note: If no evidence of working is provided, marks will not be given.

For Tasks 2 and 3, you'll have to:

1. Open the terminal in your Ubuntu Machine and run the web server by typing in the command: `python webserver.py`
2. Open your Chrome browser, interact with the Amagon website by typing **http://localhost:8080/** **or** **http://127.0.0.1:8080/** in the URL bar.
3. You will then see the Amagon.com website.

Login Credentials for Tasks 2 and 3:

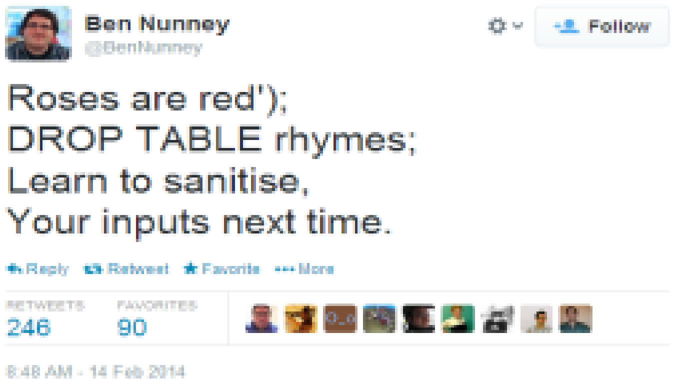
You can choose any of the following,

1) Username**:** nawaz, Password: niHari

2) Username: imran , Password: ghabrana_nahi

Task 2: SQL Injection Attacks

SQL injection vulnerabilities allow attackers to inject arbitrary scripts into SQL queries. When an SQL query is executed, it can either read or write data, so an attacker can use SQL injection to read your entire database as well as overwrite it.



Ben Nunney
@BenNunney

Roses are red');
DROP TABLE rhymes;
Learn to sanitise,
Your inputs next time.

8:48 AM - 14 Feb 2014

For the purpose of the following tasks you will have to construct simple SQL queries, if needed, and no complex SQL queries are needed for SQL Injection. You can use https://www.w3schools.com/sql/ for refreshing your MySQL basics.

Using "**Group_concat()**" function in the select clause, "**–**" comment symbol and "**| |**" symbol for string concatenation will be helpful for your task. All SQL queries are concatenated as strings, so when carrying out the attack, keep an eye out for ending and starting string quotes.

Note: Semicolons are used to end SQL statements.

Feel free to observe and manually analyze the webserver code provided to you which contains the SQL queries, to think of the malicious SQL Injection input strings.

Start the **Amagon** web server using: **python webserver.py**

Now, open your chrome browser, interact with the **Amagon** website by typing **http://localhost:8080/** in the URL bar. **During your task attempts, if the server crashes, just go to the home page or click the back button to attempt again.**

We think you are good to go then, let's begin…

Part 1: Change the Password of an Amagon User (10)

For this task, you need to login in as **sana**, who is also an active member of **Amagon.com**. The catch is that although you do not know their password, you've figured out that the login page is vulnerable to SQL injection :)

Your task is to go to the login page (**http://localhost:8080/login**) and enter a malicious SQL string in the context of the program such that it allows you to update sana's password to your liking, and then you can login to sana's account with your own specified password.

The vulnerable SQL query in the program is:

"SELECT password from accounts where username='%s'" % user

[The above query is executed when a user enters their credentials and clicks the login button. Using the above query, the program extracts the user's password and then later checks if the input password matches the password in the database (line of code: 92)]

Tip: You will have to use the 'UPDATE' SQL query to update sana's password. it will be part of your exploit string into the username input box:

