

Sri Lanka Institute of Information Technology



Bug Bounty - Report 01

Reflected XSS Vulnerability

humanizeai.io

Student Name – Wanasinghe N.K

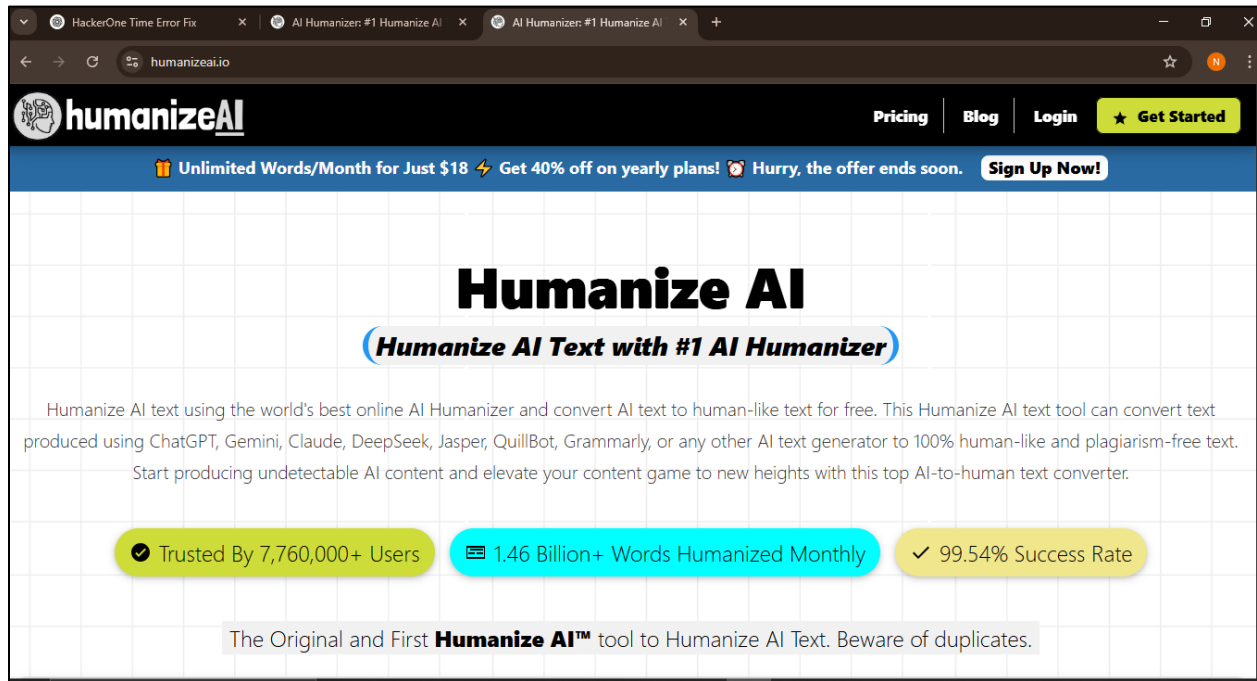
Student ID – IT23221000

IE2062 - Web Security

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

Report 01 – humanizeai.io

Main domain – <https://www.humanizeai.io/>



Reconnaissance: Gather information about the target.

I used **sublist3r** tool to find the subdomains of humanizeai.io.

```
kali linux (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
nelushi@kali: ~
File Actions Edit View Help
# Coded By Ahmed Aboul-Ela - @aboul3la

[~] Enumerating subdomains now for humanizeai.io
[~] Searching now in Baidu..
[~] Searching now in Yahoo..
[~] Searching now in Google..
[~] Searching now in Bing..
[~] Searching now in Ask..
[~] Searching now in Netcraft..
[~] Searching now in DNSdumpster..
[~] Searching now in Virustotal..
[~] Searching now in ThreatCrowd..
[~] Searching now in SSL Certificates..
[~] Searching now in PassiveDNS..
[!] Error: Virustotal probably now is blocking our requests
Process DNSdumpster-8:
Traceback (most recent call last):
  File "/usr/lib/python3.13/multiprocessing/process.py", line 313, in _bootstrap
    self.run()
    ~~~~~^
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 269, in run
    domain_list = self.enumerate()
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 649, in enumerate
    token = self.get_csrf_token(resp)
  File "/usr/lib/python3/dist-packages/sublist3r.py", line 644, in get_csrf_token
    token = csrf_regex.findall(resp)[0]
            ~~~~~^
IndexError: list index out of range
[~] Total Unique Subdomains Found: 1
link.humanizeai.io
```

