**Written Assignment 15%**

Complete the assignment and submit by end of week 13 (By 16 Jan 2023, 1200 Noon)

Submission is via LMS under week 12 section.

**SECTION A**

**Purpose**

To investigate good security features offered by a typical web application.

**Tasks**

Describe the good security features offered by a typical web application based on the features listed below. Use the learning material from blackboard (Week 1 to 9) as well as some web research. Below are the web application features.

* **List** and **describe** 3 security features that you will implement for each of the “Web Application Features” listed in SECTION B. A good example of a security feature for login page feature will be “account lockout”.
* For each of the security feature:
  + Explain how the security feature should be implemented. E.g. describe the steps or process involved.
  + Explain the security issue(s) or problem that you are trying to avoid or mitigate. You may quote references from OWASP or SANS.
  + Provide reasons to support your solution(s) or implementation. (e.g. explain why do you think your solution will work?)
* You are not required to code the application. However, you may include code snippets or screenshots to help illustrate your solution.

**Assumption/rules:**

* All the web application is assumed to be a typical ASP.NET or ASP.NET Core.
* No limit to the number of words for this assignment.
* You may provide screenshots to illustrate your solution(s)

**SECTION B**

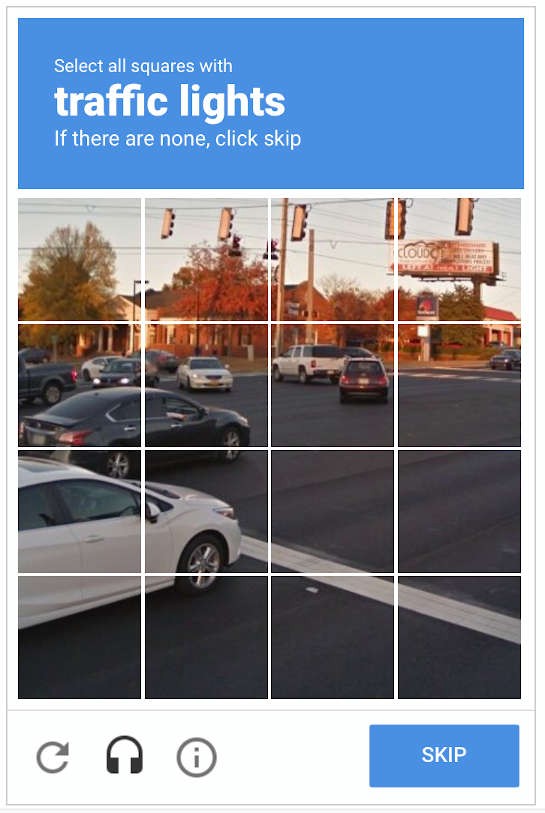
**Web Application Features (Choose any 2 from the list) – 15 marks each**

1. Login page
2. Forget Password or account recovery process
3. Payment page with Credit card Info (Credit card info are required to be saved in the database)
4. Account Registration Page

**CAPTCHA**

The first security feature that should be implemented is CAPTCHA. Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) generates challenge-test graded tests that any humans can pass but current computer programs cannot.

Image of a CAPTCHA test



To implement CAPTCHA in your website, your website needs to be registered at [ReCaptcha service](https://www.google.com/recaptcha/admin/create) to obtain a public and private key.

Add this code to your \_AppStart.cshtml file.

Text

Description automatically generated

Create a new webpage named Recaptcha.cshtml and replace the existing code with this code.

Text

Description automatically generated

CAPTCHA ensures that registration for services is only limited to humans and not bots to prevent the waste of a service’s resources and reduce opportunities for fraud. Brute force attacks are a common threat web developers face, according to an [OWASP article](https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks) by Esheridan. CAPTCHA also ensures that brute force attacks are not possible on the login page by appearing once every 20 wrong attempts without affecting user experience.

**Multi-Factor Authentication**

The second security feature that should be implemented is Multi-Factor Authentication. MFA provides an extra layer of security by having users provide more than one piece of information for identification. The information required is typically something the user knows (passwords, answers to secret questions) and has (credit cards, hardware tokens, phone).

To implement MFA, we will use a Time-based One-Time Password (TOTP) which is defined in [RFC 6238](https://www.rfc-editor.org/rfc/rfc6238).

Download the qrcode.js JavaScript Library [here](https://davidshimjs.github.io/qrcodejs/). Follow the instructions in [Scaffold Identity](https://learn.microsoft.com/en-us/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-7.0&tabs=visual-studio) to generate /Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml.

Create a new JavaScript file called qr.js in wwwroot/js and add the code below to generate the QR code.

Text

Description automatically generated

Locate the Scripts section in /Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml at the end of the file and update the code by adding a reference to the qrcode.js library.

Add the qr.js file with the call to generate the QR code.

Text

Description automatically generated

Multi-Factor Authentication ensures that hackers cannot immediately gain unauthorized access to user accounts should they manage to breach through the first layer of security. An alert will be sent to users to change their passwords before hackers can cause any more harm when attempting to breach the next layer of security. TOTP has become an industry-recommended approach for 2FA over SMS due to its shorter lifespan of the generated one-time passwords.

