**1. Same Origin Policy (SOP)**

The Same Origin Policy (SOP) is a web security mechanism implemented by browsers to restrict how documents or scripts from one origin can interact with resources from another. The "origin" is defined by the combination of domain, protocol, and port.

**Purpose**: SOP prevents malicious websites from accessing sensitive data on other websites. For example, a page from https://example.com cannot access cookies, DOM elements, or AJAX responses from https://bank.com.

**Importance**: SOP is essential to protect against attacks like Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF).

**Exceptions**: Legitimate cross-origin interactions can occur through mechanisms like Cross-Origin Resource Sharing (CORS), which explicitly allows controlled access between origins.

**Example**: If a user is logged into a banking website, SOP ensures that a malicious webpage cannot read their banking session data.

SOP forms the foundation of browser security by isolating websites, ensuring that sensitive information remains secure within its origin.

**2. Cross-Site Scripting (XSS) Attack**

Cross-Site Scripting (XSS) is a security vulnerability where attackers inject malicious scripts into trusted websites. These scripts execute in the victim’s browser, leading to theft of sensitive data, session hijacking, or redirection to malicious websites.

**Types**:

1. **Stored XSS**: Malicious scripts are permanently stored on the target server (e.g., in a database).
2. **Reflected XSS**: The payload is part of the request and reflected in the response, executed when a victim clicks on a link.
3. **DOM-Based XSS**: Scripts are executed due to vulnerabilities in the website's client-side scripts.

**Example**: A comment section on a website might allow the input of <script>alert('Hacked')</script>. If not sanitized, the script is executed when another user visits the page.

**Mitigation**: Input validation, output encoding, and using Content Security Policies (CSP) are effective defenses against XSS.

**3. Cross-Site Request Forgery (CSRF) Attack**

CSRF is a type of malicious exploit where an attacker tricks a user into submitting an unwanted action on a website or web application that the user is authenticated to.

**Mechanism**:

* The attacker embeds a malicious request in an email or third-party website.
* When the victim clicks the link or loads the malicious page, the browser sends the request using the victim’s session cookies.
* Example: A user logged into their bank is tricked into clicking a link that transfers funds to the attacker’s account.

**Impact**: Unauthorized actions like fund transfers, password changes, or email modifications can be performed.

**Mitigation**:

1. Anti-CSRF tokens: Unique tokens for each session to validate legitimate requests.
2. SameSite cookies: Prevent cookies from being sent with cross-site requests.
3. Referer header validation: Ensure requests come from trusted sources.

**4. SQL Injection Attack**

SQL Injection is a critical vulnerability where attackers execute malicious SQL queries by exploiting unsanitized inputs in web applications.

**Mechanism**: SQL commands are injected into input fields to manipulate the backend database.  
**Example**:

A screenshot of a computer

Description automatically generated

This bypasses authentication.

**Impact**:

1. Access to sensitive data.
2. Deletion or modification of database contents.
3. Potential control over the database server.

**Mitigation**:

1. Use parameterized queries and prepared statements.
2. Implement strict input validation.
3. Limit database privileges and implement robust error handling.

**5. Clickjacking Attack**

Clickjacking occurs when an attacker overlays an invisible iframe containing a legitimate webpage over a fake interface. When users interact with the visible interface, they unknowingly interact with the hidden iframe.

**Impact**:

1. **Stealing user actions**: Clicking a "Like" button, subscribing to a service, or making financial transactions.
2. **Unauthorized permissions**: Activating the webcam or microphone without consent.

**Example**: A user is tricked into clicking a button that appears to close a pop-up but actually triggers a payment action in the hidden iframe.

**Mitigation**:

1. **X-Frame-Options Header**: Prevents embedding the website in iframes.
2. **Frame-busting scripts**: Detect and block iframes attempting to overlay the site.
3. Content Security Policy (CSP): Restricts frame origins to trusted sources.

**6. Content Security Policies (CSP)**

Content Security Policy (CSP) is a security feature that helps prevent XSS, clickjacking, and other attacks by specifying which resources (scripts, styles, images) are allowed to load and execute.

**How it Works**: CSP is implemented via HTTP headers or meta tags.

A screenshot of a computer

Description automatically generated**Benefits**:

1. Blocks unauthorized scripts.
2. Reduces attack surface for XSS.
3. Prevents data injection attacks.

**Challenges**: Misconfigured CSPs can reduce functionality or leave gaps in protection.

**7. Web Tracking**

Web tracking refers to the collection of data about a user’s online activity, including visited websites, clicks, and preferences.