Nmap – Network scanning and enumeration

I found all the open ports and detected the running services on the target server using Nmap.

```
(nelushi@kali)-[~]
└─$ nmap -Pn -sV humanizeai.io
Starting Nmap 7.95 ( https://nmap.org ) at 2025-05-01 14:39 CDT
Nmap scan report for humanizeai.io (172.66.40.202)
Host is up (0.013s latency).
Other addresses for humanizeai.io (not scanned): 172.66.43.54 2606:4700:3108::ac42:2b36 2606:4700:3108::ac42:28ca
Not shown: 995 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
25/tcp    open  smtp?
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
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port25-TCP:V=7.95%I=7%D=5/1%Time=6813CDE8%P=x86_64-pc-linux-gnu%r(Hello
SF:;2A,"552\x20Invalid\x20domain\x20name\x20in\x20EHLO\x20command.\r\n");

Service detection performed. Please report any incorrect results at https://nmap.org/submit/.
Nmap done: 1 IP address (1 host up) scanned in 39.05 seconds
```

Amass – Subdomain and DNS mapping

I found all the subdomains related to the target domain using Amass.

```
(nelushi@kali)-[~]
└─$ amass enum -d humanizeai.io
humanizeai.io (FQDN) → ns_record → phoenix.ns.cloudflare.com (FQDN)
humanizeai.io (FQDN) → ns_record → kipp.ns.cloudflare.com (FQDN)
humanizeai.io (FQDN) → mx_record → mailstore1.secureserver.net (FQDN)
humanizeai.io (FQDN) → mx_record → smtp.secureserver.net (FQDN)
link.humanizeai.io (FQDN) → cname_record → track.smtp2go.net (FQDN)
app.humanizeai.io (FQDN) → a_record → 172.66.40.202 (IPAddress)
app.humanizeai.io (FQDN) → a_record → 172.66.43.54 (IPAddress)
app.humanizeai.io (FQDN) → aaaa_record → 2606:4700:3108::ac42:2b36 (IPAddress)
app.humanizeai.io (FQDN) → aaaa_record → 2606:4700:3108::ac42:28ca (IPAddress)
172.66.40.0/21 (Netblock) → contains → 172.66.40.202 (IPAddress)
172.66.40.0/21 (Netblock) → contains → 172.66.43.54 (IPAddress)
2606:4700:3108::/48 (Netblock) → contains → 2606:4700:3108::ac42:2b36 (IPAddress)
2606:4700:3108::/48 (Netblock) → contains → 2606:4700:3108::ac42:28ca (IPAddress)
13335 (ASN) → managed_by → CLOUDFLARENET - Cloudflare, Inc. (RIROrganization)
13335 (ASN) → announces → 172.66.40.0/21 (Netblock)
13335 (ASN) → announces → 2606:4700:3108::/48 (Netblock)
track.smtp2go.net (FQDN) → a_record → 185.3.93.228 (IPAddress)
```