**Input Validation**

The third security feature that should be implemented is input validation. Input validation is the testing of any input (or data) provided by a user or application against expected criteria. It uses two distinct sets of criteria

for validation. These criteria for comparison can be an **allow list** or **deny list.**

To implement input validation, here are some examples of effective enforcement of syntactic and semantic correctness:

1. Type conversion with strict exception handling
2. Enforce minimum and maximum value range checks for numerical parameters and dates, minimum and maximum length checks for strings
3. Create an array of allowed values for small sets of string parameters (days in a week)
4. Add regular expressions for any other structured data covering the whole input string and not using “any character” wildcard (. or \S)

Input validation ensures only properly formed data is entering the workflow in a system to prevent malformed data from persisting in the database and triggering malfunction of various downstream components.

Report Template ( ARIAL, FONT 12)

Student Full Name : Teh Siang En

Admin No : 213032L

Tutorial Group : IT2163-01

I have chosen the following security features for my assignment:

1. Login Page (Page 2-x)
2. Payment page with Credit card Info (Page x-y)

Statement on Plagiarism

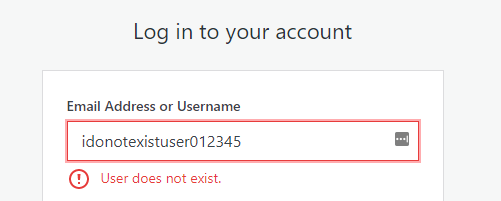
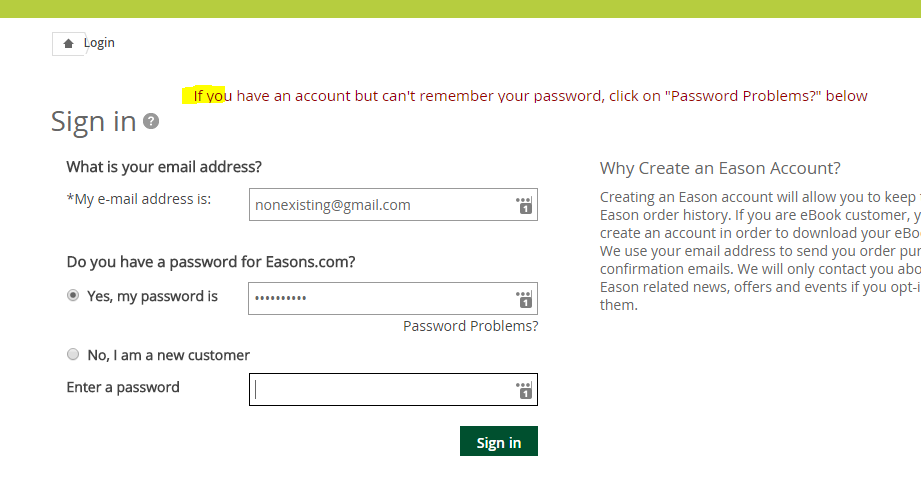
I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication.

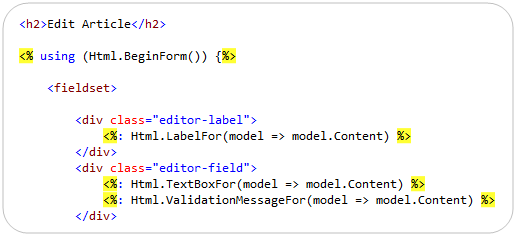
I also certify that this assignment/report has not previously been submitted for assessment in any other unit, and that I have not copied in part or whole or otherwise plagiarised the work of other students and/or persons.

|  |  |
| --- | --- |
| Signature |  |
| Date | 15/1/2023 |

**1. Login Page**

Assuming the login page provides a verbose error message stating that the user does not exist when providing an invalid username and user-supplied input is not validated.

* 1. **User Enumeration**  
     User enumeration is when an application responds with a verbose error message stating that the user does not exist or when the difference in server response timing gives a hint that the username is valid/invalid.  
     Referenced from OWASP - [Testing for Account Enumeration and Guessable User Account](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account)  
       
       
       
     User enumeration can be remediated by implementing proper handling through the use of generic error messages instead of verbose error messages on valid/invalid usernames or adding a delay before a successful authentication redirect. This prevents attackers from enumerating all valid users of the application to further utilize them to perform targeted attacks such as brute-forcing and social engineering.  
       
     

* 1. **Cross-Site Scripting (XSS)**  
       
     The OWASP definition of XSS is whenever an application includes untrusted data in a new web page without proper validation or escaping or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript.   
       
     XSS attacks are divided into three types:  
     1. Reflected XSS (non-persistent) – scripts are executed on the victim’s side and not stored on the server  
     2. DOM-based XSS – attack payload is executed as a result of modifying the Document Object Model (DOM) environment.  
     3. Persistent (Stored) XSS – injected scripts are stored on vulnerable servers  
       
     To reduce the risk of XSS, all input data must be validated, ensure that only allowListed data is allowed and that variable outputs in a page are encoded before returning to the user.  
       
