Client Side Security

Client Side Security

- Today's web applications are complex,
  - often made up of a mix of existing software, open-source and third-party code, and
  - custom JavaScript and HTML all integrated via application program interfaces (APIs).

- While web applications are hosted and maintained on an organization's server, they actually run on an end user's browser.

Client Side Security

- The scripts that run the applications are referred to as ‘client-side scripts.’

- These scripts create an incredibly dynamic environment that enable a high level of functionality,

- but also facilitate tremendous risk since the combination of potentially flawed or vulnerable systems, servers, codes, and applications creates the perfect scenario for threat actors to leverage in client-side attacks

client-side attacks

- The Open Web Application Security Project® (OWASP) lists 12 client-side security risks that organizations need to ensure they've mitigated to prevent attacks:

Document Object Model (DOM)-based Cross-site Scripting

- Sometimes also called just ‘cross-site scripting’ or ‘XSS’,

- this is a vulnerability that affects websites and enables an attacker to inject their own malicious code onto the HTML pages displayed to users.

- If the malicious code is executed by the victim’s browser, the code performs actions, such as stealing credit card information or sensitive credentials

JavaScript Injection

- This type of vulnerability is considered a subtype of XSS
- involving the injection of malicious JavaScript code executed by the end user's browser application
- JavaScript injunctions can be used
  - to modify the content seen by the end user,
  - to steal the user's session cookies, or
  - to impersonate the user

HTML Injection

- Another type of cross-site scripting attack, an HTML injection involves injecting HTML code via vulnerable sections of the website

- Usually, the purpose of the HTML injection is to change the website's design or information displayed on the website

Client-side URL Redirection

- In this type of attack, an application accepts untrusted input that contains a URL value

- that causes the web application to redirect the user to another, likely malicious page controlled by the attacker.

Cascading Style Sheets (CSS) Injection

- Attackers inject arbitrary CSS code into a website, which is then rendered in the end user's browser.

- Depending on the type of CSS payload,
  - the attack could lead to XSS,
  - user interface (UI) modifications or
  - the exfiltration of sensitive information, like credit card data

Client-side Resource Manipulation

- This type of vulnerability enables the threat actor to control the URL that links to other resources on the web page, thus enabling cross-site scripting attacks.

Cross-origin Resource Sharing (CORS)

- Poorly configured CORS policies can facilitate cross-origin attacks like cross-site request forgery (CSRF)

Cross-site Flashing

- Because Flash applications are often embedded in browsers, flaws or vulnerabilities in the Flash application could enable cross-site scripting attacks

Clickjacking or UI Redress Attack

- This type of attack involves a threat actor using multiple web page frame layers to trick a user into clicking a button or link on a different page from the one intended

- Keystrokes can also be hijacked using this technique.

- By using style sheets, i frames, and text boxes, a threat actor can trick the user into thinking they're entering login credentials or bank account information into a legitimate website, when, in fact, they are actually typing into a frame controlled by the attacker

Web Messaging

- Also called cross-document messaging,
- web messaging enables applications running on different domains to communicate securely
- If the receiving domain is not configured, problems could arise related to redirection or the website leaking sensitive information to unknown or malicious servers

Local Storage

- Sometimes called web storage or offline storage, local storage enables JavaScript sites and apps to store and access the data without any expiration date
-
- Thus, data stored in the browser will be available even after closing the browser window.

- Since the storage can be read using JavaScript, a cross-site scripting attack could extract all the data from the storage.

- Malicious data could also be loaded via JavaScript

Countermeasures:

- Install antivirus software, anti-spyware software, and firewall protection on all workstations, servers, and wireless devices

- Ensure the latest system software patches are applied regularly.

- Maintain a complete backup system of all data on all systems use a separate server or external hard drive or network location to store backups that are no longer needed and keep them off the system they were created on.

Countermeasures:

- If a user is logging in from an unknown location or IP address, consider blocking access from those locations (access control lists).

- Prevent unauthorized access to accounts.

- Use strong passwords and avoid common passwords or patterns that can lead to vulnerabilities

- Limit login attempts (user lockout)