Wafw00f – Firewall Detection

Command used – wafw00f <https://www.humanizeai.io/>



```
( W00f! )
Home
VBox

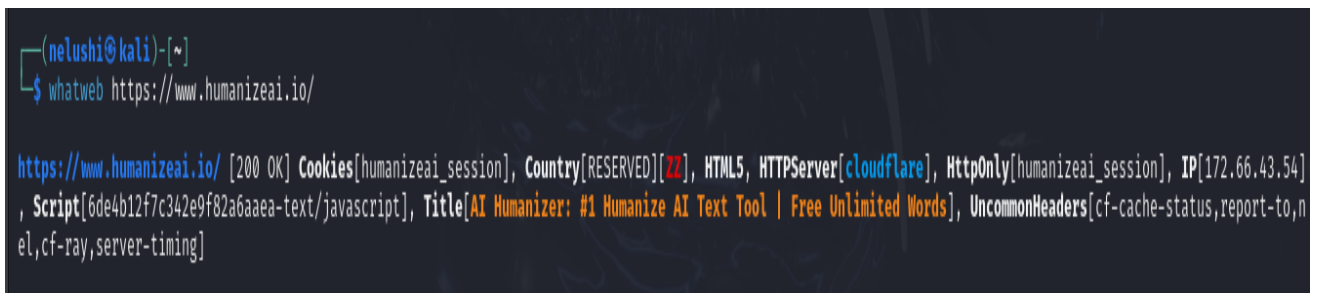
404 Hack Not Found
405 Not Allowed
403 Forbidden
502 Bad Gateway
500 Internal Error

~ WAFW00F : v2.2.0 ~
The Web Application Firewall Fingerprinting Toolkit

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

Whatweb – to identify the technologies used by the site.

Commans used – whatweb <https://www.humanizeai.io/>



