# **Web Application Security: Vulnerabilities, Exploitation & Learnings**

## **1. Introduction**

Web applications are prone to various security vulnerabilities, which can be exploited by attackers to compromise sensitive data, disrupt operations, or gain unauthorized access. This document covers common web vulnerabilities, practical exploitation using **Burp Suite (Community Edition), OWASP ZAP, and Kali Linux**, and key security learnings.

## **2. Common Web Application Vulnerabilities & Exploitation Steps**

### **2.1 SQL Injection (SQLi)**

#### **Overview:**

SQL Injection (SQLi) is a severe vulnerability that allows attackers to manipulate SQL queries by injecting malicious input, potentially leading to unauthorized data access, database modification, or even complete system compromise. It includes attacks like union-based, error-based, and blind SQLi. Best practices to mitigate these risks involve using prepared statements, input validation, least privilege access controls, and web application firewalls to prevent malicious SQL execution.

#### **Exploitation Steps:**

* Identify an input field vulnerable to SQL injection (e.g., login form, search box).

Use payloads like:  
 ' OR 1=1 --

" OR "1"="1"

admin' --

* If the database is vulnerable, the query returns all records, allowing unauthorized access.

#### **Tools Used:**

* **Burp Suite** (Intercept requests and modify SQL payloads)
* **SQLmap** (Automate SQL injection testing)

### **2.2 Cross-Site Scripting (XSS)**

#### **Overview:**

Cross-Site Scripting (XSS) is a type of security vulnerability found in web applications that allows attackers to inject malicious scripts into content that is then served to users. This can occur when a website does not properly validate or sanitize user input, enabling an attacker to execute arbitrary JavaScript in the context of another user's browser.

#### **Exploitation Steps:**

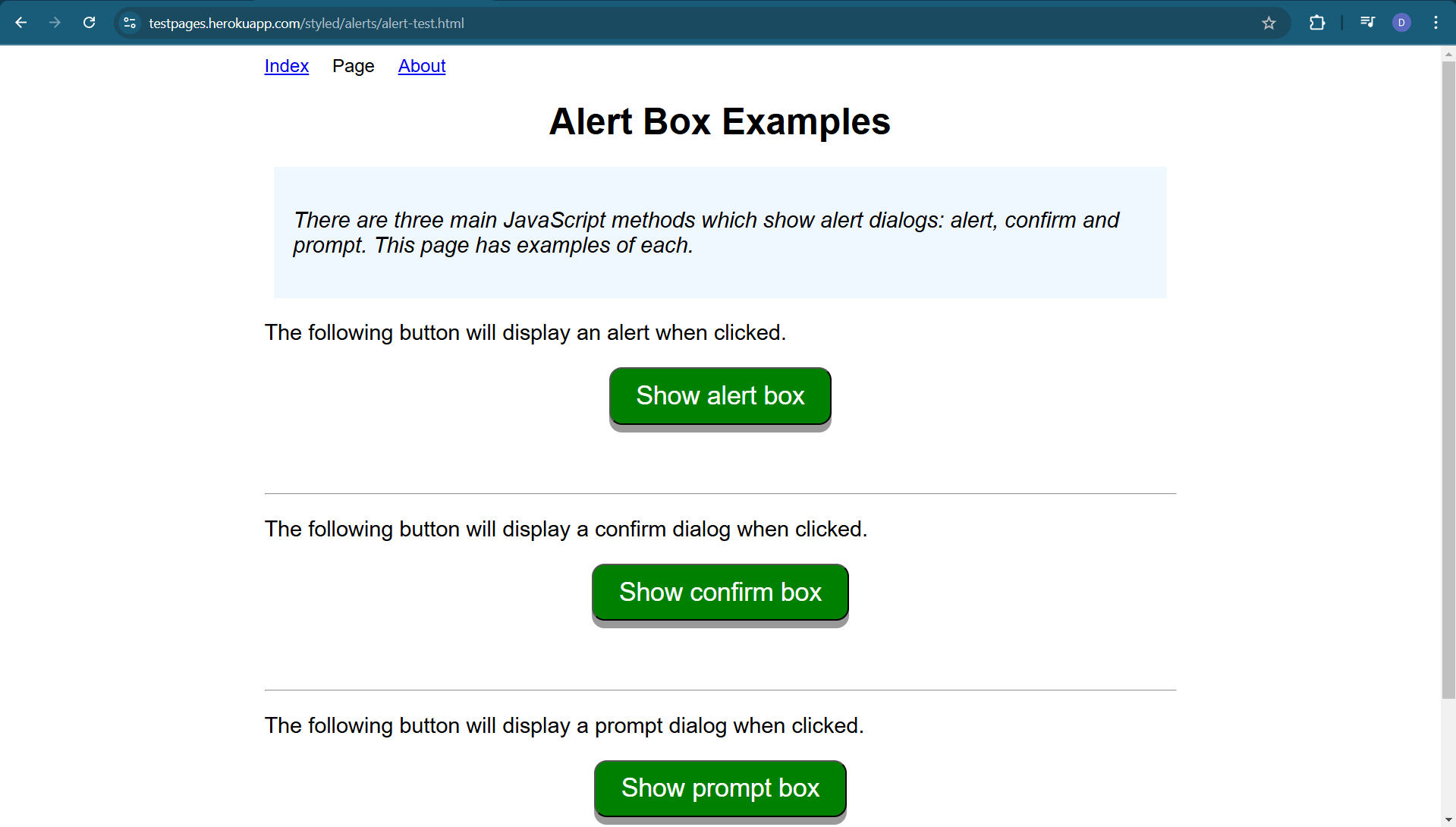
* Identify a form or input field that doesn’t sanitize user input.

