**TASK-4**

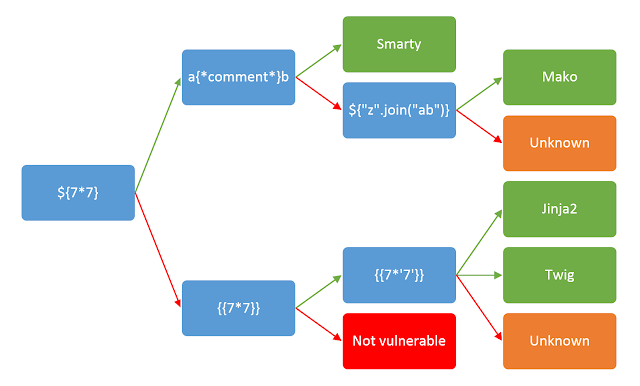
**Understanding of Web Application Vulnerabilities**

**(Other than Top 10 OWASP Vulnerabilities)**

**What is Web Application Vulnerability?**

* Web application vulnerabilities involve a system flaw or weakness in a web-based application.They have been around for years, largely due to not validating or sanitizing form inputs, misconfigured web servers, and application design flaws, and they can be exploited to compromise the application’s security.
* These [vulnerabilities](https://www.rapid7.com/fundamentals/vulnerabilities-exploits-threats/) are not the same as other common types of vulnerabilities, such as network or asset. They arise because web applications need to interact with multiple users across multiple networks, and that level of accessibility is easily taken advantage of by hackers.

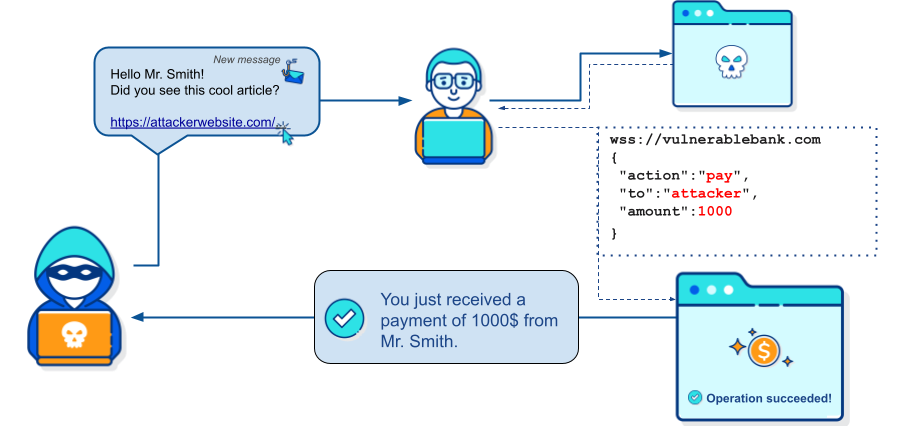
**1)Server-Side Template Injection**



Server-Side Template Injection is a vulnerability that arises when an application allows user-supplied input to be directly embedded within server-side templates. Exploiting this vulnerability involves injecting malicious code or expressions into template variables, which are then executed within the template rendering process. This can lead to unauthorized access to sensitive data, remote code execution, or complete server compromise, depending on the context and severity of the vulnerability. The business impact of a successful SSTI attack can range from data breaches to full server takeover, resulting in reputation damage, financial losses, and legal liabilities.

**Mitigation**:To mitigate SSTI vulnerabilities, developers should validate and sanitize user inputs before embedding them in templates. Employ safe templating engines that automatically escape user inputs and prevent code execution. Implement a strict content security policy (CSP) to limit the sources of templates and avoid dynamic generation of templates from user inputs. Regular security audits and penetration testing can help identify and address SSTI vulnerabilities before they are exploited in the wild. Keeping software and libraries up to date also reduces the risk of known vulnerabilities being exploited.

