# CS473: Network Security

# Homework 1 - Spring 2024

**Lead TAs:** Seemal Tausif and Zoha Hayat Bhatti
**Due Date:** 12th February 2024
**Total Marks:** 140

## Submission Guidelines:

You will have to submit a SINGLE zipped folder on LMS, containing a pdf with screenshots of the webpage after the attack as a 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 downloadPDF.html file for Task 1. Submit your file with the naming convention rollnumber_HW1.zip.

## Installation Guidelines:

For this homework, you need to have an Ubuntu Machine or Ubuntu VM. Please make sure you have Sqlite3 (it is by default installed on Ubuntu machines - simply enter sqlite3 on your terminal to check) and Python (any 2.x version) installed as well. Lastly, use Google Chrome, for the purpose of this homework (although Firefox works too). This setup will be needed in tasks 3 and 4.
For Ubuntu setup these links might be helpful:

1) https://www.youtube.com/watch?v=v1JVqd8M3Yc
2) https://www.geeksforgeeks.org/how-to-install-ubuntu-on-virtualbox/

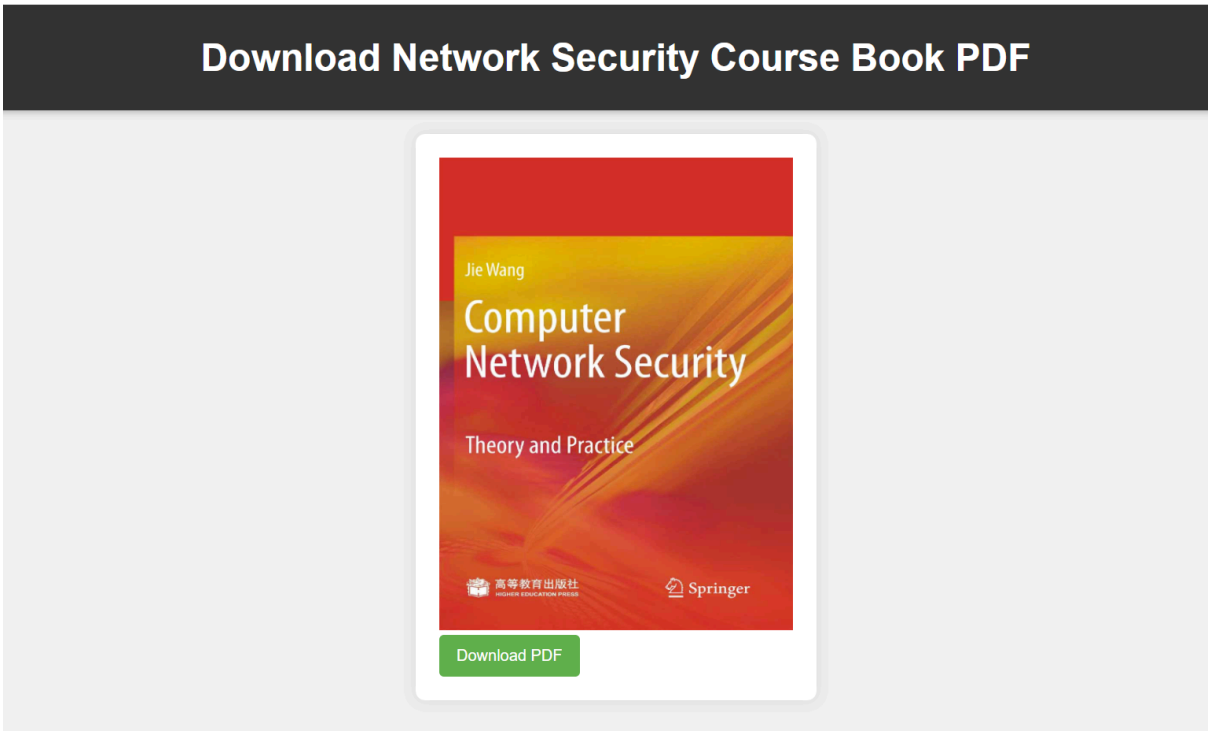
## Task 1: Clickjacking (15 marks)

### Part 1A: (10 marks)

Clickjacking is a deceptive attack technique that involves overlaying frames on a webpage, leading users to unknowingly interact with a different page than they perceive. 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: downloadPDF.html and malicious.html files. Open these files to study and play around with the code. To view the actual website

page, open the HTML file on your Chrome browser. You should see the following:


**malicious.html**




**downloadPDF.html**

You will complete the skeleton code given to you in the downloadPDF.html file and submit this file on LMS. You are only allowed to make changes to the specified slots highlighted in the code in the downloadPDF.html file. You do not need to modify

any existing code. You will also need to explain the reasoning for any changes you make to the file.

