Web Application Programming

Week 9: Web Application Security(Common web vulnerabilities)

By Elubu Joseph - MSc.IS

Lecturer

Department of Information Technology

Kumi University

Email: josebulinda@gmail.com or jose@kumiuniversity.ac.ug

Agenda

1. Summary of previous lecture
2. Web Application Security(
    i. Introduction to web application security
    ii. Common web vulnerabilities (e.g., XSS, CSRF, SQL injection))
3. Build User Registration System

Summary of previous lecture

1. Building Dynamic Web Applications(Interaction with databases:
   i. Build User Login system,
   ii. User authentication and authorization
   iii. Sessions management)

Intro to web application security

Web application security refers to the practice of protecting web applications and the data they handle from security threats, vulnerabilities, and unauthorized access.

Ensuring the security of web applications is critical because they often process sensitive information and are exposed to the public internet, making them susceptible to various security risks.

Intro to web application security

Key aspects of web application security:

1. Authentication and Authorization:

i. Implement strong user authentication mechanisms to verify the identities of users. Use password hashing, multi-factor authentication (MFA), and secure authentication protocols.

ii. Authorize users to access only the resources and data they are entitled to. Role-based access control (RBAC) is a common approach.

2. Data Encryption:

Encrypt data in transit using secure protocols like HTTPS (SSL/TLS) to protect information exchanged between the user's browser and the web server.

3. Session Management:

Securely manage user sessions, including session creation, maintenance, and termination. Avoid session fixation and implement secure session storage and tokens.

Intro to web application security

Key aspects of web application security:

4. **Input Validation**:

   Implement input validation to prevent common security vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). Use input filtering, validation libraries, and parameterized queries for databases.

5. **Cross-Site Scripting (XSS) Mitigation**:

   Sanitize and escape user-generated content to prevent the execution of malicious scripts in a user's browser. Use security headers like Content Security Policy (CSP) to mitigate XSS attacks.

6. **Cross-Site Request Forgery (CSRF) Protection**:

   Generate and validate anti-CSRF tokens to prevent attackers from tricking users into making unintended requests on their behalf.

Intro to web application security

Key aspects of web application security:

8. **Security Headers**:

   Utilize security headers, such as HTTP Strict Transport Security (HSTS), to enforce secure connections and mitigate certain types of attacks.

9. **Secure File Uploads**:

   If your application allows file uploads, validate and restrict file types, store them in secure locations, and consider implementing content inspection to detect malicious files.

10. **API Security**:

    Secure APIs and ensure proper authentication and authorization for API endpoints. Implement rate limiting and monitor API usage.

Intro to web application security

Key aspects of web application security:

11. **Error Handling**:

    Avoid revealing sensitive information in error messages. Customize error messages and handle errors gracefully to prevent attackers from gaining insights into your application's internal workings.

12. **Security Updates and Patch Management**:

    Keep all software components (web server, application framework, libraries, and the operating system) up to date with security patches to address known vulnerabilities.

13. **Web Application Firewall (WAF)**:

    Consider using a WAF to filter and monitor incoming web traffic, providing an additional layer of protection against common web application attacks.

Intro to web application security

Key aspects of web application security:

15. **Security Testing**:

    Conduct security assessments, including penetration testing, vulnerability scanning, and code reviews, to identify and remediate security issues.

16. **Logging and Monitoring**:

    Implement logging to record security-relevant events and anomalies. Regularly review logs and set up alerts for suspicious activities.

17. **User Education and Training**:

    Train your development and operations teams in secure coding practices and raise awareness among end-users about safe online behaviors.

Intro to web application security

Key aspects of web application security:

20. Compliance and Regulations:

Ensure that your web application complies with relevant data protection and privacy regulations, such as Ugandan Data protection and privacy Act-2019, GDPR or CCPA, as applicable.

21.Incident Response Plan:

Develop and maintain an incident response plan to respond to and mitigate security incidents effectively.

Web application security is an ongoing process that requires a combination of technology, best practices, and vigilance. It's essential to stay informed about emerging threats and continuously update security measures to protect your web applications and user data.