     Encoding escapes user input by interpreting it as data, not as code.  
     Validation filters user input so that the browser interprets it as code without malicious commands.  
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
       
     **1.3 Injection**  
       
     According to the [OWASP Injection Theory](https://owasp.org/www-community/Injection_Theory#:~:text=Injection%20is%20an%20attacker's%20attempt,being%20sent%20to%20an%20interpreter.), injection is an attempt by attackers to send data to an application in a way that will change the meaning of commands being sent to an interpreter.   
       
       
     Successful attacks will result in significant data breaches and incurred costs. Due to the complexity of injection, it can be overwhelming to even security specialists. Untrusted data often comes from HTTP requests in the form of URL parameters, form fields, headers or cookies. Every interpreter has parsers that can trick interpreters into interpreting data as commands.  
       
     There are two types of SQL injection:  
     1. In-band SQL injection (Simple SQLi)  
      a. Error-based – relies on an error message thrown by the database server to obtain information about the structure of the database  
      b. Union-based – leverages the UNION SQL operator to combine the results of two or more SELECT statements into a single result to be returned as an HTTP response  
       
     2. Inferential SQL injection (Blind SQLi)  
      a. Boolean-based blind SQLi – relies on sending an SQL query to force the application to return a TRUE or FALSE result  
      b. Time-based blind SQLi – forces the database to wait for a specified amount of time before responding, using the response time to indicate whether the result of the query is TRUE or FALSE  
       
     Below is an example of SQL injection  
     Graphical user interface, text, application, email

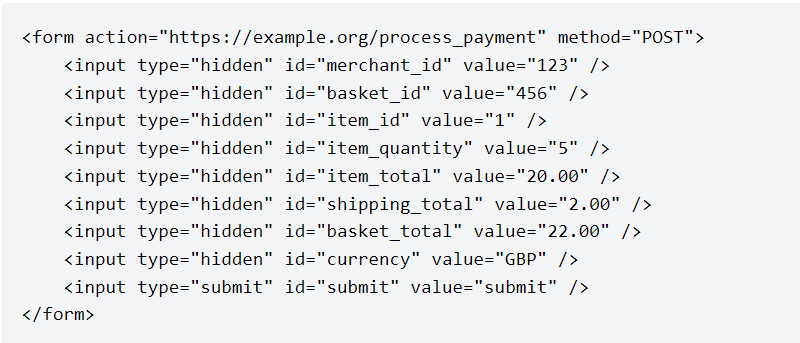
     Description automatically generated  
     Graphical user interface, text, application

     Description automatically generated  
       
       
       
       
     To mitigate the risk of SQL injection, use prepared statements with parameterized queries to distinguish code between data, regardless of what the user-supplied input is. Additionally, escaping all-user supplied input to ensure the database does not confuse input with SQL code to avoid SQL injection possibilities.  
     Graphical user interface, text, application

     Description automatically generated  
       
     Finally, the implementation of stored procedures through the use of parameters can prevent SQL injection. Below is the revised code for a stored procedure.  
     Graphical user interface, text, application, email

     Description automatically generated

**2. Payment page with credit card information**

Assuming the payment page redirects users to a third-party payment gateway. Client-side requests are item IDs, quantities, and prices. User-supplied input is not properly validated.   
  
  
**2.1 Parameter Tampering**  
According to the [OWASP Business Logic Testing](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/10-Test-Payment-Functionality), most e-commerce websites allow users to add items to a basket before checking out. If the website does not validate user-supplied quantity values, it may be possible to specify a decimal quantity or a negative quantity. To modify the contents of the basket, users can repeatedly remove items until the quantity is negative and add or update a negative quantity value. It may also be possible to tamper with the values if the whole basket detail is passed instead of just the total value to the third-party payment gateway. Similarly, since the price is included in the client-side request, it is possible to tamper with the prices between the application and the gateway.  
  
By modifying the HTML form or intercepting the POST request, the prices can be modified to effectively purchase them for less. Most payment gateways do not accept price values below 0, so 0.01 is likely to succeed. However, some payment gateways accept negative values to process refunds.  
  
  
To mitigate the risk of tampering, input validation should be in place for all user-supplied inputs. Details of the request should be encrypted and hashed to verify that the transaction has not been tampered with.

**2.2 Breaking Payment Flows**  
Multiple stages in the checkout or payment process mean that it is possible to cause unintended behavior to occur. For example, modifying the shipping address after entering the billing details to reduce shipping costs removing items after shipping details to avoid minimum basket value, etc. It is also possible to redirect users to the successful transaction page by simply including the transaction ID.  
  
To mitigate the risk of payment flows breaking, ensure data is encrypted and hashed upon the confirmation of details to ensure proper payment flow.

**References**

1. OWASP Top 10 - <https://owasp.org/www-project-top-ten/>
2. <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/10-Test-Payment-FunctionalityBbb>
3. <https://redhuntlabs.com/blog/10-most-common-security-issues-found-in-login-functionalities.html>
4. <https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html>
5. <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html>
6. <https://www.mikesdotnetting.com/article/113/preventing-sql-injection-in-asp-net>
7. Lecture Slides