



**Web Security – IE2062**

**Topic: Bug Bounty Report 5**

**Y2S2.WE.CS**

**Name: S.D.W.Gunaratne**

**(IT23241978)**

# Table of Content

## **1) How I started?**

## **2) Introduction**

2.1 Domain

1.2 Severity

## **3) Vulnerability**

3.1 Vulnerability title

3.2 Vulnerability description

3.3 Affected components

3.4 Impact assessment

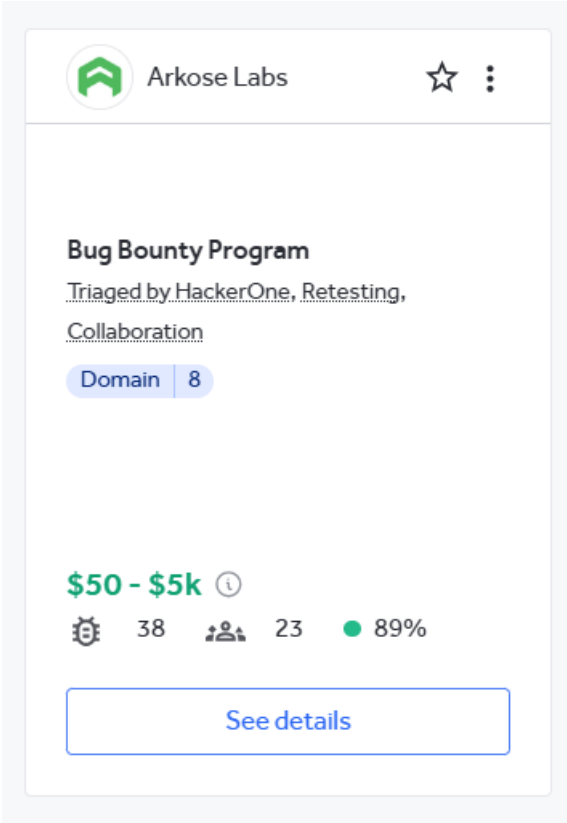
3.5 Steps to reproduce

3.6 Proof of concept

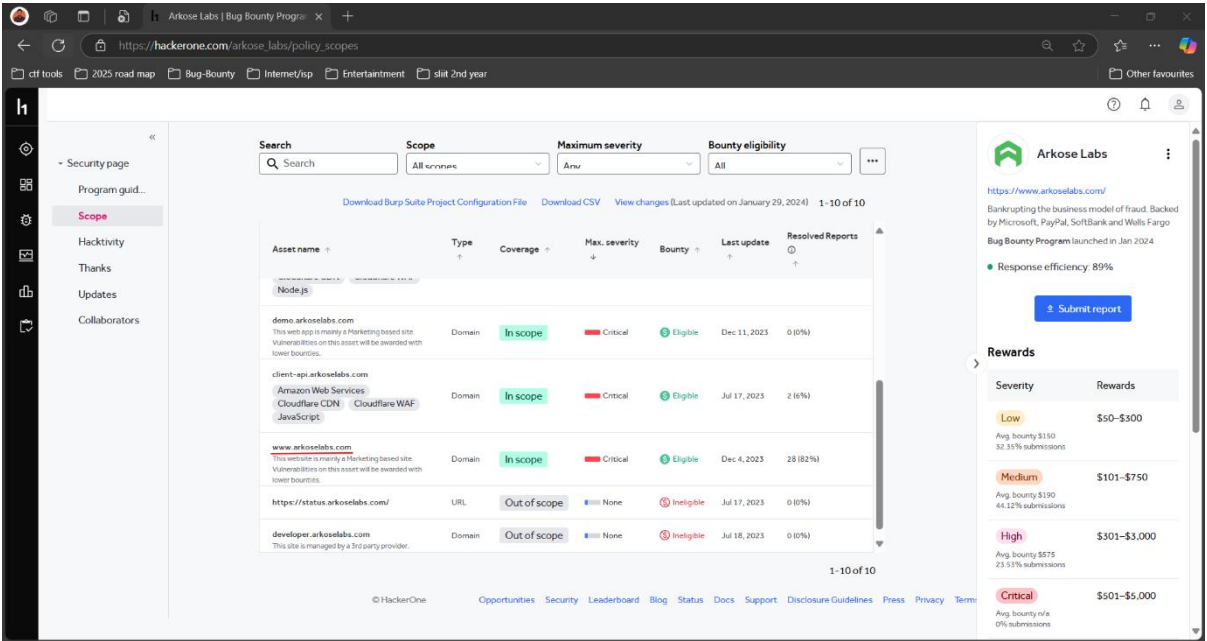
3.7 Proposed mitigation or fix

# How I started?

1. Once I search from hacker one, I saw a ArkoseLabs bug bounty program.

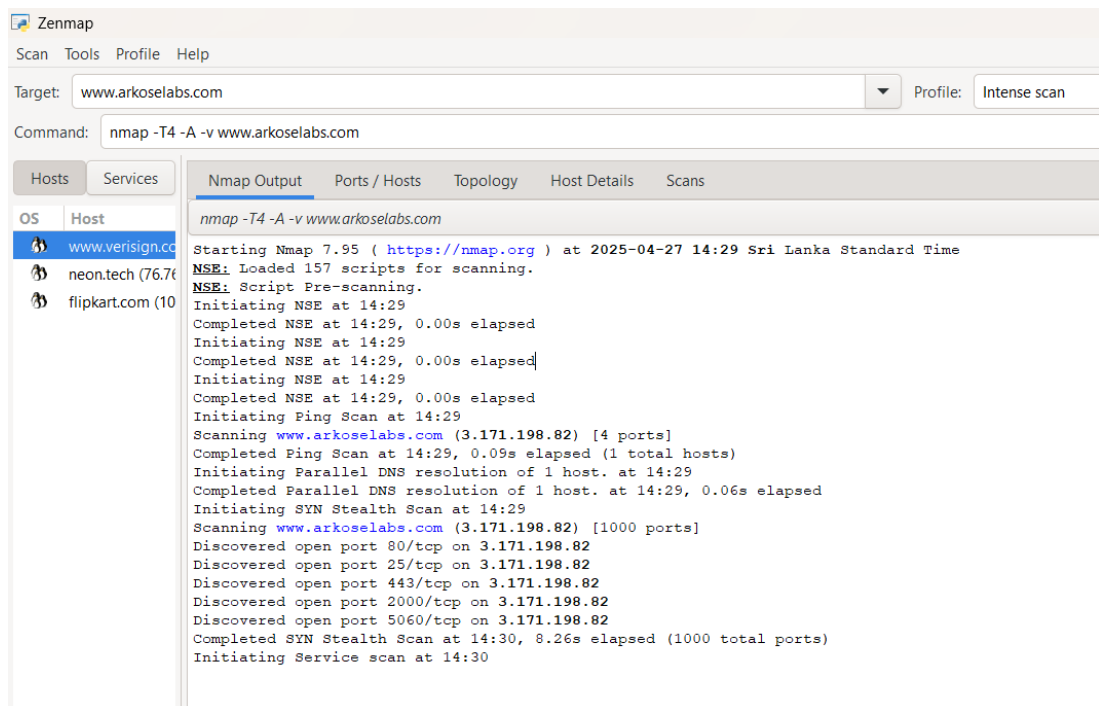


2. Then, I discovered allowed domains scope, so that I choose <https://www.arkoselabs.com> .

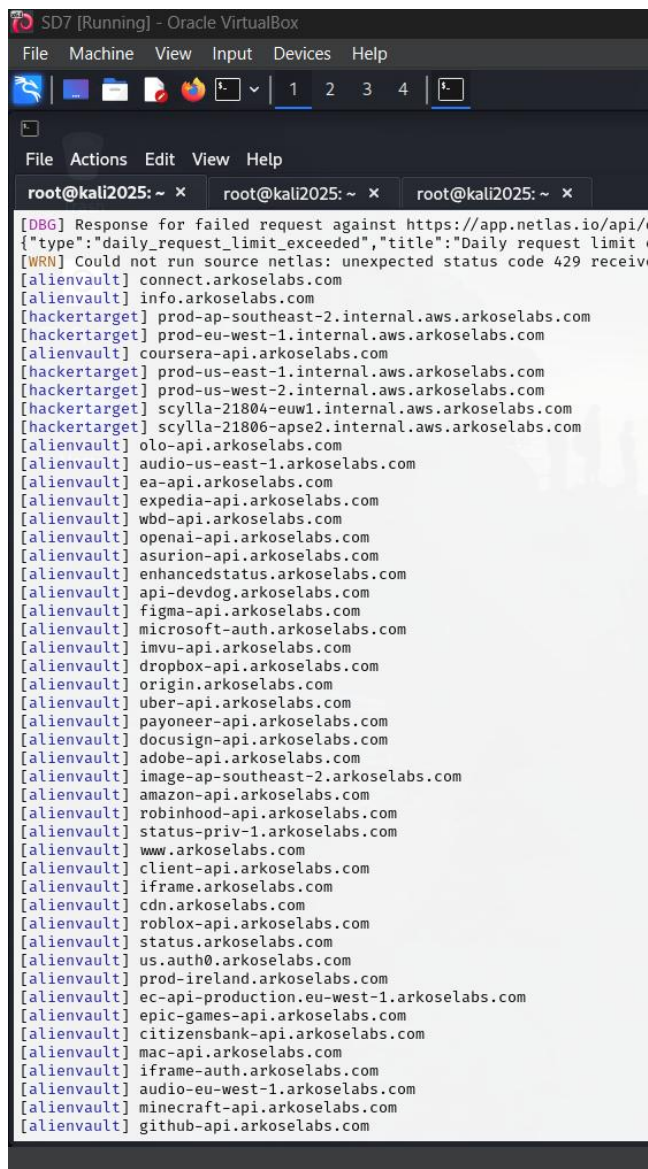


3. I use several methods/tools to do penetration testing.

4. First, I used Nmap. It helps me to find what are the open ports, Identify the web technologies such as web servers.



5. Secondly, I used Subfider tool to find hidden or forgotten web asserts. Because hidden web assert can have poor security, unpatched vulnerabilities.



```
SD7 [Running] - Oracle VirtualBox