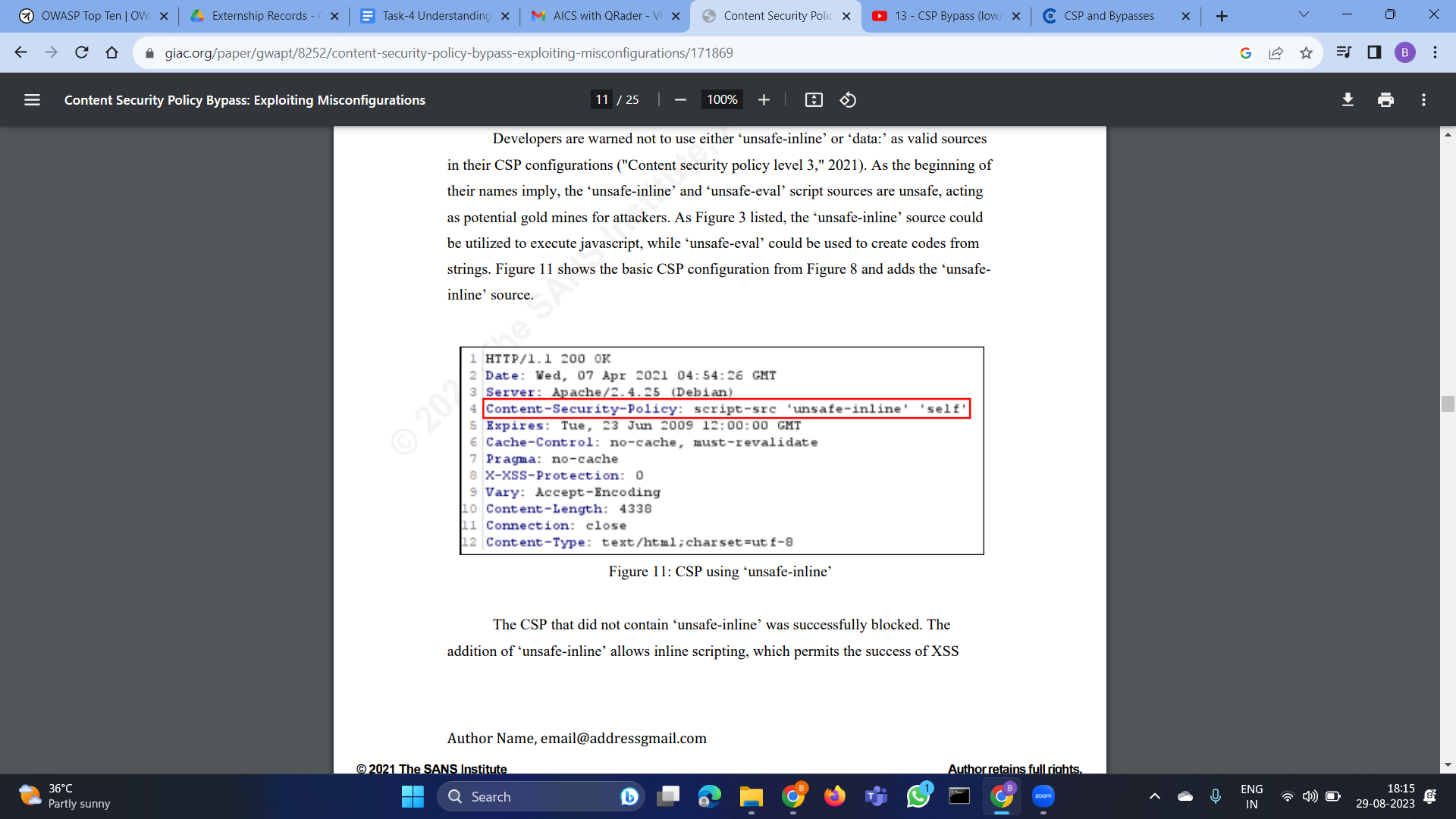
**2)Cross-site WebSocket hijacking**

****

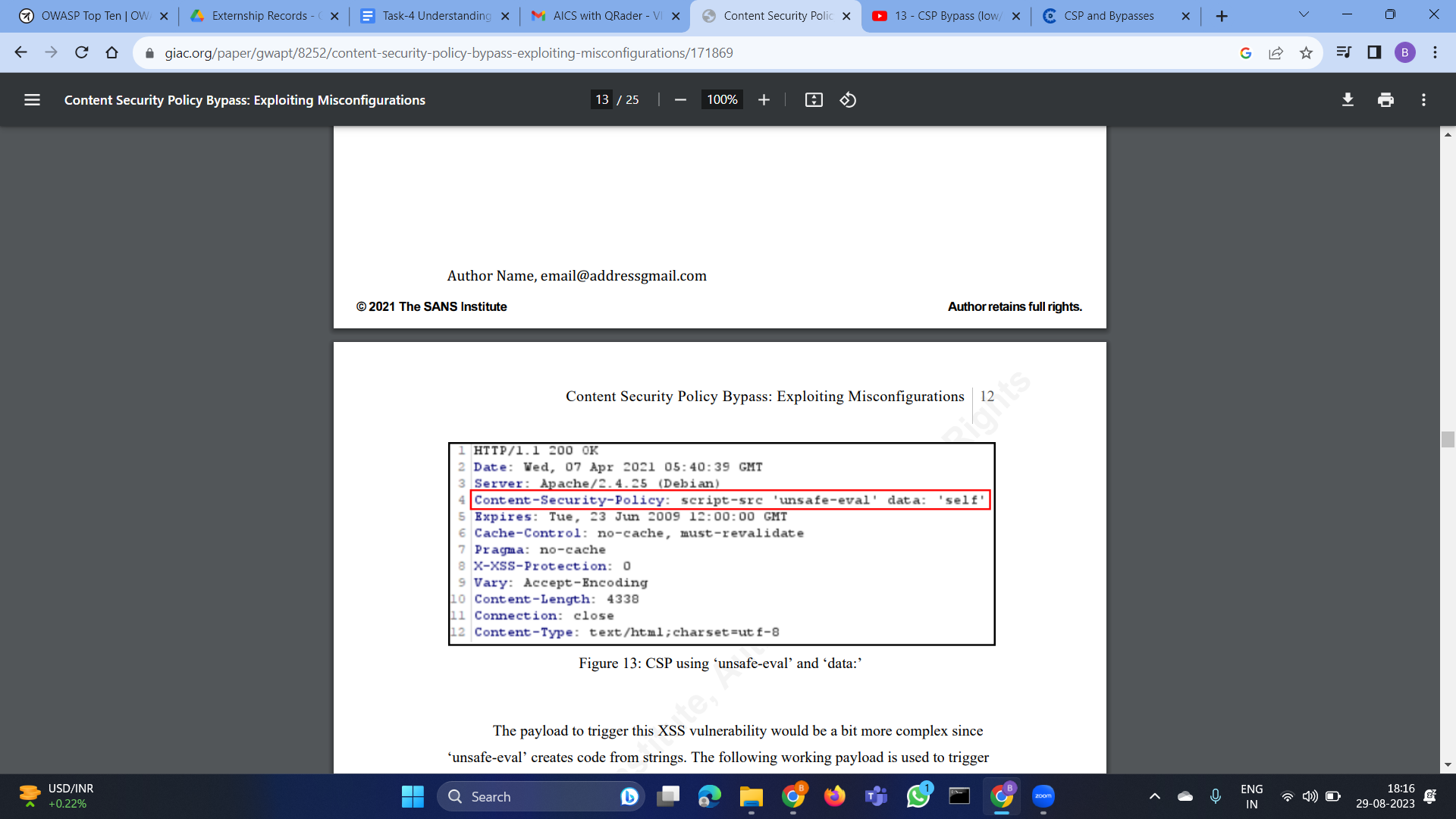
Cross-Site WebSocket Hijacking occurs when an attacker tricks a user into connecting to a malicious WebSocket server, exploiting browser trust to execute unauthorized actions. By sending crafted WebSocket requests, attackers can potentially manipulate user data or perform actions on their behalf. This can lead to data breaches and unauthorized operations, affecting user privacy and system integrity.