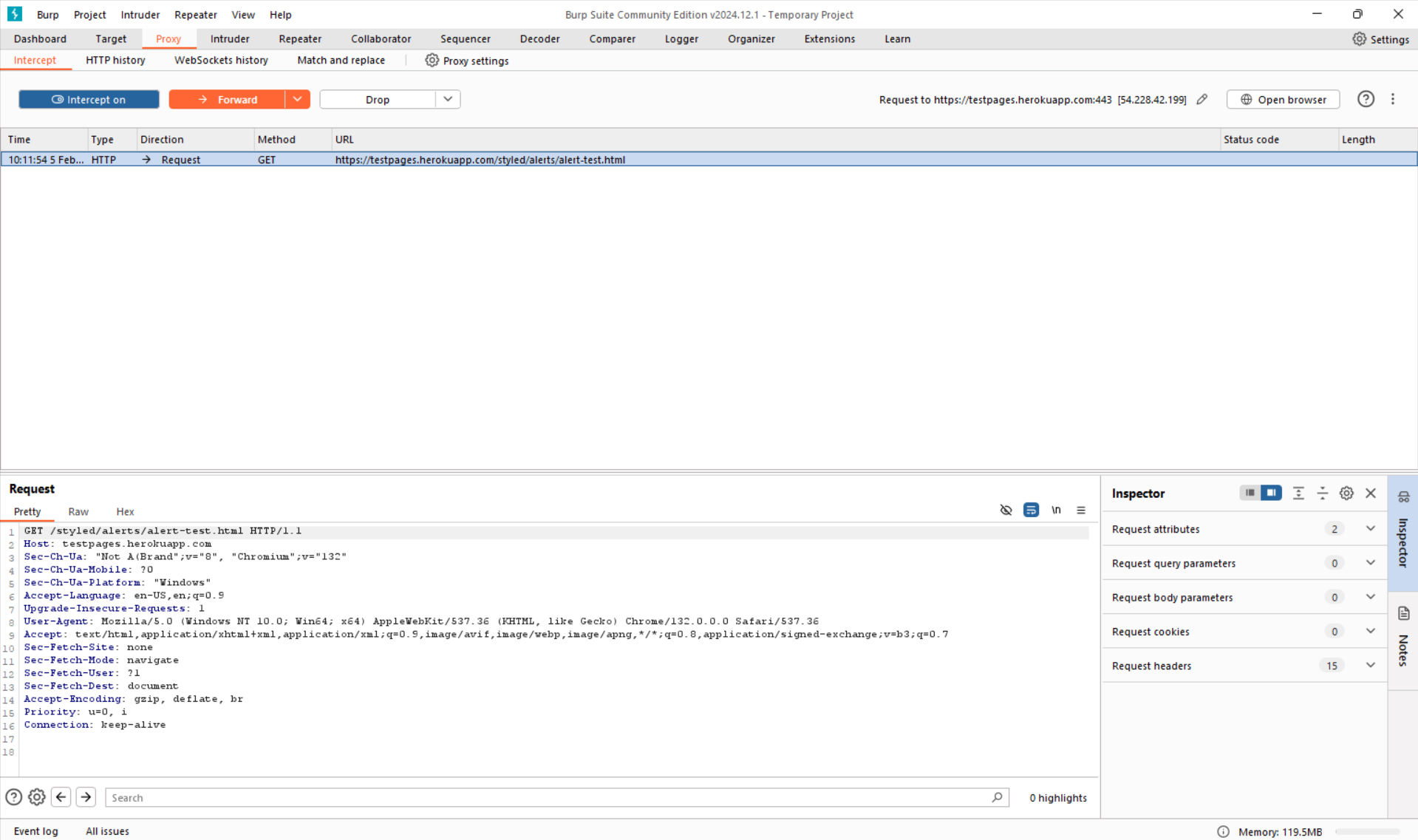
Inject JavaScript payloads like:  
<script>alert('XSS Attack!');</script>