Common web vulnerabilities

Common web vulnerabilities

Web vulnerabilities are weaknesses or security issues in web applications that can be exploited by malicious actors to compromise the confidentiality, integrity, or availability of data or the functionality of the application. Common web vulnerabilities include:

1. **Cross-Site Scripting (XSS)**:

   Stored XSS: Attackers inject malicious scripts into a web application, and the scripts are then displayed to other users, typically through user-generated content (e.g., comments or messages).

   Reflected XSS: Malicious scripts are injected into a URL, and users are tricked into clicking the link, causing the script to run in their browsers.

2. **Cross-Site Request Forgery (CSRF)**:

   Attackers trick authenticated users into making unauthorized requests to a web application, typically by crafting malicious links or forms. These requests can result in actions being taken on behalf of the victim without their consent.

Common web vulnerabilities

Key aspects of web application security:

3. SQL Injection:

Attackers inject malicious SQL queries into input fields, exploiting vulnerabilities in poorly sanitized or unsanitized data. This can lead to unauthorized access to databases and the retrieval, modification, or deletion of sensitive data.

4. Insecure Deserialization:

This vulnerability occurs when an application deserializes untrusted data. Attackers can manipulate serialized data to execute code, potentially leading to remote code execution.

5. Broken Authentication:

Poorly implemented or weak authentication mechanisms can lead to issues such as session fixation, weak password policies, and credential stuffing attacks.

Common web vulnerabilities

Key aspects of web application security:

8. **Security Misconfiguration**:

   This involves misconfigured security settings, such as improperly configured permissions, default passwords, and exposed sensitive files or directories. Attackers can exploit these misconfigurations to gain unauthorized access.

9. **Sensitive Data Exposure**:

   Failure to properly protect sensitive data can result in data exposure. This includes issues like weak encryption, insufficient authentication, and data leaks through unsecured APIs.

10. **XML External Entity (XXE) Injection**:

    Attackers can exploit misconfigured XML parsers by injecting malicious XML entities, potentially leading to disclosure of internal files or remote code execution.

Common web vulnerabilities

Key aspects of web application security:

11. Server-Side Request Forgery (SSRF):

Attackers manipulate an application into making requests to unintended internal or external resources, potentially leading to data exposure or further attacks.

12.Broken Access Control:

Inadequate access controls can allow unauthorized users to access restricted functionality, view other users' data, or perform actions they should not be allowed to.

13.File Inclusion Vulnerabilities:

Poorly sanitized user input used to include files can result in directory traversal attacks and arbitrary code execution.

Common web vulnerabilities

Key aspects of web application security:

14. Unvalidated Redirects and Forwards:

Attackers exploit open redirects and forwards in web applications to trick users into visiting malicious websites or performing unintended actions.

15.Remote File Inclusion (RFI):

This vulnerability occurs when an attacker can include external files or scripts in the application, potentially leading to remote code execution.

16.API Security Issues:

Poorly secured APIs can be susceptible to various issues, including unauthorized access, rate limiting bypass, and exposure of sensitive data.

Common web vulnerabilities

Key aspects of web application security:

17. **Content Security Policy (CSP) Bypass**:

    Attackers may attempt to bypass CSP directives, allowing them to execute scripts or load resources not approved by the security policy.

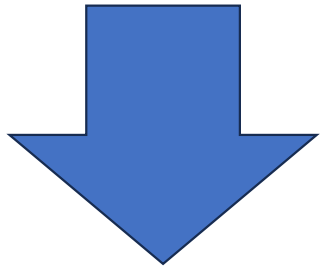
These vulnerabilities underscore the importance of implementing robust security practices, conducting regular security testing, and staying informed about emerging threats to protect web applications and their users from exploitation.

Building User Registration System

User registration system is a component or process that allows individuals to create user accounts or profiles within an application system.

The primary purpose of a registration system is to collect user information, authenticate users, and manage their access and privileges within the system.

Here are Key components and characteristics of a registration system:



Building User Registration System

Key components and characteristics of a registration system:

1. **User Information Collection**:

   Registration systems typically collect user information, which can include fields such as first name, last name, email address, username, password, and additional profile details. The specific information required may vary based on the application's needs.