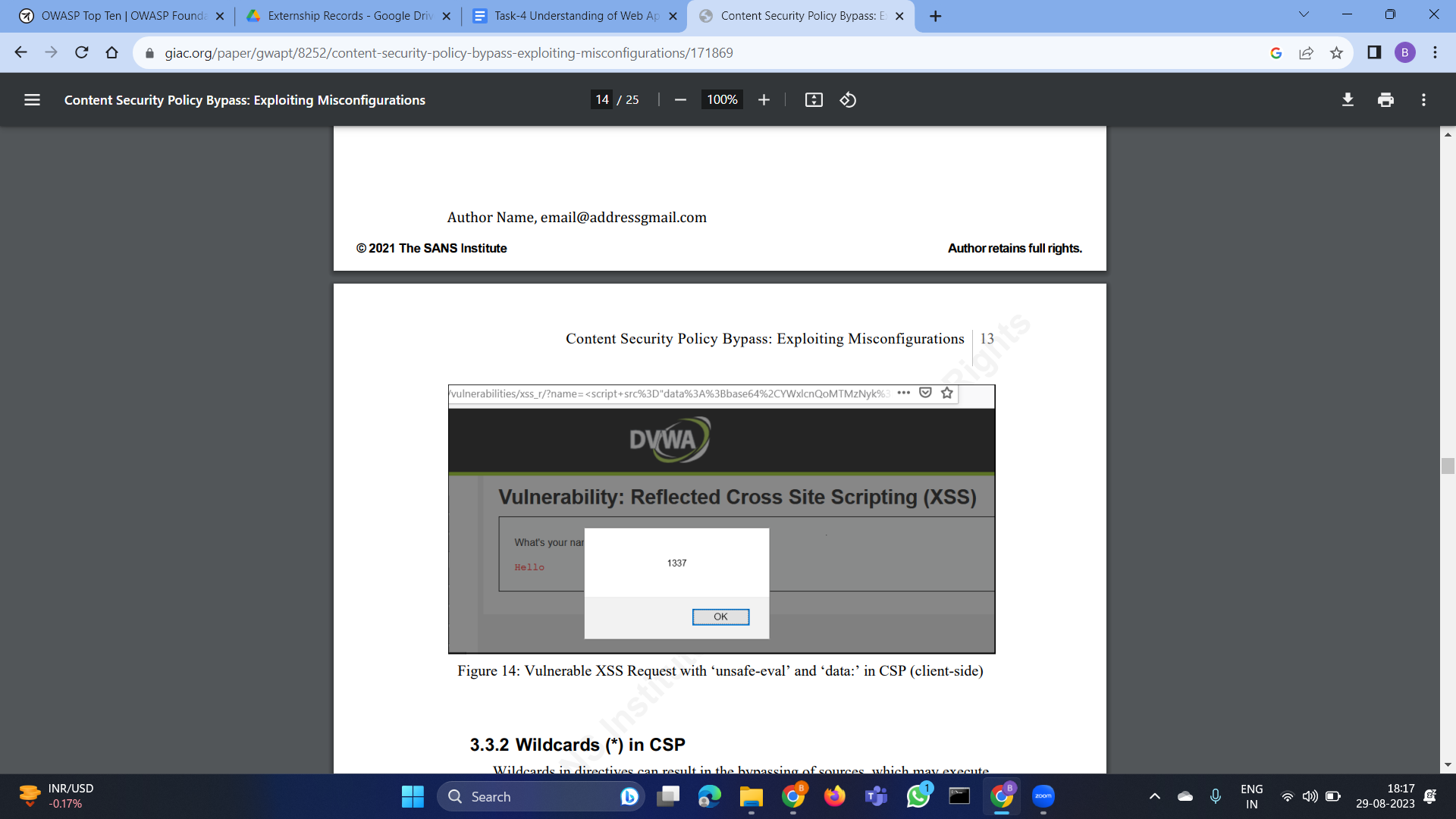
**Mitigation:** Use secure authentication and authorization mechanisms to control WebSocket connections. Validate WebSocket requests and implement same-origin policies to prevent cross-site attacks.