File Machine View Input Devices Help

root@kali2025: ~ x root@kali2025: ~ x root@kali2025: ~ x

[DBG] Response for failed request against https://app.netlas.io/api/
{"type":"daily_request_limit_exceeded","title":"Daily request limit exceeded"}
[WRN] Could not run source netlas: unexpected status code 429 received
[alienvault] connect.arkoselabs.com
[alienvault] info.arkoselabs.com
[hackertarget] prod-ap-southeast-2.internal.aws.arkoselabs.com
[hackertarget] prod-eu-west-1.internal.aws.arkoselabs.com
[alienvault] coursera-api.arkoselabs.com
[hackertarget] prod-us-east-1.internal.aws.arkoselabs.com
[hackertarget] prod-us-west-2.internal.aws.arkoselabs.com
[hackertarget] scylla-21804-euw1.internal.aws.arkoselabs.com
[hackertarget] scylla-21806-apse2.internal.aws.arkoselabs.com
[alienvault] olo-api.arkoselabs.com
[alienvault] audio-us-east-1.arkoselabs.com
[alienvault] ea-api.arkoselabs.com
[alienvault] expedia-api.arkoselabs.com
[alienvault] wbd-api.arkoselabs.com
[alienvault] openai-api.arkoselabs.com
[alienvault] asurion-api.arkoselabs.com
[alienvault] enhancedstatus.arkoselabs.com
[alienvault] api-devdog.arkoselabs.com