2. **User Authentication**:

   During registration, the system typically verifies user identity through a username or email address and a secure password. This authentication process helps ensure that only authorized individuals can access the system.

3. **Data Validation and Verification**:

   Registration systems often include data validation to ensure that the information provided by users is in the correct format and adheres to predefined rules. This validation helps maintain data integrity and security.

Building User Registration System

Key components and characteristics of a registration system:

4. **Email Confirmation**:

   Many registration systems incorporate an email confirmation step. After registration, the system sends an email to the user's provided email address with a verification link. The user must click the link to confirm their email and activate their account.

5. **User Role and Privileges**:

   Registration systems can assign user roles and privileges, determining what actions or features users can access within the system. Common roles include regular users, administrators, and moderators.

6. **Account Activation and Deactivation**:

   A registration system allows for account activation upon successful registration. Additionally, users may have the ability to deactivate or delete their accounts if needed.

Building User Registration System

Key components and characteristics of a registration system:

7. **User Profile Management**:

   Users can typically manage their profiles, including updating personal information, changing passwords, and uploading profile pictures or additional details.

8. **Security Measures**:

   Registration systems should incorporate security measures to protect user data. This may involve secure password storage, encryption, and measures to prevent unauthorized access.

9. **User Experience**:

   A well-designed registration system provides a user-friendly experience. This includes clear and informative registration forms, error handling, and feedback messages.

Building User Registration System

Key components and characteristics of a registration system:

10. **Captcha and Security Checks**:

    To prevent automated account creation and spam, registration systems may use CAPTCHA challenges or other security checks.

11. **Terms of Service and Privacy Policies**:

    Users are often required to accept the application's terms of service and privacy policy during registration.

12. **User Data Storage and Management**:

    Registration systems store user data securely, often in a database. The data should be accessible only to authorized users.

Building User Registration System

Key components and characteristics of a registration system:

13. **Logging and Monitoring**:

    Registration systems may log user registration activities and monitor for suspicious or fraudulent registration attempts.

A registration system is a fundamental component of many applications, enabling users to create accounts, personalize their experiences, and access specific features. It plays a crucial role in user management, access control, and data security.

Building User Registration System Requirements

Building a registration system for a web application is a critical component, and you should pay careful attention to various requirements to ensure it functions smoothly, securely, and meets your users' needs.

Here are the key requirements for building a registration system:

1. **User Registration Form**:

   Develop a user-friendly registration form that collects necessary information, including first name, last name, email, username, and password. Additional profile information may be collected depending on your application's requirements.

2. **Data Validation**:

   Implement server-side data validation to ensure that user input is in the correct format and adheres to predefined rules. This prevents invalid or malicious data from being submitted.

Building User Registration System Requirements

Key requirements for building a registration system:

3. **Password Security**:

   Store user passwords securely by using strong, salted, and hashed password storage mechanisms (e.g., bcrypt). Do not store plain text passwords in your database.

4. **Username and Email Uniqueness**:

   Ensure that usernames and email addresses are unique to prevent duplicate accounts. Implement checks to verify whether a username or email is already in use before allowing registration.

5. **Email Confirmation**:

   Send a confirmation email to the user's provided email address with a verification link to activate the account. This step helps ensure the email is valid and owned by the user.

Building User Registration System Requirements

Key requirements for building a registration system:

6. **Account Activation**:

   Users should confirm their email addresses or perform an activation step before gaining full access to the system. Inactive accounts should have limited functionality.

7. **User Role Management**:

   Assign user roles and permissions based on user type (e.g., regular users, administrators, moderators) to control what users can and cannot do within the system.

8. **Privacy and Data Handling**:

   Implement measures to protect user data and privacy, including adherence to data protection regulations (e.g., GDPR- General Data Protection Regulation by EU-2018). Users should be informed about how their data is used and have the option to manage their privacy settings.

Building User Registration System Requirements

Key requirements for building a registration system:

9. **Terms of Service and Privacy Policy**:

   Require users to accept the terms of service and privacy policy during registration. Ensure these documents are accessible and up to date.

10. **Captcha and Security Checks**:

    Use CAPTCHA or other security checks to prevent automated account creation and spam.