**3)Content-Security-Policy (CSP) Bypass**

****

****

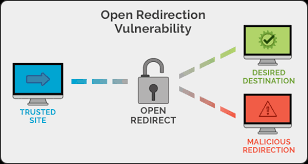
****

****

Content-Security-Policy Bypass involves evading the CSP protection to inject malicious scripts into a website. Attackers find ways to bypass CSP restrictions and execute arbitrary code, potentially leading to cross-site scripting (XSS) attacks and data leakage.

**Mitigation:** Regularly update and tighten the CSP rules to minimize potential bypasses. Utilize proper input validation and sanitize user-generated content to thwart XSS attacks.

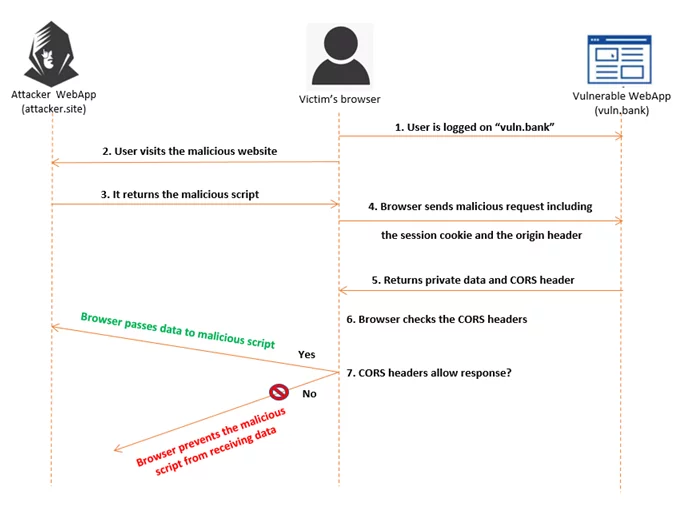
**4)Unvalidated Redirects And Forwards**



Unvalidated Redirects and Forwards occur when an application forwards users to a URL without proper validation. Attackers manipulate these redirects to send users to malicious websites, phishing sites, or other harmful locations.

**Mitigation:** Validate and sanitize all redirect URLs, and avoid using user-supplied inputs in redirects. Implement safe URL redirection mechanisms to prevent open redirects.

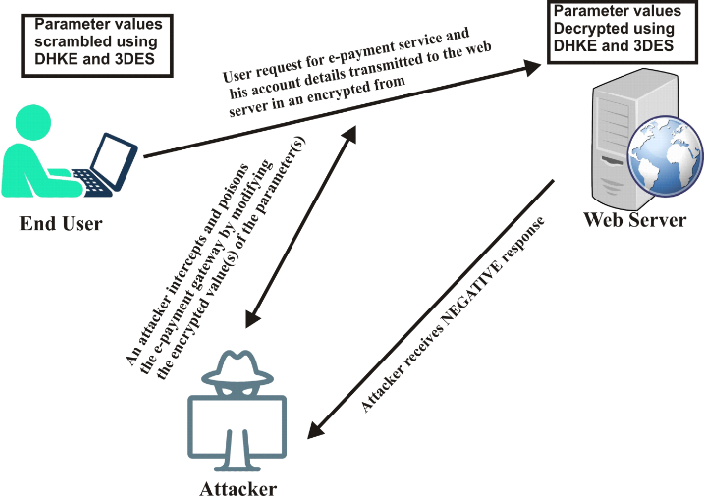
**5)Cross-Origin Resource Sharing Misconfiguration**

****

Cross-Origin Resource Sharing (CORS) Misconfiguration happens when a web application improperly configured its CORS policy, allowing unintended cross-origin requests. Attackers can exploit this to perform unauthorized actions on behalf of users.