[alienvault] figma-api.arkoselabs.com
[alienvault] microsoft-auth.arkoselabs.com
[alienvault] imvu-api.arkoselabs.com
[alienvault] dropbox-api.arkoselabs.com
[alienvault] origin.arkoselabs.com
[alienvault] uber-api.arkoselabs.com
[alienvault] payoneer-api.arkoselabs.com
[alienvault] docusign-api.arkoselabs.com
[alienvault] adobe-api.arkoselabs.com
[alienvault] image-ap-southeast-2.arkoselabs.com
[alienvault] amazon-api.arkoselabs.com
[alienvault] robinhood-api.arkoselabs.com
[alienvault] status-priv-1.arkoselabs.com
[alienvault] www.arkoselabs.com
[alienvault] client-api.arkoselabs.com
[alienvault] iframe.arkoselabs.com
[alienvault] cdn.arkoselabs.com
[alienvault] roblox-api.arkoselabs.com
[alienvault] status.arkoselabs.com
[alienvault] us.auth0.arkoselabs.com
[alienvault] prod-ireland.arkoselabs.com
[alienvault] ec-api-production.eu-west-1.arkoselabs.com
[alienvault] epic-games-api.arkoselabs.com
[alienvault] citizensbank-api.arkoselabs.com
[alienvault] mac-api.arkoselabs.com
[alienvault] iframe-auth.arkoselabs.com
[alienvault] audio-eu-west-1.arkoselabs.com
[alienvault] minecraft-api.arkoselabs.com
[alienvault] github-api.arkoselabs.com
```