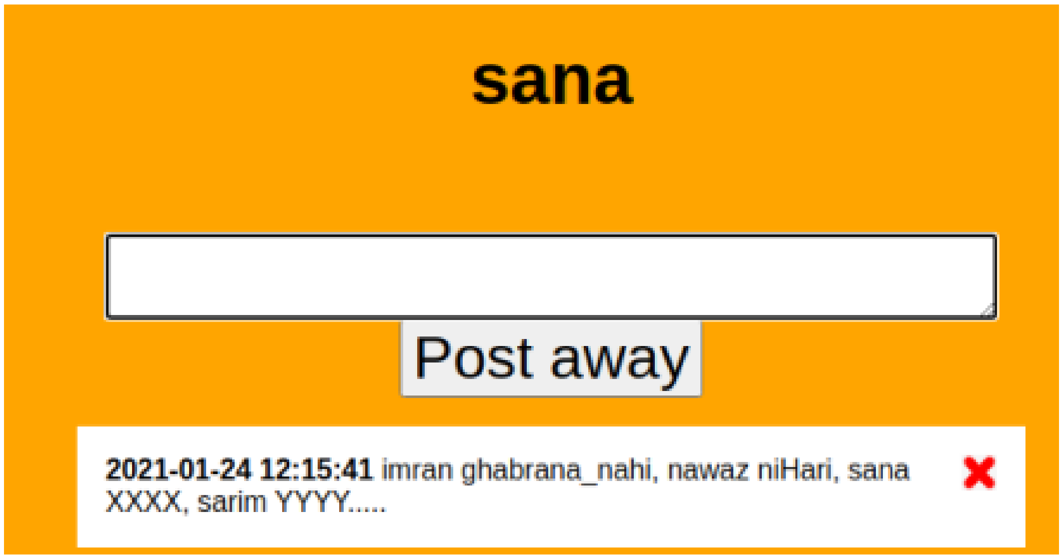
How to craft the input is up to you. You have to:

1. Document the malicious input string you used.

2. Paste a screenshot of you logged in as “sana”.
3. Briefly explain why you observed the behavior and why the exploit string you used, worked.

Part 2: Password Dump via Amagon’s Post (15)

For this task, you have to carry out SQL Injection through the Amagon post input box. Craft a user input such that when you press the ‘Post Away’ button and the page refreshes, all usernames and their corresponding passwords are dumped out as a post on the web application. Such as:



2021-01-24 12:15:41 imran ghabrana_nahi, nawaz niHari, sana XXXX, sarim YYYY.....

The SQL queries that are vulnerable for this task and can be exploited are:

(1) "INSERT INTO posts VALUES ('%s', '%s', datetime('now', 'localtime'));" % (user, post)
[When posting on Amagon, the above SQL query stores/inserts that post for a particular user, with the datetime of the post made, into the database. (line of code: 124)]

(2) "SELECT body, time from posts where username='%s' order by time desc limit 10" % (user)
[After posting, when the page refreshes the above, SQL query is made to extract out all the 10 newest posts (ordered by time) and display them on the page. (line of code: 115)]

Tip: Try to enter a SQL string, such that it becomes part of (1). Instead of your post, something else from the database gets stored in ‘posts’ Table. When (2) is called to display the posts, all passwords and corresponding usernames are displayed as the most recent post. You have to exploit (1), (2) will work naturally.

You have to:
1. Document the malicious input string you used.
2. Paste a screenshot of the password and username SQL dump as a post.
3. Briefly explain why you observed the behaviour and how the attack worked.

Task 3: Cross-Site Request Forgery (CSRF)

Amagon.com is a web application that allows you to post amazing stuff and also delete them. All posts are public so you can click on ‘Browse Users’ to see other users and then navigate to their posts.
The website is vulnerable to CSRF attacks, and you being a malicious person want to exploit it and cause trouble. You know that when you post, the web application stores your post in the SQL database and refreshes the page, which causes all stored posts in the database to be shown on the screen and become part of the HTML.

The terminal window that you used to start the webserver, will have logs of all URL redirects and GET requests that are made from your interaction with the Amagon website. Use that to your advantage as well, using it as a simplified local sniffer (packet capturer).

For the purpose of the following tasks, you will have to create hyperlinks using anchor tags. Use the following resource if you don't know how to: https://www.w3schools.com/html/html_links.asp

The goal here is to find a way to perform account-changing actions on behalf of a logged-in Amagon user without their knowledge.

Note: All the following attacks need to be performed through the Amagon post user input box (login and click on ‘My Posts’) page, unless otherwise specified.

Part 1: Log them out (10)

Post a socially engineered post, such as:

“There’s a concert next week! Find latest updates here: www.concerts.com“

So that whenever a user clicks on the specified link, he/she is logged out of their account instead.

Use nawaz’s account to carry out the attack, and imran’s account as a victim for the attack illustration, by logging in as imran, going to nawaz’s profile using the “Browse Users” tab and opening the link which would cause imran to get logged out of his account.

Part 2: Woah, did I actually post that? (15)

Use the same socially engineered post as before, craft your malicious post input in such a way that whenever a user clicks on the link, they automatically share a post stating “COVID isn’t over yet”.

