

# SWEN.755 - Software Architecture

## Assignment 5: Secure Session Management Testing

Avinash Amudala| Sam Singh Anantha| Haotian Zhang

<b>Architecture Breakers:</b> .....	<b>2</b>
1. Insecure Storage of Session Identifiers.....	2
2. Susceptibility to Session Fixation Attacks.....	2
3. Session Hijacking.....	3
4. Improper Session Expiration.....	3
<b>Test Cases:</b> .....	<b>3</b>
1. Test for Session Expiration (addresses Improper Session Expiration) [PASS].....	4
2. Test for Session ID Change After Login (addresses Susceptibility to Session Fixation Attacks) [PASS].....	4
3. Test for Insecure Session Cookie Transmission [FAIL].....	5
4. Test for Insecure Session Cookie Transmission [FAIL].....	6

# Architecture Breakers:

## 1. Insecure Storage of Session Identifiers

**Explanation:** When session identifiers (like cookies) are stored insecurely, they are susceptible to being intercepted and used by unauthorized parties. This can happen if the session data is not encrypted or if it's transmitted over an unsecured connection (like HTTP instead of HTTPS).

**Problem:** If attackers gain access to these identifiers, they can impersonate legitimate users, potentially gaining unauthorized access to sensitive data and functionalities.

**Mitigation:** Always transmit session identifiers over secured connections (HTTPS). Ensure that the cookies are set with security flags like `HttpOnly` and `Secure`. Consider implementing additional layers of security like encryption for session data.

## 2. Susceptibility to Session Fixation Attacks

**Explanation:** Session fixation occurs when an attacker is able to fixate (set or predict) the session ID of another user's session. For example, by tricking a user into authenticating with a session ID known to the attacker.

**Problem:** Once the user authenticates, the attacker can use the predetermined session ID to hijack the session.

**Mitigation:** Generate a new session ID with every new session and especially after a user logs in. Ensure that session IDs are not predictable and are invalidated after logout.

### 3. Session Hijacking

**Explanation:** Session hijacking involves an attacker taking over a user's session, typically after the user has authenticated. This can be done by stealing the session cookie through methods like Cross-Site Scripting (XSS) attacks or sniffing unencrypted network traffic.

**Problem:** An attacker can gain unauthorized access to the application, performing actions as if they were the legitimate user.

**Mitigation:** Implement HTTPS to secure data in transit. Use the `HttpOnly` flag on cookies to prevent access via client-side scripts. Educate users about secure browsing practices. Consider adding additional checks, like validating the user's IP address or user-agent string, although these can have limitations.

### 4. Improper Session Expiration

**Explanation:** If a session does not expire after a user logs out or after an adequate period of inactivity, the session remains active. This can be a problem, especially if the user is using a public or shared computer.

**Problem:** Anyone with access to the user's computer could use the still-active session to access the application as the user.

**Mitigation:** Implement automatic expiration of sessions after a period of inactivity and instantly destroy sessions upon user logout. Set a reasonable timeout for sessions, considering the context of your application (e.g., shorter for banking apps).

## Test Cases:

### 1. Test for Session Expiration (addresses Improper Session Expiration) [PASS]

**Description:** This test checks whether a user session expires after a set period of inactivity, which is crucial for preventing unauthorized access to an active session left by a user.

**Test Implementation:**

- Log in to the application to start a session.
- Wait for a period longer than the session timeout period (e.g., 70 seconds).
- Attempt to access a protected route that requires an active session.
- Verify that the request is denied due to session expiration (expecting a 403 Forbidden response).

**How It Addresses the Breaker:** This test ensures that sessions expire after a certain period, mitigating the risks associated with improper session expiration. If the session does not expire as expected, it indicates a vulnerability where sessions remain active longer than they should, posing a security risk.

### 2. Test for Session ID Change After Login (addresses Susceptibility to Session Fixation Attacks) [PASS]

**Description:** This test ensures that a new session ID is issued upon user login, which is a key defense against session fixation attacks.

**Test Implementation:**

- Initiate a session and record the session ID.
- Perform a login operation.
- Check that a new session ID is issued post-login.
- Verify that the new session ID is different from the initial one.

**How It Addresses the Breaker:** By ensuring that the session ID changes upon login, this test verifies that the application is not vulnerable to session fixation attacks, where an attacker might set a known session ID before the user logs in. The change of session ID upon authentication is a critical step in preventing such attacks.

### 3. Test for Insecure Session Cookie Transmission **[FAIL]**

**Objective:** This test aims to verify the security of session cookies transmitted by the application, focusing on the Secure flag. The absence of the flags can significantly increase the risk of security breaches.

**Description:**

- The Secure flag ensures that cookies are only sent over secure HTTPS connections, protecting them from being intercepted over unsecured HTTP connections. This is crucial for preventing session hijacking attacks where an attacker could capture unencrypted session cookies.

**Test Implementation:**

- Initiate a session by making a GET request to a common application route (such as the homepage).
- Examine the response to ensure the presence of the Set-Cookie header.
- Analyze the session cookie to verify the inclusion of the Secure attribute.

**Expected Outcome:**

- The session cookie should include the Secure attribute, indicating it will only be transmitted over secure (HTTPS) connections.

**How It Addresses the Breaker:**

- Without the Secure flag, session cookies are at risk of being intercepted during transmission over unsecured connections (like HTTP), leading to potential session hijacking.

## 4. Test for Insecure Session Cookie Transmission **[FAIL]**

**Objective:** This test aims to verify the risk of session hijacking by checking the HttpOnly flag. The absence of the flag can significantly increase the risk.

**Description:**

- The HttpOnly flag prevents access to cookies via client-side scripts, safeguarding against Cross-Site Scripting (XSS) attacks that attempt to steal session cookies.

**Test Implementation:**

- Initiate a session by making a GET request to a common application route (such as the homepage).
- Examine the response to ensure the presence of the Set-Cookie header.
- Analyze the session cookie to verify the inclusion of HttpOnly attribute.

**Expected Outcome:**

- The session cookie should contain the HttpOnly attribute, preventing access to the cookie via client-side scripts.

**How It Addresses the Breaker:**

- If the HttpOnly flag is absent, the session cookie becomes accessible to client-side scripts, increasing the vulnerability to XSS attacks where attackers can steal session cookies.