```
(nelushi@kali)-[~]
$ whatweb https://www.humanizeai.io/

https://www.humanizeai.io/ [200 OK] Cookies[humanizeai_session], Country[RESERVED][22], HTML5, HTTPServer[cloudflare], HttpOnly[humanizeai_session], IP[172.66.43.54], Script[6de4b12f7c342e9f82a6aaea-text/javascript], Title[AI Humanizer: #1 Humanize AI Text Tool | Free Unlimited Words], UncommonHeaders[cf-cache-status,report-to,nel,cf-ray,server-timing]
```

Vulnerability 01

Domain

<https://www.humanizeai.io/>

Vulnerability title

Reflected XSS Vulnerability

Vulnerability description

Reflected XSS Vulnerability

Cross-Site Scripting (XSS) is a security vulnerability through which an attacker manipulates the entry of malicious scripts that are then injected into pages being viewed by other users.

Reflected Cross-Site Scripting is a type of client-side vulnerability associated with web applications.

User-supplied data is included in response HTML without validation or encoding, creating this vulnerability.

It is not stored; it is reflected off the server as in the name.

It is executed via a link, an email, or third-party website.

This type of XSS executes as soon as the victim opens the crafted URL.

This is dependent on social engineering to trick the victim into clicking or submitting something malicious.

How Reflected XSS works

The attacker frames a malicious URL statement with JavaScript payload in one of its parameters.

Ex: [https://example.com/search?q=<script>alert\('XSS'\)</script>](https://example.com/search?q=<script>alert('XSS')</script>)

When they click the link (in an email, DM), the webserver reflects back the malicious input directly into the HTML response — like this:

`<p>You searched for: <script>alert('XSS')</script></p>`