11. **User Profile Management**:

    Allow users to manage their profiles, update personal information, change passwords, and upload profile pictures or additional details.

Building User Registration System Requirements

Key requirements for building a registration system:

12. **Account Deactivation and Deletion**:

    Provide users with the ability to deactivate or delete their accounts if they choose to do so.

13. **Account Recovery**:

    Implement a process for users to recover their accounts if they forget their passwords or need to reset them.

14. **Logging and Monitoring**:

    Log registration activities and monitor for suspicious or fraudulent registration attempts. Implement alerting systems for suspicious activities.

Building User Registration System Requirements

Key requirements for building a registration system:

15. **Security Measures**:

    Implement security measures to protect against common web application vulnerabilities, such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF).

16. **User Experience (UX)**:

    Design registration forms and processes with a focus on user experience, including clear instructions, user-friendly error messages, and mobile responsiveness.

17. **Testing and Quality Assurance**:

    Thoroughly test the registration system to ensure it functions correctly and securely, including functional testing, security testing, and usability testing.

Building User Registration System Requirements

Key requirements for building a registration system:

8. **Documentation**:

   Document the design, architecture, and usage of the registration system for future reference and for other developers working on the project.

9. **Legal and Regulatory Compliance**:

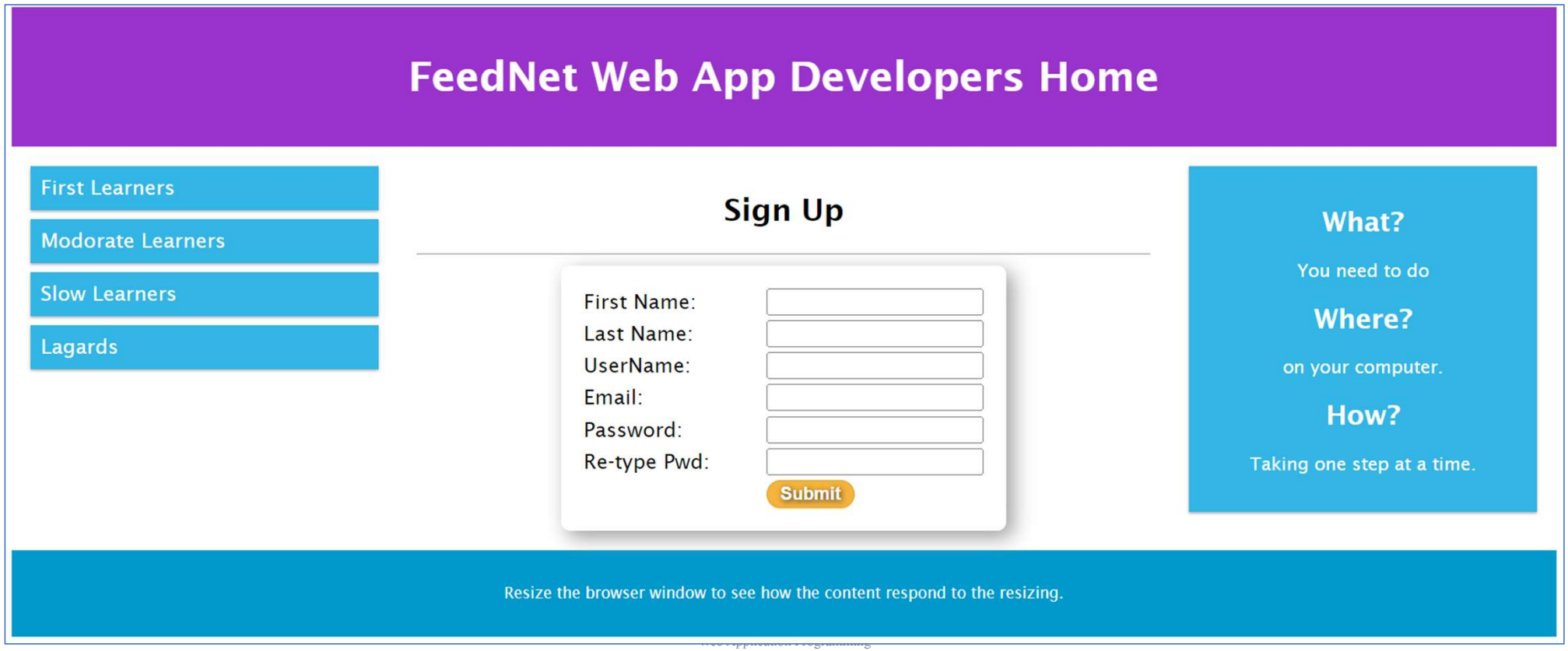
   Ensure that the registration system complies with all relevant legal and regulatory requirements, especially if you're dealing with user data and privacy.