Use the nawaz’s account to carry out the attack, and imran’s account for the attack illustration by going to nawaz’s profile using the “Browse Users” tab and opening the link. imran should have a malicious post posted on his timeline due to clicking on the link in nawaz’s post.

Part 3: Extended CSRF (20)

In the previous task, you generated exploit hyperlinks which resulted in account log out and malicious posts when a victim clicked on the socially engineered link.

Now, you have to carry out a similar attack but instead, whenever a victim clicks on a link in someone else’s post, they see an alert box that contains their cookie (session_id), **every time their post’s page is refreshed**.
You can get the value of the cookie using: document.cookie.
Think along the lines of what you did in the previous part and Reflected XSS attacks.
Can CSRF and reflected XSS be merged together in some way?

You need the following to show up for any user that clicks on your malicious hyperlink, every time their page refreshes: (your cookie value can be different)

As an example for clarification, nawaz posts on his account that includes a link. Imran then view’s nawaz’s post and clicks on the link. Then, when he goes to his own posts page, he is alerted with his session id. Imran then posts something and upon clicking the “post away” button, imran is alerted again.

You have to:
1. Document a screenshot of the above attack, as illustrated.
2. Write your exploit input string.
3. Briefly explain why you observed such a behavior and how this attack works.

Note: Failure to provide the required evidence and missing details will result in a loss of marks.

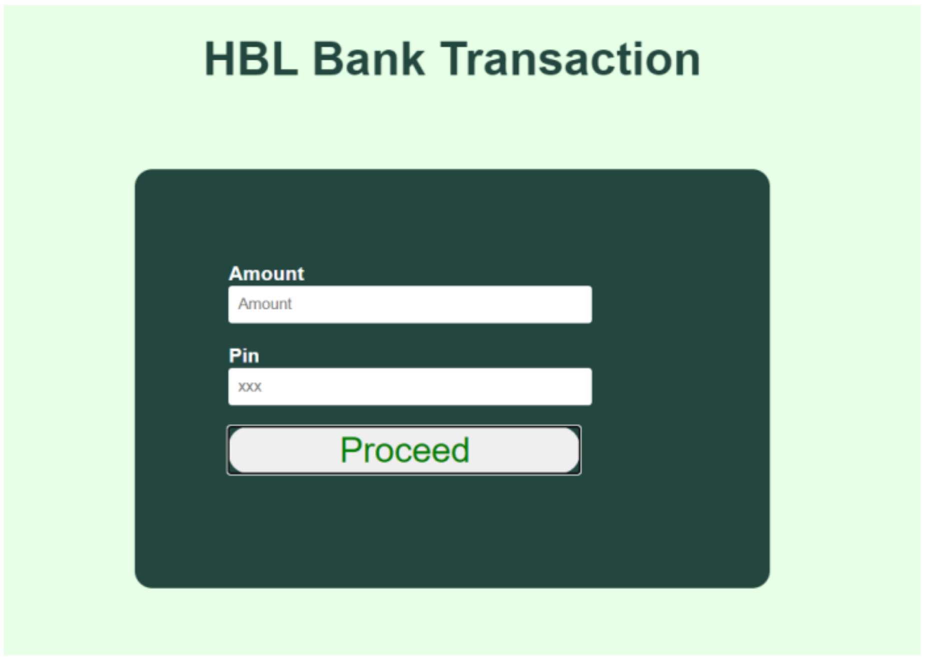
Task 4: Clickjacking (15)

Part 1 (10)