**Techniques**:

1. **Cookies**: Store user data to track behavior across sessions.
2. **Browser Fingerprinting**: Identifies users based on unique browser settings and plugins.
3. **Tracking Pixels**: Invisible images that report user activity.

**Uses**:

1. Personalized advertisements.
2. Website analytics.
3. User experience improvement.

**Concerns**: Web tracking raises privacy issues, as users may not consent to data collection. Tools like ad blockers, privacy-focused browsers, and legislation like GDPR mitigate tracking.

**8. Session Management and User Authentication**

**Session Management** involves securely managing user sessions by assigning unique session identifiers (session IDs). These IDs track users' interactions with a web application.  
**User Authentication** verifies user identity using credentials like passwords, tokens, or biometrics.

**Best Practices**:

1. Securely store session IDs (e.g., in HttpOnly cookies).
2. Use HTTPS to encrypt session data in transit.
3. Implement Multi-Factor Authentication (MFA).

**Common Attacks**:

1. Session Hijacking: Stealing session IDs to impersonate a user.
2. Session Fixation: Forcing a user to use a known session ID.

**9. HTTPS**

HTTPS (HyperText Transfer Protocol Secure) encrypts communication between the browser and server using SSL/TLS protocols.

**Advantages**:

1. **Encryption**: Prevents data theft during transmission.
2. **Authentication**: Verifies that the website is genuine via SSL certificates.
3. **Integrity**: Ensures data has not been tampered with during transit.

**Implementation**: Websites obtain certificates from trusted Certificate Authorities (CAs). Modern browsers warn users about non-HTTPS websites to improve security awareness.

**SSL (Secure Sockets Layer)** and **TLS (Transport Layer Security)** are cryptographic protocols designed to secure communication over a network. They provide **encryption, authentication, and data integrity** to ensure that sensitive information, such as login credentials, payment details, and personal data, is transmitted securely.

**10. Threat Modeling**

Threat modeling is a structured approach to identifying, analyzing, and mitigating security threats during the development of a system.

**Process**:

1. **Identify Assets**: Determine valuable resources (e.g., sensitive data).
2. **Identify Threats**: Recognize potential risks (e.g., SQL Injection, XSS).
3. **Analyze Vulnerabilities**: Assess weak points in the system.
4. **Mitigate Threats**: Develop countermeasures to address vulnerabilities.
5. **Prioritize Risks**: Focus on threats with high likelihood and impact.

**Outcome**: By proactively identifying and addressing risks, threat modeling ensures a secure system design.

UNIT 4

**1. Android vs. iOS Security Model**

**Android and iOS** are two leading mobile operating systems, each with distinct security models.

1. **Android Security Model**:
   * Open-source, providing flexibility but increasing vulnerability to attacks.
   * Relies on Google Play Protect for app scanning and malware detection.
   * App permissions allow user control over data access.
   * Uses sandboxing to isolate apps.
2. **iOS Security Model**:
   * Closed-source, with a tightly controlled app ecosystem.
   * Strong code signing: Apps must be reviewed and approved by Apple.
   * Hardware-backed security: Secure Enclave protects sensitive data.
   * Frequent updates for security patching.

**Comparison**:

* Android is more vulnerable to malware due to third-party app stores.
* iOS emphasizes strict control, making it harder for malicious apps to infiltrate.

**Example**: Android users are more prone to malicious apps due to APK sideloading, while iOS users rely heavily on App Store reviews.

**2. Threat Models**

**Threat modeling** identifies, assesses, and mitigates potential security threats in a system.

**Steps**:

1. **Identify Assets**: Determine critical resources (e.g., user data, app functionality).
2. **Threat Analysis**: Analyze possible attacks, such as data breaches or malware.
3. **Vulnerability Assessment**: Identify weak points in the system.
4. **Mitigation**: Implement security measures (e.g., encryption, access controls).

**Types of Threats**:

* External threats: Hackers, malware, phishing.
* Internal threats: Insider attacks or negligence.

**Example**: A banking app may model threats like unauthorized transactions or session hijacking, leading to stronger authentication mechanisms.

**3. Information Tracking**

**Information tracking** refers to collecting and analyzing user data, often for personalized services or advertising.

**Mechanisms**:

1. Cookies and tracking pixels monitor browsing behavior.
2. Mobile apps collect GPS data and preferences.
3. Device fingerprinting tracks users without cookies.

**Uses**:

* Personalized ads, app improvements, and fraud detection.

**Concerns**:

* Privacy invasion, data misuse, and identity theft.

**Example**: Social media apps track user preferences to deliver targeted ads. GDPR and CCPA regulate data collection practices.

**4. Rootkits**

A **rootkit** is a malicious software tool that gains unauthorized access to a system while remaining hidden.