6. Thirdly, I used Wafwoof tool to find website is protected by a WAF (web application firewall). Because if WAF is active, so pen tester do their test without blocked, and they can do their testing with bypass WAF.



## 1) Introduction

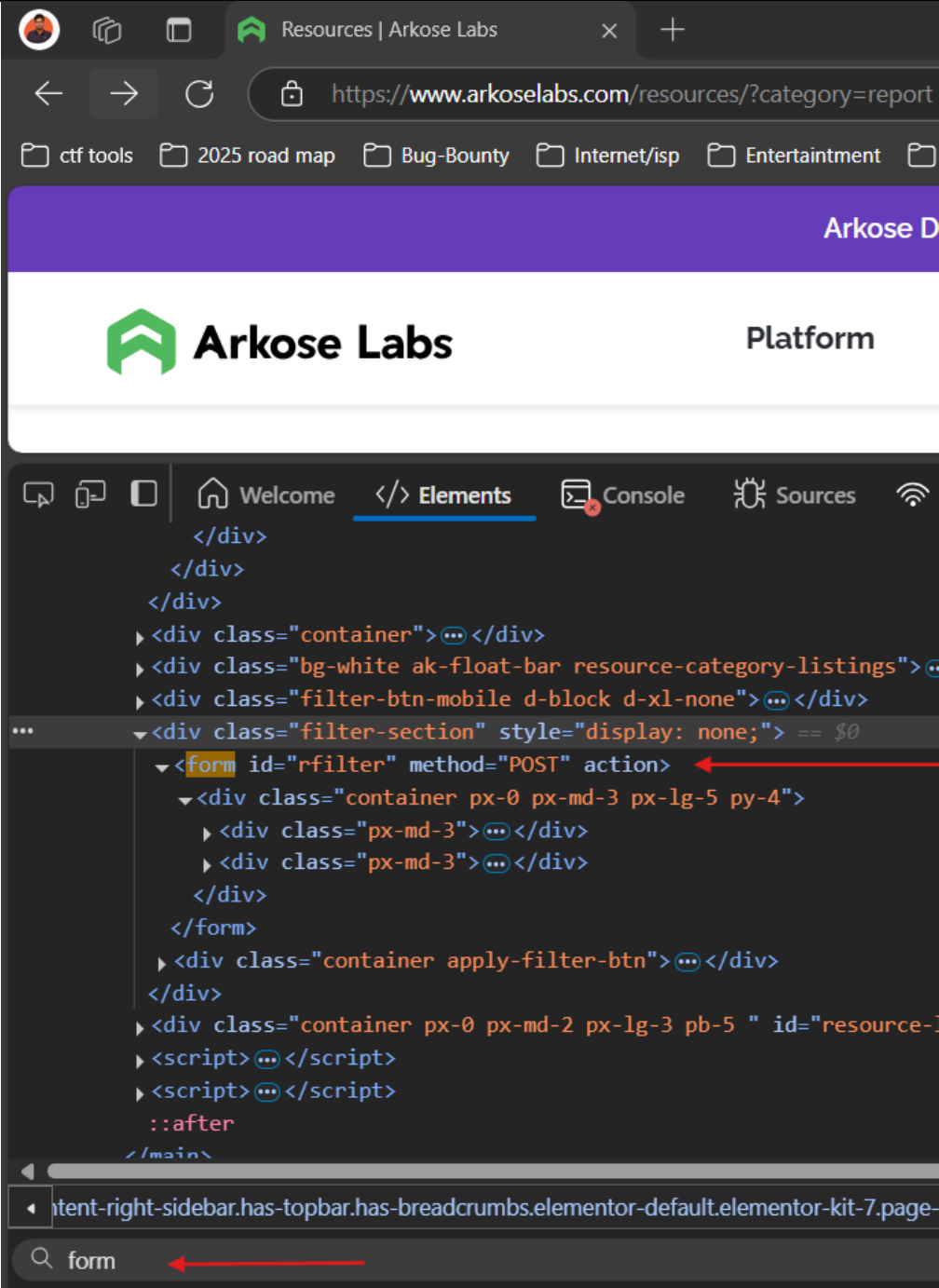
1.1 Domain	<a href="https://www.arkoselabs.com">https://www.arkoselabs.com</a> <a href="https://www.arkoselabs.com/resources/?category=report">https://www.arkoselabs.com/resources/?category=report</a>
1.2 Severity	• Medium