Therefore, the victim's browser executes this embedded javascript code.

The attacker's script gets executed in the context of the victim's session which can allow:

- Stealing Cookies
- Redirection
- Manipulating UI
- Unauthorized Actions

Affected component

The issue is in the **main input text field**, which processes user input without proper sanitization.

Impact assessment

Severity – High

- Arbitrary JavaScript Execution

Attackers can inject scripts into the victim browser, which can then lead to different exploits.

- Session Hijacking

Attackers steal session cookies to impersonate a user using `document.cookie`

- Credential Theft

Fake login forms or keylogging scripts can collect username and passwords of users.

- User Redirection to Malicious Sites

Users can be tricked into visiting phishing or malware sites.

- Browser Exploration

Malicious scripts may exploit browser-specific vulnerabilities, giving a way to more serious attacks.

- Denial of Service

Through infinite loops, alerts, or DOM manipulations, scripts can crash the browser.

- Damage User Trust

Suspicious activities like popups, sliding windows, and redirects reduce the credibility of the platform.

- Bypassing Client-Side Validations

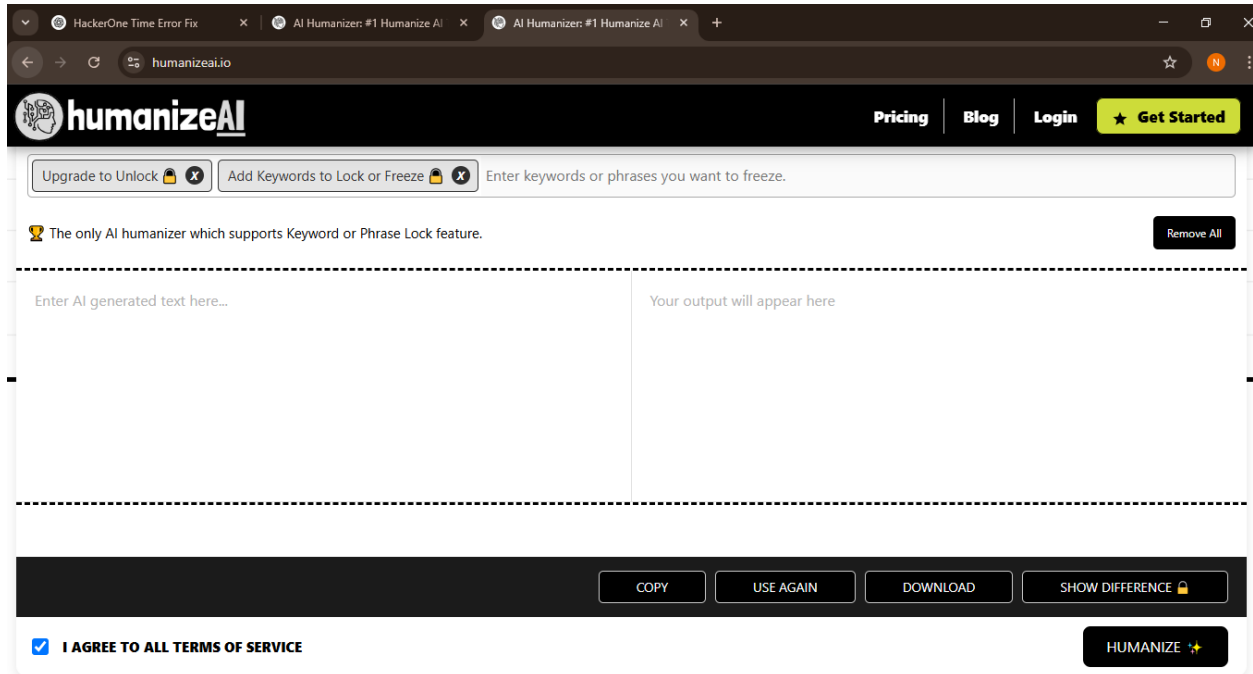
Attackers will manipulate client-side checks that can lead to security or logic being bypassed.

- Targeted Attacks (Spear Phishing)

Spear-phishing is when a personalized payload is engineered for a specific target to increase success rates.

Steps to reproduce with Proof of Concept (poc)

1. First, I navigated to the vulnerable site (humanizeai.io) and found the text input field.



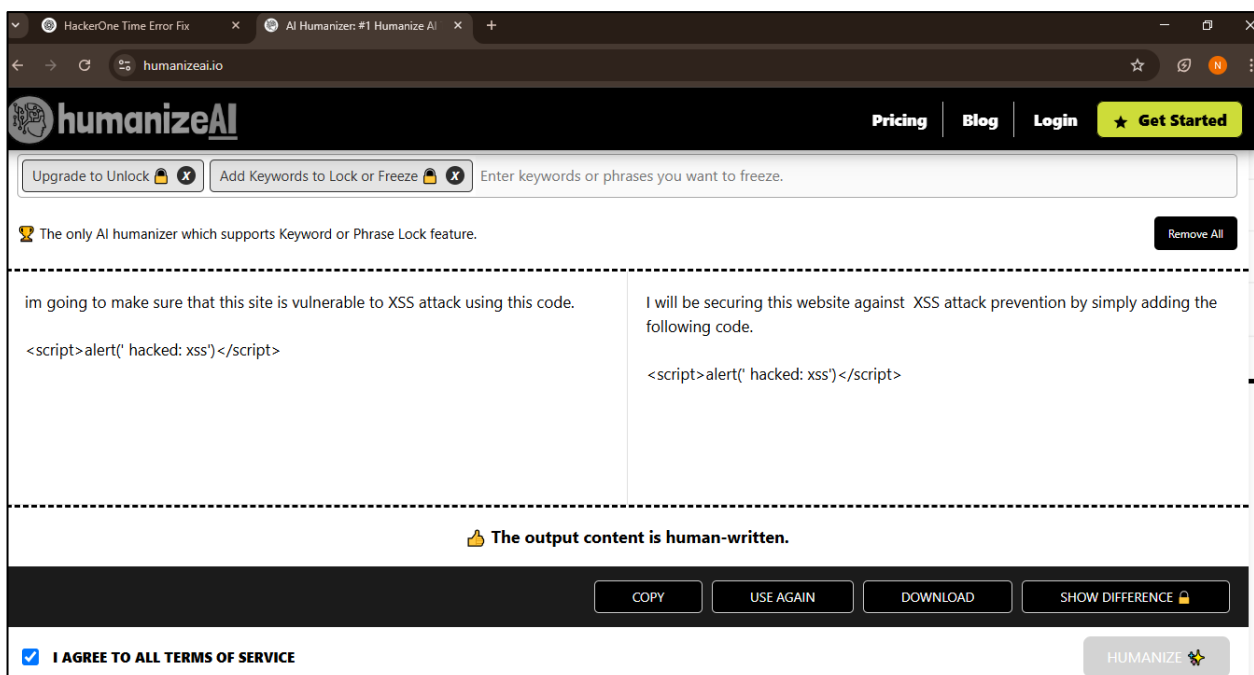