Some relevant resources:

1) HTML: https://www.w3schools.com/html/
2) CSS: https://www.w3schools.com/css/
3) Clickjacking: https://portswigger.net/web-security/clickjacking

**Part 1B: (5 marks)**

Describe how the Frame Busting technique is used to prevent clickjacking. What mechanisms can be employed for effective frame busting?

**Task 2: Scenario Based Questions (10 marks)**

Apply your understanding of web security concepts, specifically focusing on Cross-Site Request Forgery (CSRF) and SQL Injection, to solve the problems below.

1) You have discovered a web application that utilises both cookies and CSRF tokens for session management. The server only accepts HTTP requests containing both a valid HTTP cookie header and a valid CSRF token in the POST parameters. However, you suspect there might be a way to bypass this defence. Devise a method to bypass the CSRF token protection and execute a successful CSRF attack on this web application. Provide a step-by-step description of your approach.
2) In the login functionality of a web application, user input is utilised in constructing SQL queries without proper sanitization, potentially exposing the system to SQL Injection attacks. The login mechanism is implemented as follows:

```
username = user_input('Enter your username: ')
password = user_input('Enter your password: ')
query = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password + "';"
```

Your objective is to identify and exploit the SQL Injection vulnerability in the login system to retrieve sensitive information, such as user credentials. Develop and execute a SQL Injection attack by providing the injected SQL statement along with a detailed explanation of your strategy.

**Task 3: SQL Injection Attacks (25 marks)**

SQL injection vulnerabilities expose a web application to the injection of malicious scripts into SQL queries. This enables attackers to manipulate the queries, potentially allowing unauthorised access to or modification of the database. For the tasks ahead, you will construct simple SELECT queries (e.g., Select X From Y). No advanced SQL queries are required; you can refer to W3Schools' MySQL Basics for a quick refresher.

In your exploration, consider using the GROUP_CONCAT() function in the SELECT clause, -- as the comment out symbol, and || for string concatenation; these will prove helpful for the tasks. Keep in mind that all SQL queries are treated as strings, so when crafting an attack, pay attention to the starting and ending string quotes. SQL statements are terminated with semicolons.

You are encouraged to inspect and manually analyse the webserver code provided, which includes the SQL queries. This will aid in devising malicious SQL injection input strings.

Follow the steps to below to access the Fazebook website for both tasks 3 and 4:

1) Open the terminal in your Ubuntu Machine and run the web server by typing in the command: python2 webserver.py.
2) Open your Chrome browser, interact with the Fazebook website by typing http://localhost:8080/ in the URL bar.
3) You will see the Fazebook.com website.
4) Login Credentials for Tasks 3 and 4:
   You can choose any of the following:
   a) Username: nawaz, Password: niHari
   b) Username: imran , Password: ghabrana_nahi

If the server crashes during your task attempts, simply navigate to the home page or click the back button to retry. If your homepage looks like this, you're good to go:


Some relevant resources:

1) https://portswigger.net/web-security/sql-injection
2) https://www.guru99.com/learn-sql-injection-with-practical-example.html

**Part 3A: Unauthorised Password Change (10 marks)**

In this task, you'll exploit a SQL injection vulnerability to change the password of an active Fazebook.com user named Sana. Even though you do not possess Sana's password, you've discovered a vulnerability in the login page that allows for SQL injection. Your objective is to manipulate the SQL query executed during login, enabling you to update Sana's password to a desired value. Once successful, log in to Sana's account using your specific password.

The instructions are as follows:

1) Exploit Crafting:
    a) Navigate to the Fazebook login page (http://localhost:8080/login).
    b) Inject a malicious SQL string into the username input box or URL parameter to modify Sana's password to "#Ipwndyou."
    c) Document the crafted input string, explaining how you formulated it to achieve the intended exploit.
2) Screenshot:

   a) Capture and paste a screenshot demonstrating that you have successfully logged in as "Sana" using your specific password.
3) Behaviour Observation
   a) Briefly explain the observed behaviour during and after the exploit.

Tips:

1) The vulnerable SQL query in the program is:
   SELECT password from accounts where username='%s"' % user
   (This 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))
2) The provided SQL query to update Sana's password to "#Ipwndyou":
   UPDATE accounts SET password='#Ipwndyou' WHERE username='sana';
   (Include this in your exploit string into the username input box/URL parameter.)

**Part 3B: SQL Injection for Password Dump (15 marks)**

In this task, you will leverage a SQL injection vulnerability within Fazebook's post input box to execute an attack that results in the dumping of all usernames and their corresponding passwords as a post on the web application. Your goal is to craft a user input that, when the 'Post Away' button is pressed, manipulates the SQL queries used for storing and retrieving posts. As a result, the web application will inadvertently display a post containing sensitive information.