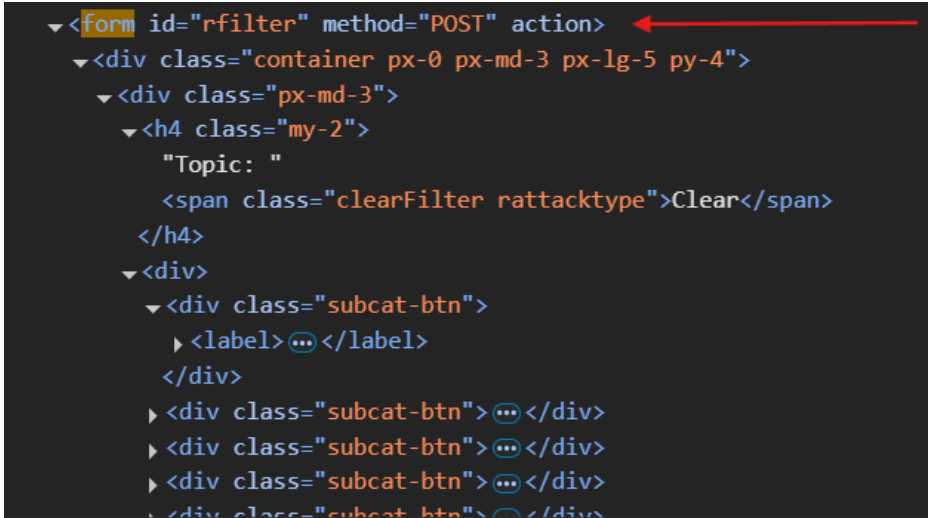
## 2) Vulnerability

2.1 Vulnerability title	Absence of Anti-CSRF Tokens  OWASP_2021_A01 CWE-352
2.2 Vulnerability description	<p>No Anti-CSRF tokens were found in a HTML submission form.</p> <p>A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim.</p> <p>The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be.</p> <p>Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.</p> <p>CSRF attacks are effective in several situations, including:</p> <ul style="list-style-type: none"><li>* The victim has an active session on the target site.</li><li>* The victim is authenticated via HTTP auth on the target site.</li><li>* The victim is on the same local network as the target site.</li></ul> <p>CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response.</p> <p>The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform</p>