**Types**:

1. **User-mode rootkits**: Modify application-level processes.
2. **Kernel-mode rootkits**: Attack the OS kernel for deeper control.
3. **Firmware rootkits**: Target device firmware for persistence.

**Impact**:

* Keystroke logging, data theft, and complete system compromise.

**Mitigation**:

* Regular updates, antivirus software, and hardware-based security measures.

**Example**: Stuxnet, a famous rootkit, was used for industrial sabotage.

**5. Access Control in Android Operating System**

**Access control** in Android regulates how apps and users interact with system resources.

**Mechanisms**:

1. **Permissions Model**: Apps request user consent to access features (e.g., camera, location).
2. **Sandboxing**: Isolates apps to prevent unauthorized interactions.
3. **Encryption**: Protects sensitive data at rest and in transit.

**Improvements**: Android’s updated permissions system in recent versions asks users for permissions at runtime.

**Example**: A social media app requesting microphone access is subject to user approval.

**6. Rooting Android Devices**

**Rooting** gives users administrative (root) privileges on Android devices.

**Advantages**:

1. Customization: Install custom ROMs or themes.
2. Control: Modify system files or remove bloatware.

**Risks**:

1. Security vulnerabilities: Malware gains unrestricted access.
2. Voided warranty: Rooting often breaches warranty agreements.
3. Bricking: Errors during rooting may render the device inoperable.

**Example**: A user may root their device to remove pre-installed apps but risks exposing sensitive data to malicious apps.

**7. Repackaging Attacks**

**Repackaging attacks** involve modifying legitimate apps to include malicious code and redistributing them.

**Mechanism**:

1. A hacker decompiles an app, injects malware, and re-signs it.
2. The modified app is distributed via unofficial app stores.

**Impact**:

* Data theft, unauthorized transactions, or spreading ransomware.

**Mitigation**:

1. Use anti-tampering measures like code obfuscation.
2. Verify app sources before installation.

**Example**: Fake versions of popular banking apps are often repackaged to steal login credentials.

**8. Attacks on Apps**

Apps are vulnerable to several attacks, including:

1. **Injection attacks**: SQL or command injection exploits input fields.
2. **Data leakage**: Poor coding practices expose sensitive data.
3. **Reverse engineering**: Hackers decompile apps to extract code and credentials.

**Mitigation**:

1. Use secure coding practices.
2. Employ runtime protection tools like app shielding.

**Example**: An e-commerce app’s failure to encrypt payment data could lead to credit card theft.

**9. Whole-Disk Encryption**

**Whole-disk encryption (WDE)** secures all data on a disk by encrypting it at the hardware or software level.

**How it Works**:

* Encryption keys are needed to decrypt and access the data.

**Benefits**:

1. Protects sensitive data if a device is lost or stolen.
2. Prevents unauthorized access to disk contents.

**Example**: Android devices use File-Based Encryption (FBE) to isolate encrypted files for user accounts.

**Drawbacks**:

* Increased resource usage may reduce performance.

**10. Hardware Protection**

**Hardware protection** involves embedding security mechanisms in devices to defend against attacks.

**Examples**:

1. **Secure Enclave (iOS)**: Stores sensitive information like biometric data.
2. **Trusted Platform Module (TPM)**: Provides secure cryptographic functions.

**Impact**: Strengthens defenses against tampering, malware, and physical attacks.

**Example**: Biometric data stored in the Secure Enclave cannot be accessed by apps or the OS.

**11. Viruses, Spyware, and Keyloggers**

**Viruses**: Malicious programs that replicate and spread across systems. They corrupt files or disrupt operations.  
**Spyware**: Tracks user activity, collecting data like passwords or browsing history.  
**Keyloggers**: Record keystrokes to capture sensitive information like login credentials.

**Mitigation**:

1. Use antivirus software.
2. Regularly update devices.
3. Avoid suspicious downloads or emails.

**12. Malware Detection**

**Malware detection** involves identifying and mitigating malicious software.

**Techniques**:

1. **Signature-based**: Matches known malware signatures.
2. **Behavior-based**: Detects suspicious activities, like unauthorized file access.
3. **AI-based**: Machine learning models predict and block unknown threats.

**Example**: Google Play Protect scans apps for malware before installation.

UNIT 5

**1. Meltdown Attack**

**Meltdown** is a vulnerability in CPUs that allows unauthorized access to memory.

**How It Works**:

* Exploits out-of-order execution in modern processors.
* Bypasses memory isolation and accesses sensitive data, such as passwords or encryption keys.

**Impact**:

* Affects Intel, ARM, and some AMD processors.
* Steals sensitive data from other processes running on the same system.

