Application Security IT2163 Written Assignment 1

Student full name : Mah wen jie

Admin No : 222798P

Tutorial Group : IT2163-02

I have chosen the following security features for my assignment:

1. Login page (page 2-5)
2. Payment page with Credit card information (page 6-8)

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 : A black letter with a white background

Description automatically generated

Date : 4/1/2024

1. **Login Page**
   1. **Security issue 1: Brute forcing**

Brute forcing is when an attacker systematically attempting logins with a set of predetermined values, or/and tries possible combinations of passwords until the correct one is found.

Brute forcing can be done with dictionary attacks (with/without mutations) or traditional brute force attacks. Automation is often used in brute-force attacks to enhance the efficiency of the attacks.

Ref: <https://owasp.org/www-community/attacks/Brute_force_attack>

**Mitigation technique #1: Account Locking**

Enforce account lockout policies that temporarily lock users accounts after a few incorrect password attempts. Account lockouts can last a specific duration (10 mins, 1 hour) or it could remain locked until an administrator manually unlocks it, or the user manually resets their password.

However, account lockouts are sometimes effective, but should not be used as the sole security measure as attackers can cause denial of service (DoS) by locking out many accounts, wasting much time and resource.

IP Bans for multiple failed attempts can be put in place to prevent continuous brute force attempts, however it can be circumvented by using widely available open proxy lists.

To implement this in ASP.NET Identity, we can use the LockoutEnabled and LockoutEndDateUtc properties, the former property indicates that lockout is enabled for a user and that user has a chance to be locked out, and the latter property indicates when the lockout ends.

A close-up of a white background

Description automatically generated

code locks out account for 10 minutes, to lockout users indefinitely, set lockoutenddateutc property to DateTimeOffset.MaxValue

Ref: <https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks>

Ref: <https://dotnettutorials.net/lesson/how-to-lockout-a-user-account-in-asp-net-identity/>

**Mitigation Technique #2: CAPTCHA**

Completely Automated Public Turing Test to tell Computers and Humans Apart

CAPTCHA Can be used to catch automated “bots” and slow down attackers in brute force attempts.

There are many types of captchas but the most used captchas for login pages are google captchas version 3. They do not require visitors to solve any challenges, instead continuously monitor behavior of visitors to determine if it’s a human or bot.

The easiest method for using reCAPTCHA v3 on your page is to include the necessary JavaScript resource and add a few attributes to your html button.

Javascript:

<script src="https://www.google.com/recaptcha/api.js"></script>

<script>  
    function onSubmit(token) {  
      document.getElementById("demo-form").submit();  
    }  
  </script>

HTML:

<button class="g-recaptcha"   
        data-sitekey="reCAPTCHA\_site\_key"   
        data-callback='onSubmit'   
        data-action='submit'>Submit</button>

A black text on a white background

Description automatically generated

Ref: <https://developers.google.com/recaptcha/docs/v3>

* 1. **Security issue 2: Poor session management (session hijacking)**

Session hijacking attack is the exploitation of web session control mechanism.

It compromises the session token by stealing or predicting another user’s session token to gain access to the user’s webserver and completely bypass authentication.

Session hijacking from a login page can happen over these common methods:

man-in-the-middle attacks, session sniffing, session token predicting, client-side attacks (XSS, Trojans etc.)

Ref: <https://owasp.org/www-community/attacks/Session_hijacking_attack>

**Mitigation technique #1: Secure session management**

Implementing timeouts and not recycling session IDs can help secure user’s login credentials.

Implementing timeouts expire sessions automatically which limits exposure to attackers that can potentially hijack the sessions.

Timeouts can be added in the web.config file into the <system.web> section.



timeout = "1" is equivalent to 1 minute before session timeout

Generating new session IDs and disposing session IDs when users leave the site can prevent session fixation, an attack where the attacker sets the session ID and users log into the fixed session ID, allowing unauthorized access to the attackers.

We can use Session.Abandon() in the logout, which removes all variables stored in the session, and a new session ID will be given to the user on the next login.

But to bullet proof the attack, we can create another cookie with a unique value which will be stored on the session as well. On every page load, the cookie value will be matched with the session value, granting access if they match.

A computer code with text

Description automatically generated



Create a GUID and save it as a new session variable "AuthToken", it is also saved in a cookie named AuthToken.  
Cookie value is checked with session value, deny access if both don’t match.

Ref: <https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/how-to-prevent-session-hijacking-attacks/>

**Mitigation Technique #2: HTTPS and HTTPOnly cookies**

Ensure HTTPS (Including the login page) as it encrypts communication between client and server and allows for secure communication through use of private key exchange, preventing interception of session data through packet sniffing.

Using HttpOnly cookies also mitigates client-side scripts that come from cross-site scripting (XSS) attacks, which prevents sensitive data (like cookies) from being accessed.

Via the web.config in the system.web/httpCookies element:



or through code:

A blue background with black text

Description automatically generated

Ref: <https://owasp.org/www-community/HttpOnly>

* 1. **Security issue 3: Weak Password Policies.**

Weak password policies affect both the server and client. Allowing users to set weak passwords exposes both parties to brute force attacks, credential stuffing and dictionary attacks.

Most common passwords are still easily guessed, as they follow a common set of patterns (e.g. 123456, qwerty, password) to prevent this, we need to enforce password policies and test for them during account creation.

**Mitigation Technique #1: Enforcing stronger policies.**

Stronger password policies upon account creation or password reset includes increased password complexity and length, a minimum requirement of characters from different character sets such as lower- and upper-case letters, digits, and special characters.