10. **Scalability and Performance**:

    Design the registration system to be scalable and handle increased user registration loads efficiently.

Building a registration system that meets these requirements is essential for user data security, legal compliance, and a positive user experience. It's important to continuously monitor and update the system to adapt to evolving security threats and user needs.

Building User Registration System

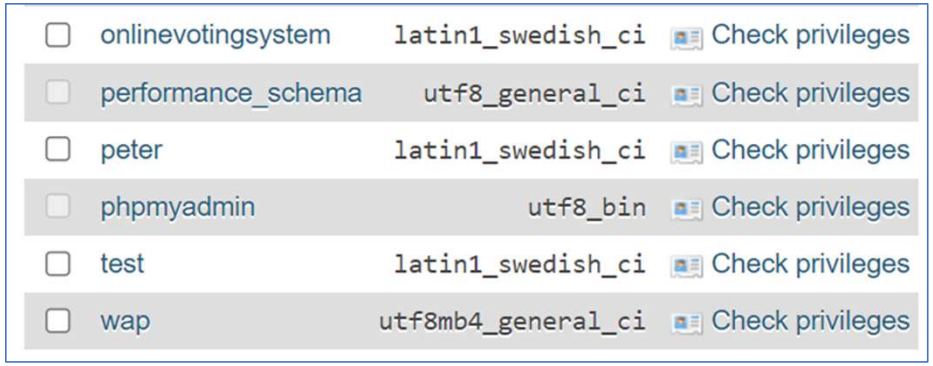


Building User Registration System Requirements

We may need the following items for user registration to be successful.

1. **Database** + a table to store users' data. in our case (**wap** + **users**)
2. **Database connection page**/code. E.g **connect.php**
3. **User Registration** page- containing registration form and data submission code. E.g. **register.php**
4. **index.php** – is the landing place after successful user registration
5. **form.css** – contains styles for the form
6. **major.css -** contains the button styles used on the form button

Database(wap) and table(users)



☐	onlinevotingsystem	latin1_swedish_ci	Check privileges
☐	performance_schema	utf8_general_ci	Check privileges
☐	peter	latin1_swedish_ci	Check privileges
☐	phpmyadmin	utf8_bin	Check privileges
☐	test	latin1_swedish_ci	Check privileges
☐	wap	utf8mb4_general_ci	Check privileges

Note the presence of wap database in this list.

id	firstname	lastname	email	username	password
1	Micheal	temba	tembo@gmail.com	temba	1234
2	Kadimba	Felix	kadims@gmail.com	kadimba	123
4	Elubu	Joseph	josebulinda@gmail.com	jose	202cb962ac59075b964b07152d234b70
5	Lemi	Agrey	codingissweet@gmail.com	ag	81dc9bdb52d04dc20036dbd8313ed055

Note current status of our users table.

Building User Registration System
Connect.php

```php
<?php
	$server = "localhost";
	$username = "wap";
	$password = "wap";
	$database = "wap";
	$conn = new mysqli($server, $username, $password, $database);

	if ($conn->connect_error) {
		die("Connection failed: " . $conn->connect_error);
	}
	else{
	//echo 'connection successful.’;
	}
?>
```