**Mitigation**:

* Software patches to isolate kernel memory.
* Hardware design changes in newer processors.

**Example**: In a cloud environment, one virtual machine could potentially access data from another.

**2. Spectre Attack**

**Spectre** exploits speculative execution in processors to trick programs into accessing arbitrary memory.

**How It Works**:

* Misdirects branch prediction to access out-of-bound memory.
* Leaks data via side channels.

**Impact**:

* Affects nearly all modern processors (Intel, AMD, ARM).
* Difficult to mitigate entirely due to its reliance on hardware features.

**Mitigation**:

* Software updates and compiler-level changes.
* Use of barriers like Retpoline to restrict speculative execution.

**Example**: Malicious JavaScript in a web browser could exploit Spectre to steal sensitive user data.

**3. Authentication and Password**

**Authentication** verifies user identities to secure systems. Passwords are the most common method.

**Key Concepts**:

1. **Strong Passwords**: Require complexity (e.g., length, special characters).
2. **Multi-Factor Authentication (MFA)**: Adds layers (e.g., SMS or biometrics).
3. **Password Management**: Use tools to store and generate secure passwords.

**Weaknesses**:

* Vulnerable to brute force, phishing, and dictionary attacks.

**Mitigation**:

1. Use hashing algorithms (e.g., bcrypt, PBKDF2) for storing passwords.
2. Enforce password policies (expiration, complexity).

**Example**: A phishing attack could trick users into revealing credentials.

**4. Access Control Concept**

**Access control** regulates who or what can access resources in a system.

**Types**:

1. **Mandatory Access Control (MAC)**: Predefined rules enforce restrictions.
2. **Discretionary Access Control (DAC)**: Owners decide permissions.
3. **Role-Based Access Control (RBAC)**: Permissions based on roles (e.g., admin, user).

**Components**:

1. **Authentication**: Verifying identity.
2. **Authorization**: Granting or denying access.

**Example**: An organization may use RBAC to limit database access to specific employees.

**5. Access Control List (ACL)**

**ACL** defines permissions for objects (files, directories, etc.).

**Structure**:

* Lists users or groups and their allowed actions (read, write, execute).

**Types**:

1. **File System ACLs**: Control file access.
2. **Network ACLs**: Manage traffic in firewalls or routers.

**Example**: A file might have an ACL specifying that only "Admin" can modify it, while "Guest" can only view it.

**6. Capability**

**Capabilities** are tokens or keys granting specific rights to users or processes.

**Key Features**:

* Fine-grained control over access.
* Only necessary rights are granted (least privilege).

**Example**: A capability token might allow a user to edit a file but not delete it.

**Difference from ACLs**: Capabilities are associated with the entity, not the object.

**7. Sandboxing**

**Sandboxing** isolates applications or processes to prevent malicious actions.

**Use Cases**:

1. Secure execution of untrusted code.
2. Contain malware in virtual environments.

**Examples**:

* Web browsers sandbox tabs to prevent cross-site attacks.
* Mobile OSes isolate apps to limit access to system resources.

**Mitigation**: Limits the damage from vulnerabilities by containing the attack surface.

**8. Threats of Hardware Trojans and Supply Chain Security**

**Hardware Trojans** are malicious modifications to hardware components.

**How They Work**:

* Introduced during manufacturing or design phases.
* Triggered to leak data, sabotage operations, or weaken defenses.

**Supply Chain Security**: Ensures trust in the hardware/software procurement process.

**Mitigation**:

1. Use verified suppliers.
2. Employ hardware verification techniques.

**Example**: A backdoor in a network chip could allow attackers remote access.

**9. Side-Channel Analysis-Based Threats and Attacks**

**Side-channel attacks** exploit information leaks from hardware.

**Types**:

1. **Timing Attacks**: Infer data from processing times.
2. **Power Analysis**: Analyze power consumption patterns.
3. **Electromagnetic Attacks**: Exploit EM emissions.

**Example**: An attacker could extract cryptographic keys by observing power consumption during encryption.

**Mitigation**:

* Randomize operations.
* Add noise to signals.

**10. Issues in Critical Infrastructure and SCADA Security**

**SCADA (Supervisory Control and Data Acquisition)** systems manage industrial processes (e.g., power grids, water systems).

**Challenges**:

1. Legacy systems with weak security.
2. Remote access introduces vulnerabilities.
3. Attacks like Stuxnet target SCADA for sabotage.

**Mitigation**:

1. Network segmentation.
2. Regular updates and patching.
3. Intrusion detection systems for SCADA networks.

**Example**: The 2015 Ukraine power grid attack disrupted electricity distribution using SCADA vulnerabilities.