The screenshot shows the humanizeai.io website. At the top, there's a navigation bar with links for Pricing, Blog, Login, and a Get Started button. Below the navigation bar, there's a section for upgrading to unlock features, with buttons for 'Upgrade to Unlock' and 'Add Keywords to Lock or Freeze'. A text input field is present for entering keywords or phrases to freeze. Below this, there's a message stating 'The only AI humanizer which supports Keyword or Phrase Lock feature.' and a 'Remove All' button. The main content area is divided into two columns: 'Enter AI generated text here...' and 'Your output will appear here'. At the bottom, there's a section with buttons for COPY, USE AGAIN, DOWNLOAD, and SHOW DIFFERENCE. A checkbox for 'I AGREE TO ALL TERMS OF SERVICE' is also visible, along with a HUMANIZE button.

Reflection

The site loads normally, and it has a large input text field and a button “Humanize” to process the input.

2. I tried to inject a basic XSS payload like this and clicked the humanize button.

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



The screenshot shows the humanizeai.io website after an XSS attack. The input field contains the payload `<script>alert('hacked: xss')</script>`. The output field displays the result: `<script>alert('hacked: xss')</script>`. Below the output, there's a message stating 'The output content is human-written.' and a 'HUMANIZE' button. The interface is otherwise identical to the previous screenshot.

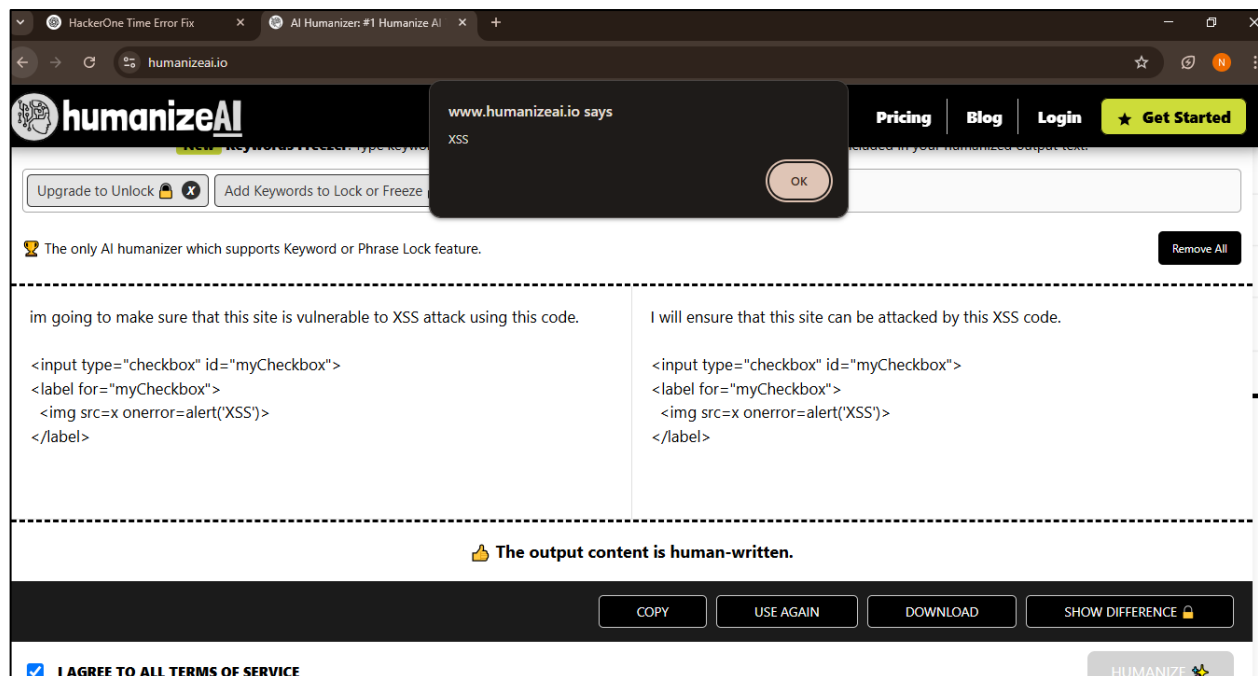
Reflection

No pop of shown, meaning that the site is sanitizing and filtering <script> tags.

XSS is an attack commonly associated with the <script> tags. Therefore, most modern web applications will either block, sanitize, or encode them.

- Next, I tried an alternative image-based payload with JavaScript in an event handler that bypasses filtering.