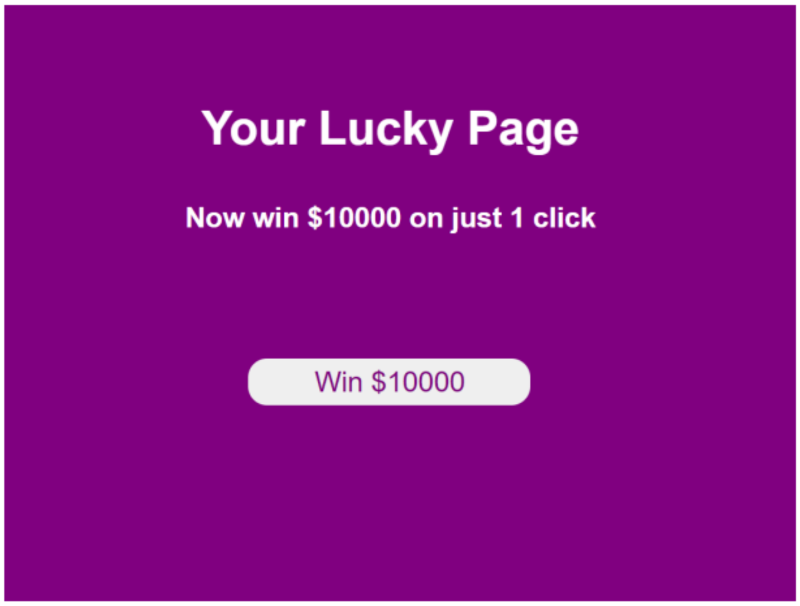
Clickjacking is an attack that misleads the victim by overlaying multiple frames and making some frames invisible. Thus the victim is displayed with one webpage but his/her action is actually on another webpage that is selected by the attackers. This attack takes advantage of the HTML property called iframe. The objective of this question is to understand how iframe with some Style property can be used to build such an attack.

You are provided with two HTML files: the dummy_hbl.html and clickjacker.html files. To study and play around with the code, open these files in your favourite text editor. To view the website page, open the HTML file on your Chrome browser. You should see the following:

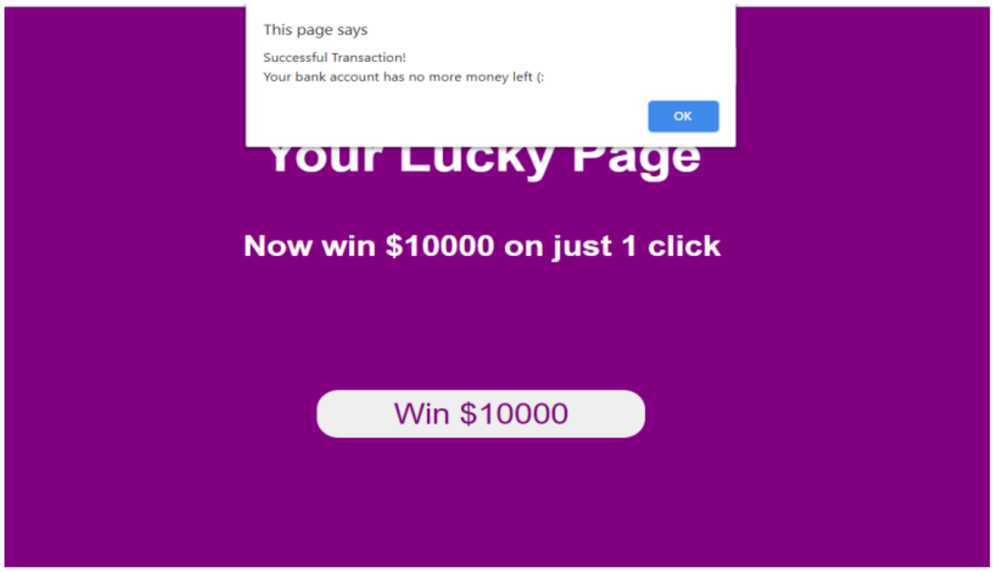
dummy_hbl.html



clickjacker.html



Your task is to overlay an iframe on HBL's transaction page and then change its styling in a way that prompts the user to click in the hope of winning a lottery. Hence, when the user clicks the 'win $10000' button, they will not be led to winning $10000 as they wanted to be but instead, a transaction would take place from their account to yours via the hidden HBL page. You should be able to see an alert message in case of a successful attack.


This page says

Successful Transaction!
Your bank account has no more money left (:

You will complete the skeleton code given to you in the clickjacker.html on text editor, and submit this file on LMS. **You are only allowed to make changes in the specified slots highlighted in the code in clickjacker.html file.** You do not need to modify any of the existing code. You are free to use the other file for reference when styling your iframe. You will also be required to share the reasoning for why you filled in what you filled.

NOTE: For the purposes of this homework we will assume dummy_hbl.html is the actual HBL page hosted on the HBL website. (So that we don't directly attack the HBL page, but a simulated version of it). Normally you would include the HBL page as <iframe src=hbl.com>, but to include the dummy page you can include the provided dummy_hbl.html file in the iframe tag.