Building User Registration System
Registration page - form

```
78      <center><H2>Sign Up</H2><hr>
79  <form id="commentform" name="reg" method="POST" action="register.php">
80  <table>
81  <tr><td width=100%>First Name:</td>
82   <td><input name="firstname">  </td>
83   </tr><td>Last Name: </td>
84   <td><input name="lastname"></td></tr>
85  <tr><td width=100%>UserName: </td>
86   <td><input type="text" name="username"></td></tr>
87  <tr><td width=100%>Email: </td>
88   <td><input type="email" name="email"></td></tr>
89  <tr><td>Password:</td>
90   <td><input type="password" name="password" ></td></tr>
91  <tr><td>Re-type Pwd:</td>
92   <td><input type="password" name="re_type_pwd" ></td>
93   </tr><tr><td></td>
94   <td><input class="button" type="submit" name="reg" value="Submit"></td>
95   </tr></table></form>
96   </center>
```

Building User Registration System
Registration page – form code explained

The code is an HTML form that serves as a user registration form. It allows users to input their personal information and submit it to a server-side script (specified in the **action** attribute) for processing. Here's an explanation of the HTML code:

```
<center><H2>Sign Up</H2><hr>
```

This part of the code is responsible for displaying a centered heading "Sign Up" with a horizontal line (hr) underneath it. It's a basic HTML heading and horizontal line element to provide a clear title for the registration section.

Building User Registration System
Registration page – form code explained

Explanation of the HTML code:

```
<form id="commentform" name="reg" method="POST" action="register.php">
```

This is the opening **<form>** tag, which defines the start of an HTML form. The attributes and their meanings are as follows:

1. **id="commentform"**: This attribute assigns an ID to the form, which can be used for styling or scripting purposes.
2. **name="reg"**: This attribute assigns a name to the form, which can be used to identify the form in scripts.
3. **method="POST"**: This attribute specifies the HTTP method used to submit the form data, which, in this case, is POST. This means that the form data will be sent to the server in the request body.
4. **action="register.php"**: This attribute specifies the URL (in this case, "register.php") to which the form data will be sent for processing when the user submits the form.

Building User Registration System
Registration page – form code explained

Explanation of the HTML code:

```
<table>
<tr> <td width=100%>First Name:</td>
<td><input name="firstname"></td> </tr>
```

This **<table>** tag is used to create an HTML table, which is a common way to structure form elements

Within the table, there is a table row **(<tr>)** that contains two table data cells **(<td>).** These two cells are used to create a label and an input field for the "First Name."

1. **<td width=100%>:** This cell defines a label for the "First Name" input field. The label is set to occupy the full width of the cell.
2. **<td><input name="firstname"></td>:** This cell contains an HTML <input> element with the name attribute set to "**firstname**." This is the actual input field where users can enter their first name.

Building User Registration System
Registration page – form code explained

The code continues in a similar structure for other form fields:

i. "**Last Name**" with an input field.

ii. "**Username**" with an input field of type "text."

iii. "**Email**" with an input field of type "email."

iv. "**Password**" with an input field of type "**password**" (for hiding the entered characters).

v. "**Re-type Pwd**" with an input field of type "**password**."

Building User Registration System
Registration page – form code explained

```
<tr> <td></td> <td><input class="button" type="submit"
name="reg" value="Submit"></td> </tr>
```

This section creates the submit button for the form. It consists of a table row with two cells. The first cell is empty (<td></td>), and the second cell contains the submit button:

<input class="button" type="submit" name="reg" value="Submit">:

This input element is of type "**submit**," which means it will trigger the form submission when clicked. The "name" attribute is set to "**reg**," and the "value" attribute is set to "Submit," which is the text displayed on the button. The "**class**" attribute is set to "button," which suggests that it might have associated CSS styles to make it appear as a button.

