Application Security Workshop - Preparation for participants:

Dear Students,

In order to participate in our Application Security Workshop you need to

- install the necessary software and my demo application “HackingExamples” on your computer.
- prepare for some application security related topics you need to understand in advance. Please prepare for these topics by fullfilling the tasks you find in this document.

Software Installation:

Please install this software on your Windows-PC:

- Visual Studio 2022 (the free Community Version is enough) and make sure that you have installed this workload:

Allthough this should be enough to run my example application we are going to use, I would recommend that you also install some addional options, that will also enable you to start new ASP.Net WebForm Applications. Find a Video that explains in detail what you need here: https://www.youtube.com/watch?v=MspiJWCfUXE

- At least one decent browser, I use and recommend Google Chrome
- Install my demo application (HackingExamples) by decompressing the ZIP in a folder and by opening the application by doubleclicking HackingExamples.sln (do not simple open the folder in VS, you need to open the Solution!). If it runs, you’re done.

Workshop preparation

Before we dive into application security in our upcoming course, it’s crucial that you have a solid understanding of some fundamental technologies. Security vulnerabilities such as **SQL Injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF)** exploit weaknesses in how web, mobile (hybrid) and even some desktop applications handle data, user input, and authentication.
To fully grasp these attacks and how to prevent them, you need to be comfortable with the following concepts:

1. **HTML Forms & HTTP Basics** – Understanding how data is sent between the browser and server is essential, as many attacks manipulate form submissions.

2. **A little bit of JavaScript and changing HTML elements using the DOM** – understanding this will help you to understand and perform advanced XSS-Attacks.
3. **SQL and basic CRUD statements –** understanding the syntax of SQL is necessary to fully understand and perfom SQL-Injection attacks.
4. **C# ans .net ASPX-**Pages – understanding the basic syntax of these technologies is important as our "Hacking Examples" are implemented using them. You will have to understand the code and fix the problems to avoid hacking attacks. You do not have to create your own applications using the ASP.Net WebForm Framework.
5. **Basic Encryption & Hashing** – Secure password storage is a critical defense against data breaches. You should understand why hashing is used instead of encryption.
6. **Same-Origin Policy (SOP) & CORS** – Web security mechanism that protect users from unauthorized data access across different domains. Attackers often try to bypass these protections. It is also important to understand XSRF-attacks.

The following tasks will help refresh your knowledge in these areas. Please complete them before the lecture to ensure you're ready for the security topics we'll cover. There is no need to submit your results, but we will talk about these topics in the course. I have also added some discussion points. Please investigate the points yourself or discuss them with your colleagues.

If you have any questions while working on the tasks, feel free to contact me at: grischa.schmiedl@fhstp.ac.at

Looking forward to our course!

FH-Prof. DI Dr. Grischa Schmiedl

***

HTML Forms & HTTP Basics Task

Objective: Understand how HTML forms work and how data is sent via GET and POST requests.

Task:

1. Create an HTML form with a text field for a username and a submit button.
2. Create two versions: one that submits via GET and one via POST.
3. Observe the difference in how data appears in the URL.
   For this task you do not have to create a page that receives the values of the form.

Discussion Points:

- What happens to the URL when using GET vs. POST?
- Why would we use one over the other?
- What are the security implications of using GET for sensitive data?

JavaScript Task

Objective: Understand and modify the DOM (document object model) dynamically.

Write a simple HTML-File including a JavaScript function that:

- Adds a new paragraph (<p>) inside an existing when a button is clicked.
- Changes the text color when hovering over the paragraph.
- Uses "innerHTML" to overide the complete BODY of the document, so that a complete new content is shown without reloading data from the server.
- Use the "inspect"-function of your browser (in Chrome available on right-click on any part of the document – last option) and monitor the changes to the DOM.

Discussion Points:

- What is the difference between the source of a document (the HTML that was sent to the browser) and the DOM?
- Add some nonsense code to your file like a non valid HTML tag <thisisnonsense>. Why is is not shown in the DOM.

SQL Task

Objective: Write basic CRUD operations. Use any SQL database you like, like mySQL/MariaDB or MS SQL Server/localdb. In my examples we will use localdb which is included in MS Visual Studio 2022.

Tasks:

1. Create a table Users with columns ID, Username, and Password (store as plaintext for now to discuss security implications later).
2. Write SQL statements for:
    a. Inserting a new user.
    b. Updating a username of an existing user (you know the id).
    c. Selecting all users sorting them by Username.
    d. Deleting a user by ID.

C# Task (ASPX Page)

Objective: Demonstrate session and cookie handling.

Task: Write a simple ASPX page where:

- A user enters his name in a form.
- The name is stored in a session and a cookie.
- The page welcomes the user using the stored data.

Basic Encryption & Hashing Task

Objective: Understand the importance of hashing and why passwords should not be stored in plain text.

Tasks:

1. Use C# to hash a password using **SHA256**.
2. Compare the output when hashing the same password multiple times (should be the same).
3. Extend the code to include **salting** and observe how the hash changes.

Discussion Points:

- Why do we use hashing for passwords instead of encryption?
- Why is salting important?
- How many different salts do we need to hash the passwords of 10 users and where would we store the salt(s)?
- What is the risk of storing passwords without hashing?

Same-Origin Policy (SOP) & CORS Task

Objective: Understand how the Same-Origin Policy (SOP) restricts JavaScript and how Cross-Origin Resource Sharing (CORS) can bypass it.

Task:

1. Create one simple HTML page and a simple json-file and put them on two sepereate instaces of a webserver:

    - index.html (runs e.g. on http://localhost:5000)
    - origin2.html (runs e.g. on http://localhost:5001)

2. Try to fetch data from the json-file while opening index.html in the browser.
3. Observe the browser's CORS error.

Expected Behavior:

• When origin1.html tries to fetch data.json, it fails due to the **Same-Origin Policy**.
• The browser should log a **CORS error**.

Discussion Points:

• What is the Same-Origin Policy?
• Why does the browser block cross-origin requests?
• How does enabling CORS solve this issue?
• What are the security risks of allowing CORS?