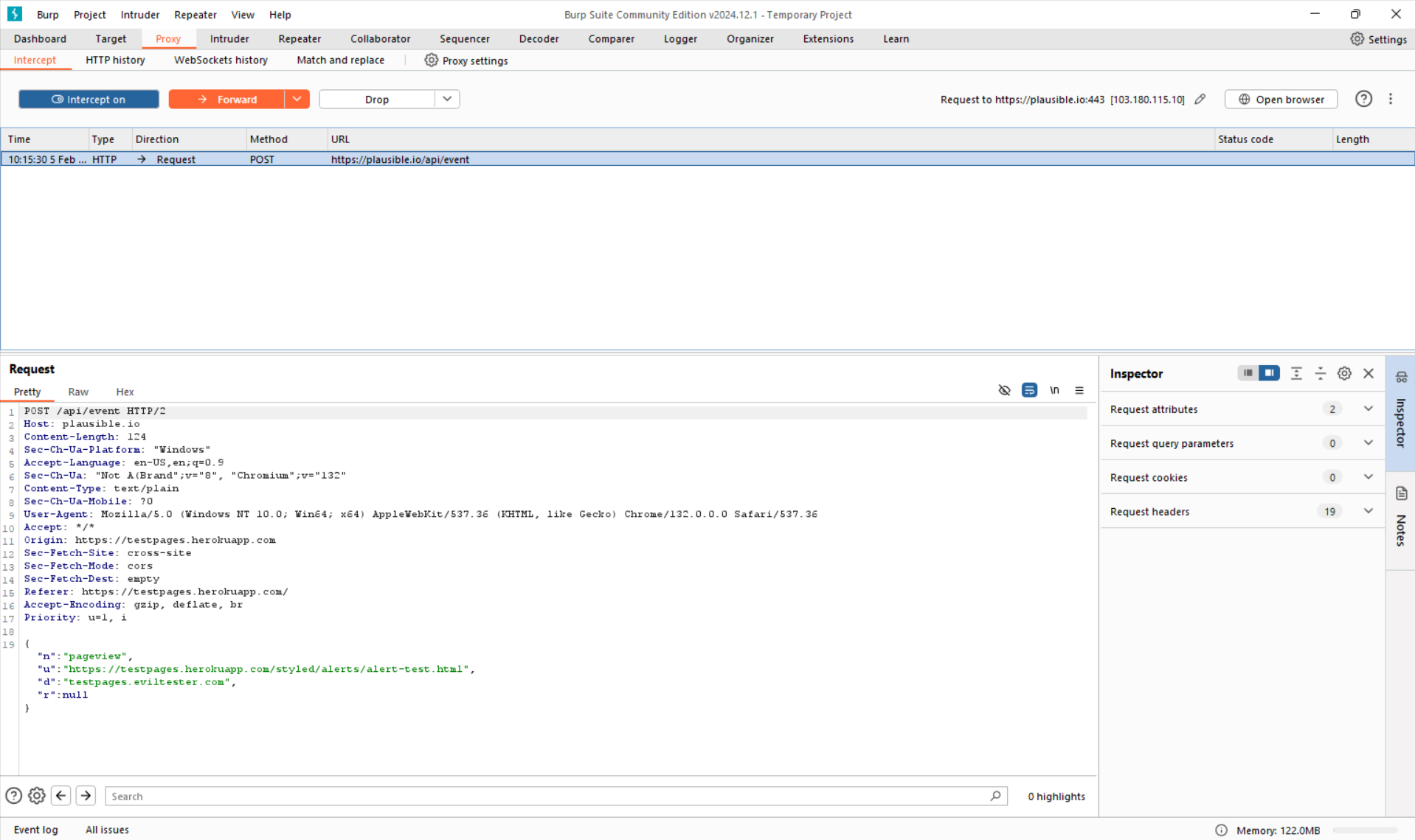
* If vulnerable, the browser executes the script, leading to session hijacking or data theft.