Complete User Registration Code php

login.php | success.php | logout.php | register.php | connect.php | code.php | login.php | core.inc.php | index.php | register.php

```php
<?php
    require "connect.php";
    require "core.inc.php";

if (isset($_POST['reg'])) {
    $firstname = $_POST['firstname'];
    $lastname = $_POST['lastname'];
    $username = $_POST['username'];
    $email = $_POST['email'];
    $password = $_POST['password'];
    $password_again = $_POST['re_type_pwd'];

    // You should add validation and security measures here before inserting the data.
    if (!empty($firstname) && !empty($lastname) && !empty($username) &&
        !empty($email) && !empty($password) && !empty($password_again)) {

        if ($password != $password_again) {
            echo "Password do not match";
        } else {
            // Check if the username or email already exists
            $stmt = $conn->prepare("SELECT id FROM users WHERE username = ? OR email = ?");
            $stmt->bind_param("ss", $username, $email);
            $stmt->execute();
            $stmt->store_result();

            if ($stmt->num_rows > 0) {
                echo "Username or email already exists. Please choose a different one.";
            } else {
                // Insert data into the table
                $password_hash = md5($password);
                $stmt = $conn->prepare("INSERT INTO users (firstname, lastname, username, email, password) VALUES (?, ?, ?, ?, ?)");
                $stmt->bind_param("sssss", $firstname, $lastname, $username, $email, $password_hash);
```

Complete User Registration Code

```php
                if ($stmt->execute()) {
                    header("Location: index.php");
                } else {
                    echo "Error: " . $stmt->error;
                }
            }
            $stmt->close();
        }
        $conn->close();
    } else {
        echo "Some field(s) are empty";
    }
}

?>
```

Complete User Registration php Code-Explained

The code above is for handling user registration while preventing duplicate usernames and email addresses in your user database. Here's an explanation of the code:

1. **User Input Validation**:

   The code begins by retrieving user input from the registration form: **firstname**, **lastname**, **username**, **email**, **password**, and **password_again**.

```
if (isset($_POST['reg'])) {
$firstname = $_POST['firstname'];
$lastname = $_POST['lastname'];
$username = $_POST['username'];
$email = $_POST['email'];
$password = $_POST['password'];
$password_again = $_POST['re_type_pwd'];
```

Complete User Registration php Code-Explained

The code above is for handling user registration while preventing duplicate usernames and email addresses in your user database. Here's an explanation of the code:

2.Input Validation:

It checks if all the required fields (i.e., **firstname**, **lastname**, **username**, **email**, **password**, and **password_again**) are not empty. If any of these fields are empty, it displays an error message: "Some field(s) are empty."

```
// You should add validation and security measures here before inserting the data.
if (!empty($firstname) && !empty($lastname) && !empty($username) &&
!empty($email) && !empty($password) && !empty($password_again)) {

if ($password != $password_again) {
        echo "Password do not match";
} else {
// Check if the username or email already exists
```

Complete User Registration php Code-Explained

The code above is for handling user registration while preventing duplicate usernames and email addresses in the database. Here's an explanation of the code:

3.Password Matching Check:

It checks if the password and the re-typed password (**password** and **password_again**) match. If they don't match, it displays an error message: "Password do not match.

```
if ($password != $password_again) {
    echo "Password do not match";
} else {
// Check if the username or email already
exists
```

Complete User Registration php Code-Explained

4. Database Query to Check for Duplicates:

The code prepares a SQL query to check if either the username or email already exists in the **users** table. It uses prepared statements to prevent SQL injection.

```
$stmt = $conn->prepare("SELECT id FROM users WHERE
username = ? OR email = ?");
$stmt->bind_param("ss", $username, $email);
$stmt->execute();
$stmt->store_result();
```

This query checks for existing records with the same username or email. If any records are found, it means a duplicate exists.

Complete User Registration php Code-Explained

5. Duplicate Check Result:

It checks if the query found any duplicates (i.e., **$stmt->num_rows > 0**). If duplicates are found, it displays an error message: "Username or email already exists. Please choose a different one."

```
// Check if the username or email already exists
$stmt = $conn->prepare("SELECT id FROM users WHERE username = ? OR email = ?");
$stmt->bind_param("ss", $username, $email);
$stmt->execute();
$stmt->store_result();

if ($stmt->num_rows > 0) {
        echo "Username or email already exists. Please choose a different one.";
} else {
//Data inserted
```

Complete User Registration Code-Explained

6. Insert Data into Database:

If no duplicates are found, the code proceeds to insert the user's information into the database. It first calculates a hash of the password using **md5** (note that using more secure hashing methods like bcrypt is recommended).

```
$password_hash = md5($password);
```

It then prepares an SQL statement to insert the user's data into the **users** table using prepared statements to ensure security.