Additionally, other policies can also be implemented to ensure that passwords remain secure and safe. Such policies include password history requirements where user’s password must not show much similarity upon each reset, ensuring that passwords do not contain easy-to-guess patterns/words, non-frequent changing of passwords, regular password expiration etc.

Regular expression, or Regex, is a simple way of enforcing long password policies in a straightforward manner.



Ensures that password meets the requirements : minimum 8 characters maximum 10 characters, at least one upper-case and lowercase letter, one number, and one special character

Ref: <https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/07-Testing_for_Weak_Password_Policy>

Ref: <https://stackoverflow.com/questions/19605150/regex-for-password-must-contain-at-least-eight-characters-at-least-one-number-a>

1. **Payment page with Credit Card Information**
   1. **Security issue 1: Insecure handling of sensitive data**

One of the biggest risks when it comes to dealing with credit card information saved in the database is the fact that there are many credit card credentials available in a convenient space for the attackers. If breached, the damage will be immense.

Attackers can also siphon out data in many ways, from SQL Injections to insecure communication.

To safely store credit card information, it is a rule of thumb to follow the PCI Security Standards as a guideline to protect sensitive card information.

Ref: <https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf>

**Mitigation Technique #1: Proper encryption**

Encrypting data in transit:

end-to-end communications security should be considered when sending sensitive data over any network. TLS (transport layer security) must be properly configured in a variety of ways as a defense mechanism for secure communications.

TLS protects web application data (in this case credit card information) from unauthorized disclosure and modification when being transmitted between client and server, and between server and back-end.

Enable and configure HTTPS in application through obtaining an SSL/TLS certificate for domain.

Using strong data encryption algorithms before storing in database:

AES (Advanced Encryption Standard) is a strong symmetric encryption algorithm commonly used to store sensitive or classified data.

Before we store credit card information, we will use AES algorithm as it is widely recognized as a secure and reliable encryption standard.

Here is a way to implement it:

A screenshot of a computer screen

Description automatically generated

plainText is a placeholder for any fields on the credit card that is to be encrypted.

Ref: <https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere>

Ref: <https://gocardless.com/guides/posts/how-to-store-credit-card-information/>

* 1. **Security issue 2: Card Cracking**

Card cracking is a brute force attack (usually automated) against application payment card processes to identify missing values (expiry date, cvc).

Card cracking allows attackers to gain unlawful and unauthorized access to other people’s cards.

**Mitigation Technique #1: 2FA**

Adding an extra layer of security on top of the credit card information input (like a password or an SMS) can prevent unauthorized access to cards.

An example would be an SMS OTP before confirming the purchase or saving of the credit card.

Steps taken:

system verifies card -> system generates 6-digit OTP (one time password) -> system prompts user for OTP -> system validates OTP -> if OTP valid grant access.

Repeat if OTP has expired or user has requested another.

A screenshot of a cell phone

Description automatically generated

Ref: [https://owasp.org/www-project-automated-threats-to-web applications/assets/oats/EN/OAT-010\_Card\_Cracking](https://owasp.org/www-project-automated-threats-to-web%20applications/assets/oats/EN/OAT-010_Card_Cracking)

**Mitigation Technique #2: CAPTCHA**

As mentioned earlier, a CAPTCHA challenge can slow down bot activity (mainly brute forcing) for the attackers.

Dealing with credit cards, implementing captcha can make it harder for attackers to crack missing information from card.

* 1. **Security issue 3: Injection**

Cyber-attacks on e-commerce platforms allow for the unauthorized online POS Skimming, which returns credit card information back to the attacker. This is often due to a weakness found in the website’s payment processing system, through the injection of malicious code.

Injection can weaken a website’s infrastructure and expose sensitive information, without a doubt security measure must be implemented.

**Mitigation Technique #1: Input validation**

Filtering user input in fields ensures that browser interprets input solely as code without any malicious commands.

By default, web.config file will have input validation already set up in the ASP.NET web app framework.

However, we can take it a step further by enabling filtering through blacklisting, whitelisting, and cleansing of input.

Ref: <https://securityintelligence.com/posts/e-commerce-skimming-the-new-pos-malware/>

References

OWASP brute force attacks + prevention:

<https://owasp.org/www-community/attacks/Brute_force_attack>

<https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks>

Account lockout implementation

<https://dotnettutorials.net/lesson/how-to-lockout-a-user-account-in-asp-net-identity/>

ReCAPTCHA v3 docs

<https://developers.google.com/recaptcha/docs/v3>

OWASP session hijacking

<https://owasp.org/www-community/attacks/Session_hijacking_attack>

session hijacking prevention

<https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/how-to-prevent-session-hijacking-attacks/>

OWASP HttpOnly

<https://owasp.org/www-community/HttpOnly>

OWASP Testing for weak password policies

<https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/07-Testing_for_Weak_Password_Policy>

Stack overflow regex for password

<https://stackoverflow.com/questions/19605150/regex-for-password-must-contain-at-least-eight-characters-at-least-one-number-a>

PCI Security standards doc

<https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf>

OWASP sensitive data protection

<https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere>

Storing CC information guide

<https://gocardless.com/guides/posts/how-to-store-credit-card-information/>

OWASP automation threats

[https://owasp.org/www-project-automated-threats-to-web applications/assets/oats/EN/OAT-010\_Card\_Cracking](https://owasp.org/www-project-automated-threats-to-web%20applications/assets/oats/EN/OAT-010_Card_Cracking)

malware through injections

<https://securityintelligence.com/posts/e-commerce-skimming-the-new-pos-malware/>