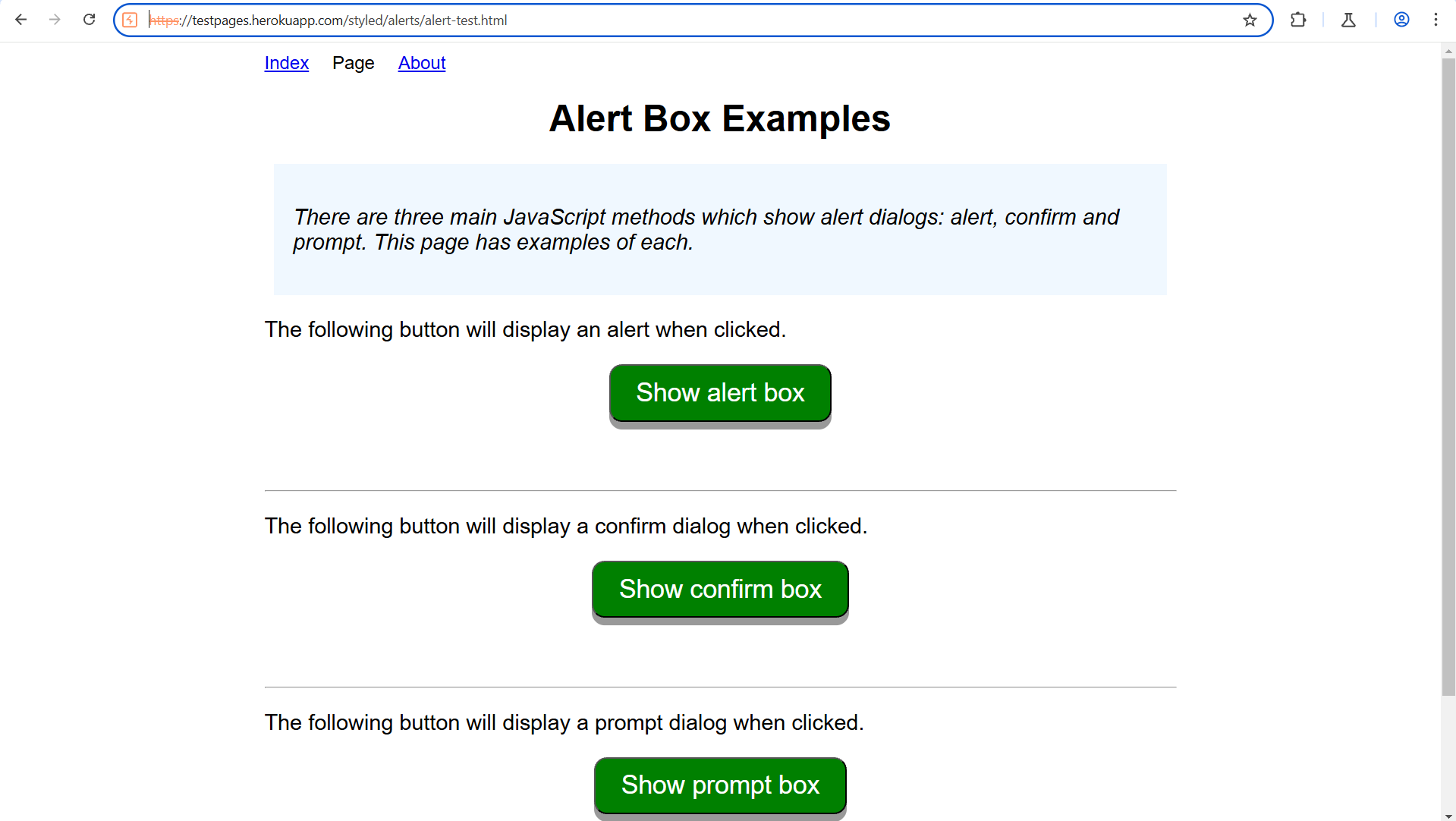
#### **Tools Used:**

* **OWASP ZAP** (Automate XSS detection)
* **Burp Suite** (Modify requests and test for stored/reflected XSS)