The instructions are as follows:

1) Exploit Crafting:
   a) Navigate to the Fazebook post input box.
   b) Craft a SQL string input that becomes part of the "INSERT INTO posts" query, ensuring it alters the stored content in the 'posts' table to include usernames and corresponding passwords.
   c) Document the malicious input string, explaining the methodology behind crafting it for successful exploitation.
2) Screenshot:
   a) Capture and paste a screenshot demonstrating the successful SQL dump of usernames and corresponding passwords as a post on the web application.
3) Behaviour Observation:

a) Briefly explain the observed behaviour during and after the exploit, emphasising why this result occurred.

Tips:

1) The vulnerable SQL queries in the program are:
    a) INSERT INTO posts VALUES ('%s', '%s', datetime('now', 'localtime'));" % (user, post)
    (When posting on Fazebook, 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))
    b) 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))
2) When attempting to exploit (1), consider crafting a SQL string that seamlessly integrates into the "INSERT INTO posts" query. Instead of a conventional post, manipulate the input so that something else from the database gets stored in the 'posts' table. As a result, when (2) is invoked to display the posts, you will successfully execute the exploit, revealing all passwords and their corresponding usernames as the most recent post.

**Task 4: Cross-Site Request Forgery (65 marks)**

Fazebook.com, an interactive web platform for sharing and deleting posts, provides users with the ability to browse and view each other's private posts. However, the website harbours a vulnerability susceptible to CSRF attacks. As a mischievous individual aiming to exploit this weakness, your objective is to disrupt the normal functionality of the website and cause unintended actions.

Here's how the exploitation unfolds: when a user makes a post, the web application stores it in the SQL database and refreshes the page, incorporating all stored posts into the HTML, visible to all users. Leveraging this behaviour, you intend to manipulate the website's functionalities by discreetly executing actions on behalf of logged-in users.

To assist you in your endeavours, the terminal window used to launch the web server logs all URL redirects and GET requests stemming from your interactions with the Fazebook website. Treat this log as a simplified local sniffer, capturing the essence of your web traffic.

For these tasks, you will harness the power of anchor tags to create hyperlinks that facilitate CSRF attacks. If you're unfamiliar with anchor tags, refer to this resource for guidance: HTML Links at W3Schools.
Your ultimate goal is to discover a method to execute account-changing actions on behalf of logged-in Fazebook users without their awareness. All the ensuing attacks must originate from the Fazebook post user input box, accessible through the 'My Posts' page after logging in.

Some relevant resources:

1) https://portswigger.net/web-security/csrf
2) https://kinsta.com/blog/csrf-attack/

**Part 4A: Logout Manipulation (10 marks)**

In this task, your objective is to incorporate a malicious link within a post on Fazebook.com that, when clicked by another user, initiates an unauthorised logout from their account. This deceptive post should entice users with a misleading headline, encouraging them to click on a provided link leading to unexpected consequences.

The instructions are as follows:

1) Craft a Deceptive Squig:
    a) From Nawaz's account, compose a post featuring a socially engineered squig.
    b) The squig's content should be enticing, prompting users to click on a link for purported updates about LUMS trending on Twitter (e.g., "LUMS trending on Twitter again! Find latest updates here: www.masalanews.com").
    c) Ensure the link is designed to log users out when clicked.
    d) Document the crafted input string.
2) Attack Illustration:
    a) Log in as Imran and navigate to Nawaz's profile using the "Browse Users" tab.
    b) Open the deceptive link within the squig post to demonstrate the attack on Imran's account.
    c) Capture and paste a screenshot illustrating the logout event on Imran's account.
3) Behaviour Explanation:

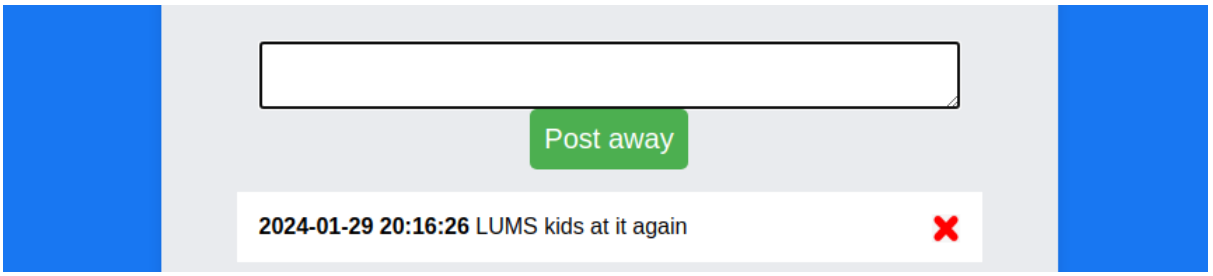
a) Provide a brief explanation of the observed behaviour, detailing how the socially engineered squig successfully triggered the logout action.