	<p>for CSRF, allowing the attack to operate within the bounds of the same-origin policy.</p> <p>What is Anti-CSRF tokens: -</p> <p>It's a secret code embedded in forms to confirm that the form submission is real(original) and not done by attacker. If token is not present, It means it is easier for attackers to conduct CSRF attacks.</p>
<p><b>2.3</b> <b>Affected components</b></p>	<p>In html form component on target application is affected.</p> <p>Field: "attacktype[]" , "industry[]"</p> <p>In our case form submits data without using an Anti-SRF token.</p> <p><i>Affected component is :</i></p> <pre>&lt;form id="rfilter" method="POST" action=""&gt;</pre>



	 <p>The screenshot shows a web browser window with the URL <code>https://www.arkoselabs.com/resources/?category=report</code>. The page has a purple header with the text "Arkose D" and a white section with the "Arkose Labs" logo and "Platform" text. Below this is a navigation bar with links like "Welcome", "Elements", "Console", "Sources", and "Wi-Fi". The "Elements" panel is open, showing the DOM tree. A red arrow points to a <code>&lt;form id="rfilter" method="POST" action"&gt;</code> element. Another red arrow points to a search bar at the bottom of the Elements panel containing the text "form".</p> <p>In our case we can see form, but it's not with CSRF token.</p>
<p><b>2.4</b> <b>Impact</b> <b>assessment</b> <b>ment</b></p>	<p>Missing CSRF token provides ease to the attackers to carry out Cross-site request forgery (CSRF) attacks. If logging user is tricking into visiting a malicious page, the attacker can perform unauthorized actions on behalf of that user.</p> <p>Here are the possible risk:</p> <ul style="list-style-type: none"> <li>• Unauthorized data manipulation</li> </ul>

	<ul style="list-style-type: none"> <li>• Actions performed without user consent</li> <li>• Possible account compromise</li> <li>• Damage to user trust and business reputation</li> </ul>
<b>2.5 Steps to reproduce</b>	<ol style="list-style-type: none"> <li>1. Login into target application using a legitimate user account.</li> <li>2. Navigate to the form (Form 4) with fields like attacktype[], industry[].</li> <li>3. Go to Developer Tools → Elements or Network tab of your browser.</li> <li>4. Do the Submit form and observe the HTTP request.</li> <li>5. Note that no CSRF token appears in the request headers or body.</li> </ol>
<b>2.6 Proof of concept</b>	<p>Below screenshot represent in our form element didn't mention about CSRF tokens.</p>  <pre> ▼ &lt;form id="rfilter" method="POST" action&gt;   ▼ &lt;div class="container px-0 px-md-3 px-lg-5 py-4"&gt;     ▼ &lt;div class="px-md-3"&gt;       ▼ &lt;h4 class="my-2"&gt;         "Topic: "         &lt;span class="clearFilter rattacktype"&gt;Clear&lt;/span&gt;       &lt;/h4&gt;       ▼ &lt;div&gt;         ▼ &lt;div class="subcat-btn"&gt;           &lt;label&gt;⋮&lt;/label&gt;         &lt;/div&gt;         &lt;div class="subcat-btn"&gt;⋮&lt;/div&gt;         &lt;div class="subcat-btn"&gt;⋮&lt;/div&gt;         &lt;div class="subcat-btn"&gt;⋮&lt;/div&gt;         &lt;div class="subcat-btn"&gt;⋮&lt;/div&gt;       &lt;/div&gt;     &lt;/div&gt;   &lt;/div&gt; </pre> <p>If it is there here are the expected out comes:</p> <pre>&lt;input type="hidden" name="csrf_token" value="..."&gt;</pre>
<b>2.7 Proposed mitigation or fix</b>	<p><i>Phase 1: Architecture and Design</i></p> <p>Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.</p> <p>For example, use anti-CSRF packages such as the OWASP CSRFGuard.</p> <p><i>Phase 2: Implementation</i></p>

Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.

### *Phase 3: Architecture and Design*

Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).

Note that this can be bypassed using XSS.

Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.

Note that this can be bypassed using XSS.

Use the ESAPI Session Management control.

This control includes a component for CSRF.

Do not use the GET method for any request that triggers a state change.

### *Phase 4: Implementation*

Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