### **2.3 Cross-Site Request Forgery (CSRF)**

#### **Overview:**

Cross-Site Request Forgery (CSRF) allows hackers to exploit authenticated users by sending malicious forms. The tutorial demonstrates crafting a fake form to transfer money from a bank account without needing the user's credentials. It emphasizes understanding CSRF's mechanics and encourages viewers to subscribe for more cybersecurity insights.

#### **Exploitation Steps:**

* Identify a form submission that lacks CSRF protection.
* Craft a malicious request (e.g., changing user passwords without permission).
* Send the link to a victim (social engineering).

#### **Tools Used:**

* **Burp Suite** (Intercept and modify requests)
* **OWASP CSRF Tester**

### **2.4 Broken Authentication**

#### **Overview:**

Broken authentication is a significant vulnerability allowing unauthorized access to web applications. It includes attacks like credential stuffing and automated logins. Best practices to mitigate these risks involve implementing multi-factor authentication, enforcing password complexity, limiting failed logins, and managing session IDs securely to protect user data effectively.

#### **Exploitation Steps:**

* Test for default/weak credentials (e.g., admin/admin).

Perform **brute-force attacks** using:  
 hydra -l admin -P passwords.txt http://target.com/login

* Test for session fixation or missing logout functionalities.

#### **Tools Used:**

* **Hydra** (Brute-force attack)
* **Burp Suite** (Session manipulation)

### **2.5 Security Misconfigurations**

#### **Overview:**

Security misconfiguration is a critical vulnerability arising from improper security settings in applications, servers, or databases. It includes issues like default credentials, unnecessary features, misconfigured security headers, and verbose error messages. Best practices to mitigate these risks involve regularly updating software, disabling unused features, enforcing secure configurations, and conducting security audits to prevent unauthorized access and data exposure.

#### **Exploitation Steps:**

* Identify exposed admin panels or debug pages.
* Check for **default credentials** (e.g., admin:admin).

Enumerate directories using:  
 dirb <http://target.com/>

#### **Tools Used:**

* **Dirb** (Directory enumeration)
* **Burp Suite** (Identify misconfigurations)

### **2.6 Server-Side Request Forgery (SSRF)**

#### **Overview:**

Server-Side Request Forgery (SSRF) is a critical vulnerability that allows attackers to manipulate a server into making unintended requests to internal or external resources. It can lead to unauthorized data access, internal system exposure, or even remote code execution. Best practices to mitigate these risks involve enforcing strict allowlists, validating and sanitizing user inputs, restricting network access, and using metadata service protections to prevent unauthorized requests.

#### **Exploitation Steps:**

* Identify a vulnerable URL parameter (e.g., image fetcher feature).

Inject internal IP payloads:  
 http://localhost:8080/admin

* If vulnerable, gain access to internal resources.

#### **Tools Used:**

* **Burp Suite** (Modify requests)
* **Kali Linux** tools

##### **2.7 XML External Entity (XXE) Attack**

**Overview:**

XML External Entity (XXE) Attack is a security vulnerability that arises when an application improperly processes XML input containing external entity references. Attackers exploit this flaw to gain unauthorized access to sensitive data, conduct server-side request forgery (SSRF), execute denial-of-service (DoS) attacks, or even achieve remote code execution.

To mitigate XXE attacks, best practices include disabling external entity parsing in XML parsers, enforcing strong input validation, using less vulnerable data formats like JSON, implementing secure XML libraries, and restricting file system and network access for untrusted XML inputs.