```
<input type="checkbox" id="myCheckbox">
<label for="myCheckbox">
  <img src=x onerror=alert('XSS')>
</label>
```



Reflection

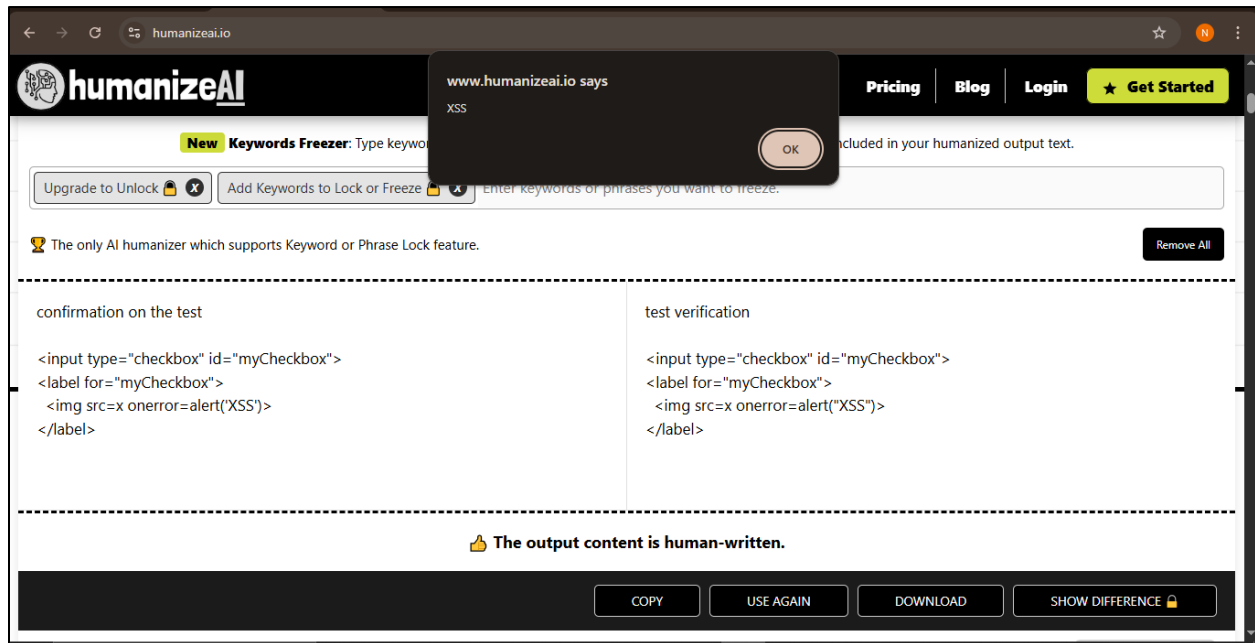
I got a pop-up alert saying "XSS".

This alert proves that the **input is reflected unsanitized**, and javascript was successfully executed.

This is a clever way to exploit XSS, **using onerror** event of the tag, executes JavaScript `alert('XSS')` when the image fails to load because `src=x` is invalid.

The HTML I inserted would have been parsed directly into the DOM with no sanitization for tags like or for attributes like onerror.

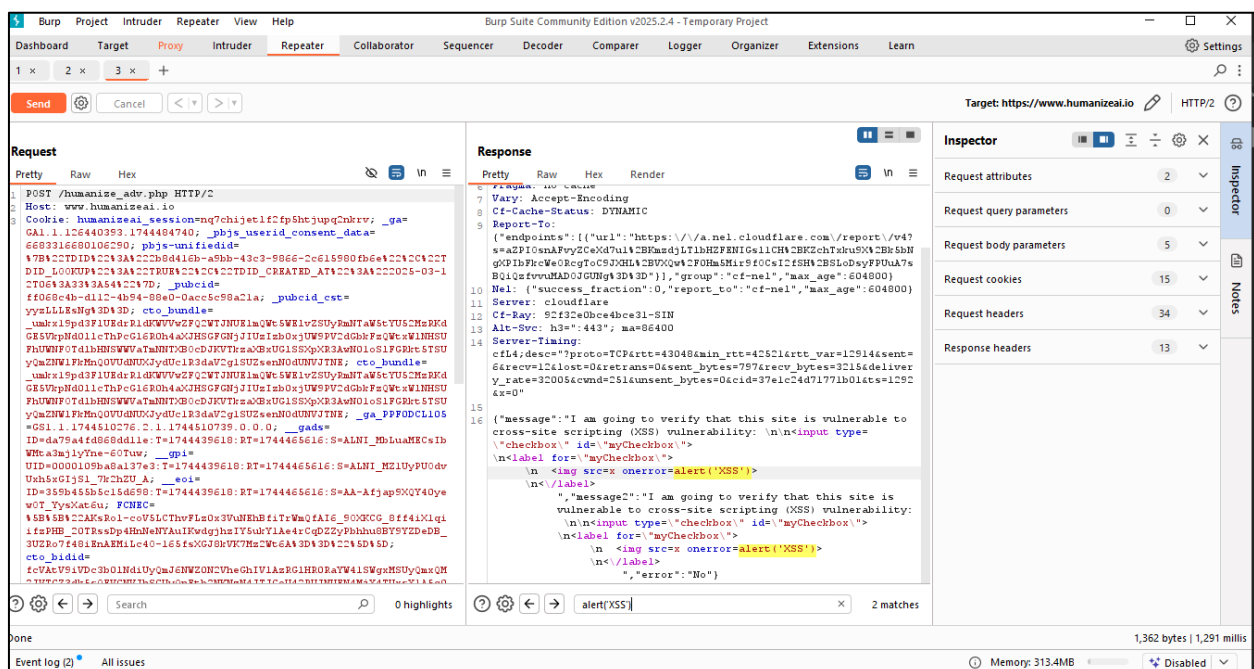
4. I repeated the same procedure to confirm that it's not a one-time occurrence.



Reflection