### Part 4B: Covert Auto-Share Manipulation (15 marks)

In this task, your objective is to extend the impact of your socially engineered post on Fazebook.com. Utilising the same deceptive post as in Part 1, craft a malicious post input that, when clicked by another user, automatically triggers the sharing of a post with the content "LUMS kids at it again."

The instructions are as follows:

1) Craft a Covert Auto-Share Squig:
    a) Using Nawaz's account, modify the socially engineered squig post to include a hidden mechanism.
    b) Design the post input in a way that, upon clicking the link, users unwittingly share a post on their timeline with the message "LUMS kids at it again."
    c) Document the crafter input string.
2) Attack Illustration:
    a) Log in as Imran and access Nawaz's profile via the "Browse Users" tab.
    b) Click on the link within the modified squig post to demonstrate the attack on Imran's account. The malicious post on Imran's account will look like this:




    c) Capture and paste a screenshot showcasing the auto-shared post on Imran's timeline.
3) Behaviour Explanation:
    a) Provide a concise explanation of the observed behaviour, emphasising how the covert auto-share mechanism successfully executed the "LUMS kids at it again" post on Imran's timeline.

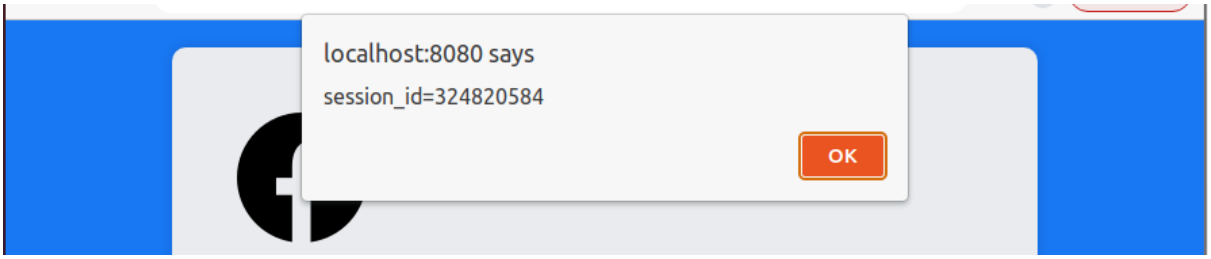
### Part 4C: Advanced CSRF with Cookie Extraction (20 marks)

Building upon the previous tasks, your objective is to conduct an advanced CSRF attack that combines aspects of Cross-Site Request Forgery (CSRF) with Reflected

Cross-Site Scripting (XSS). When a victim clicks on your engineered link, an alert box should appear containing their cookie (session_id) every time the post page is refreshed. You should use Nawaz's account to make the malicious post and Imran's account to demonstrate the alert box appearing on every page refresh.

The instructions are as follows:

1) Execute Advanced CSRF Attack:
    a) Craft an exploit hyperlink that, when clicked, triggers a CSRF attack with a reflected XSS element.
    b) The goal is to display an alert box containing the victim's cookie (session_id) upon each refresh of their post page. The alert box will look like this (the session id may be different):




2) Document the Attack:
    a) Capture and document a screenshot illustrating the successful execution of the attack.
3) Exploit Input String:
    a) Provide the exploit input string you used to achieve the desired behaviour.
4) Behaviour Explanation:
    a) Briefly explain the observed behaviour, outlining how the combination of CSRF and reflected XSS elements led to the display of an alert box containing the victim's cookie upon page refresh.

**Task 5: Theoretical Questions (25 marks)**

1) In the context of web security, what is Cross-Site Scripting (XSS) and how does it differ from Cross-Site Request Forgery (CSRF)? Provide examples to illustrate each.
2) Explain the difference between Stored XSS and Reflected XSS attacks. Provide real-world scenarios for each, and discuss the potential impact of these attacks on users and web applications.
3) How does the Same-Origin Policy (SOP) impact the effectiveness of Cross-Site Scripting (XSS) attacks? Explain why XSS attacks can still occur despite the existence of the SOP.

4) Describe the potential consequences of a successful SQL Injection attack on a web application. Provide examples of scenarios where SQL Injection could compromise sensitive information or lead to unauthorised access.
5) Explain the concept of Click-Jacking in web security. How do attackers use invisible frames to mislead users, and what are the potential risks associated with Click-Jacking attacks?