**Exploitation Steps:**

1. Identify an endpoint accepting XML input.

Inject an XXE payload:  
<!DOCTYPE foo [

<!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>

<foo>&xxe;</foo>

1. If vulnerable, retrieve sensitive files or cause system disruptions.

**Tools Used:**

* Burp Suite (Modify XML requests)
* OWASP ZAP

##### **2.8 Insecure Direct Object References (IDOR)**

**Overview:**

Insecure Direct Object References (IDOR) is a critical vulnerability that occurs when an application exposes direct access to objects, such as database records or files, without proper authorization checks. Attackers can manipulate request parameters (e.g., modifying a user ID or file path) to access or modify unauthorized data, leading to data breaches, privilege escalation, or unauthorized actions.

To mitigate IDOR risks, best practices include implementing proper access control mechanisms, enforcing least privilege principles, using secure object references (such as opaque tokens instead of direct IDs), and regularly testing for authorization flaws through security audits and penetration testing.

**Exploitation Steps:**

1. Identify predictable object references (e.g., profile?id=123).
2. Modify the request to access unauthorized data (e.g., profile?id=124).
3. If vulnerable, access another user's private information.

**Tools Used:**

* Burp Suite (Modify requests)
* OWASP ZAP

##### **2.9 Remote Code Execution (RCE)**

**Overview:**

Remote Code Execution (RCE) is a critical security vulnerability that allows attackers to execute arbitrary code on a target system remotely. This can lead to complete system compromise, unauthorized data access, malware deployment, or even full control over affected servers. RCE vulnerabilities often arise from insecure input handling, deserialization flaws, command injection, or unpatched software.

To mitigate RCE risks, best practices include validating and sanitizing user inputs, avoiding dynamic code execution from untrusted sources, implementing strict access controls, applying security patches promptly, and using techniques like Web Application Firewalls (WAFs) and runtime application self-protection (RASP).

**Exploitation Steps:**

1. Identify an input field vulnerable to command injection.
2. Inject malicious payloads like:  
    ; cat /etc/passwd
3. If successful, execute remote commands on the server.

**Tools Used:**

* Metasploit
* Burp Suite

##### **2.10 Clickjacking**

**Overview:**

Clickjacking is a web security vulnerability where an attacker tricks users into clicking on a disguised or invisible element, such as a button or link, without their knowledge. This can lead to unintended actions like changing security settings, initiating unauthorized transactions, or granting permissions to malicious applications. Attackers achieve this by embedding a legitimate website inside an invisible iframe and overlaying deceptive UI elements.

To mitigate clickjacking, best practices include implementing the X-Frame-Options HTTP header (DENY or SAMEORIGIN), using the Content Security Policy (CSP) frame-ancestors directive, employing JavaScript-based frame-busting techniques, and ensuring that sensitive actions require explicit user confirmation.

**Exploitation Steps:**

1. Embed the vulnerable site inside an invisible iframe:  
    <iframe src="http://target.com" style="opacity:0;position:absolute;"></iframe>
2. Trick users into clicking on hidden buttons.
3. If vulnerable, users unknowingly perform attacker-controlled actions.

**Tools Used:**

* Burp Suite (Identify X-Frame-Options headers)
* OWASP ZAP

## **3. Tools Used in Testing**

### **3.1 Burp Suite (Community Edition)**

* Intercepts HTTP requests/responses.
* Identifies security vulnerabilities in web applications.
* Supports **manual and automated attacks**.

### **3.2 OWASP ZAP**

* Automated scanner for security vulnerabilities.
* Finds **XSS, SQLi, CSRF, and misconfigurations**.
* Provides detailed security reports.

### **3.3 Kali Linux (via VirtualBox)**

* Contains **penetration testing tools** like Metasploit, SQLmap, and Hydra.
* Ideal for network and web security testing.

## **4. Key Learnings & Security Best Practices**

✔ **Implement proper input validation & sanitization** to prevent **SQLi and XSS**. ✔ **Use strong authentication mechanisms** (MFA, hashed passwords). ✔ **Regularly update & patch web applications** to mitigate known vulnerabilities. ✔ **Employ security headers** (CSP, X-Frame-Options, HSTS) to reduce risk. ✔ **Conduct regular penetration testing** using tools like Burp Suite, OWASP ZAP, and Kali Linux. ✔ **Implement a Web Application Firewall (WAF)** for added protection.

## **5. Conclusion**

**By understanding and exploiting vulnerabilities in test environments, we can better secure web applications from real-world attacks. This document provides hands-on techniques and security best practices to mitigate risks effectively. Regular security testing and adherence to secure coding practices are crucial for maintaining a robust security posture.**