**Mitigation:**Properly configure CORS settings to restrict cross-origin requests. Avoid using wildcard origins and use strict access controls to limit resource exposure.

**6)HTTP Parameter Pollution (HPP)**

****

HTTP Parameter Pollution occurs when an attacker manipulates URL parameters to confuse an application's processing logic, potentially leading to unauthorized data exposure or incorrect actions.

**Mitigation:** Validate and sanitize input parameters to prevent parameter pollution. Implement strict input validation and parsing mechanisms to avoid confusion.

**7)Cross-Site Request Forgery (CSRF) with JSON**

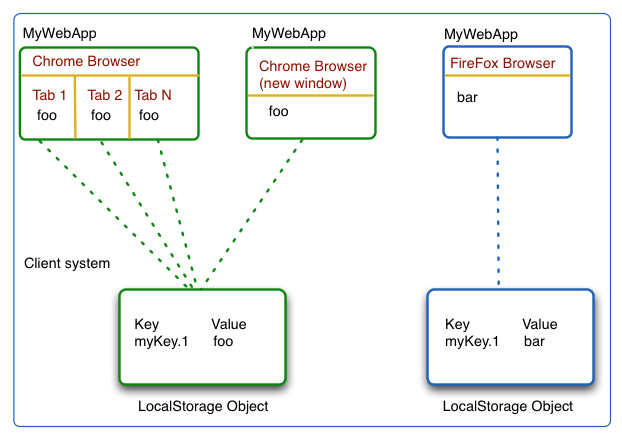
Cross-Site Request Forgery with JSON is a variation of CSRF where attackers manipulate JSON data to perform unauthorized actions on a user's behalf. By tricking users into executing malicious JSON requests, attackers can compromise user accounts and perform unintended actions.



**Mitigation:** Apply anti-CSRF tokens to JSON requests and validate requests on the server side. Use security libraries to defend against CSRF attacks targeting JSON endpoints.

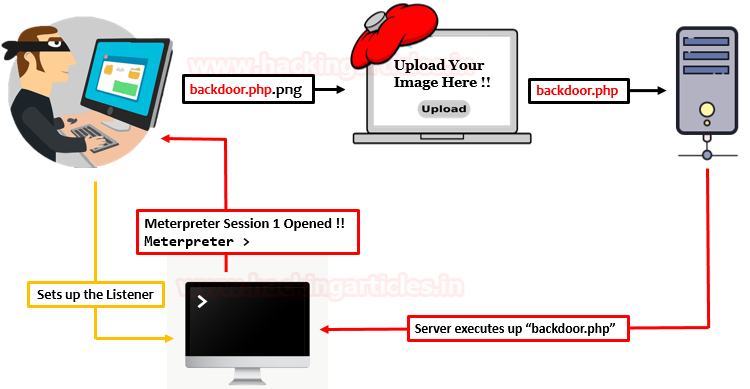
**8)Html5 local storage vulnerabilities**

HTML5 Local Storage Vulnerabilities involve insecure use of local storage in web applications. Attackers exploit these to steal sensitive data stored locally in the user's browser.



**Mitigation:** Avoid storing sensitive information in local storage. Encrypt data before storing and implement proper access controls to prevent unauthorized access.

**9)File upload vulnerabilities**

****

File Upload Vulnerabilities occur when an application allows users to upload files without proper validation. Attackers can upload malicious files to compromise the server or users' systems.

**Mitigation:** Enforce strict file type and size validation for uploads, store uploaded files outside the web root, and scan uploaded files for malware.

**10)Business Logic Vulnerabilities**

****

Business Logic Vulnerabilities involve manipulation of the expected application workflow. Attackers exploit these to gain unauthorized access or privileges, leading to data breaches or unauthorized actions.

**Mitigation:** Implement strong access controls, validate user actions on the server side, and perform thorough threat modeling to identify potential vulnerabilities in the application's logic.