Complete User Registration php Code-Explained

```
$stmt = $conn->prepare("INSERT INTO users (firstname,
lastname, username, email, password) VALUES (?, ?, ?, ?,
?)");
$stmt->bind_param("sssss", $firstname, $lastname, $username,
$email, $password_hash);
```

If the insert operation is successful, the user is redirected to the "**index.php**" page.

If there is an error during insertion, it displays an error message: "Error: "

followed by the error message from the database.

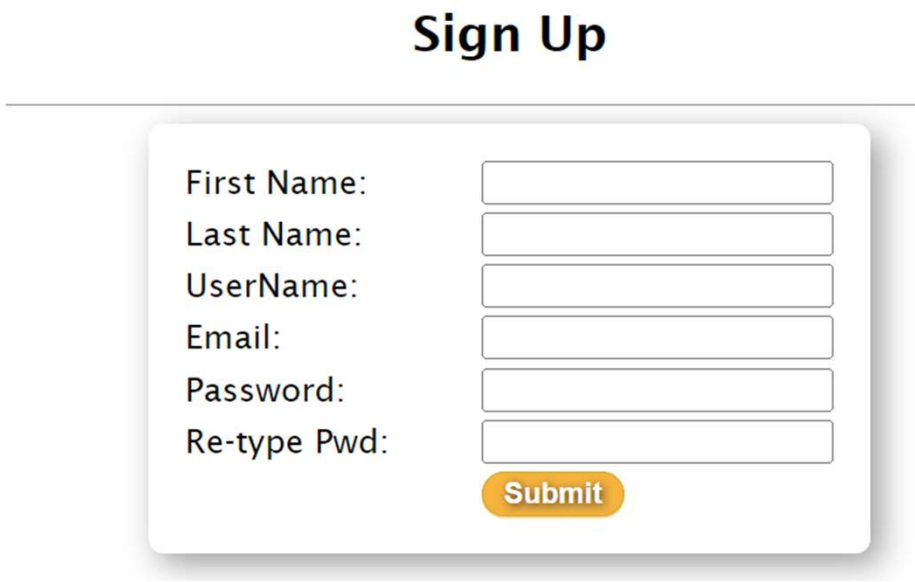
Complete User Registration php Code-Explained

7. Database and Statement Handling:

The prepared statement is closed and the database connection is closed after the operations are completed.

This code improves the user registration process by checking for duplicate usernames and email addresses in the database before allowing the user to create an account. It enhances data integrity and reduces the likelihood of duplicate user entries. However, it's essential to consider stronger password hashing methods and additional security measures for a production system, OpenAI. (2021).

Complete User Registration form display



Finished form display

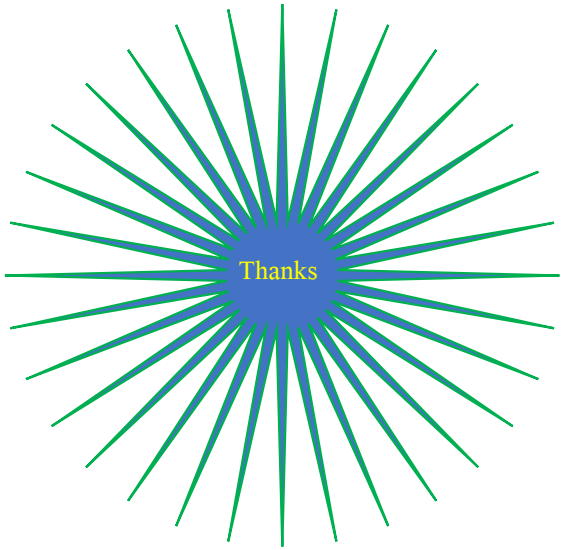
[Visit Registration page]

Summary

Summary

1. Web Application Security(
   i. Introduced web application security
   ii. Looked at Common web vulnerabilities (e.g., XSS, CSRF, SQL injection))
2. Built User Registration System(with each part of code explained)

Thank you for Listening



References

JavaScript Data Types. OpenAI Knowledge Base. OpenAI. (2021, September 22)..