Each time I submit the image-based payload, the XSS alert triggers again and again. This confirms that it's a reproducible **Reflected XSS** vulnerability.

5. I used **Burp Suite** checked the HTTP response of the request by sending the request to the repeater.

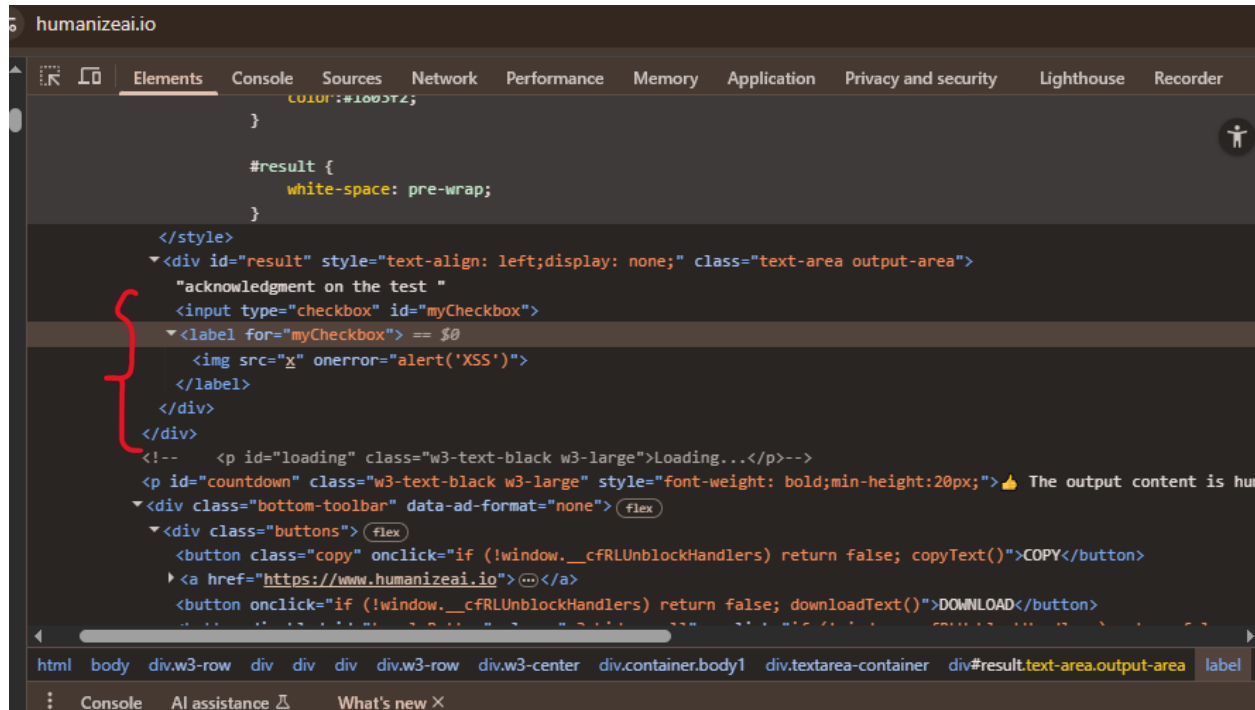


Reflection

The payload is reflected in the HTTP response.

It gets executed immediately when the victim submits input.

6. I checked the site's source code to see whether the payload is stored.



Reflection

It is stored inside an **id** called **results**.

I observed that whenever I inject a payload, it will be shown in the source code but when I refresh the page it will be removed.

How I decided that this is a Reflected XSS

- ✓ My payload shows up in an output only after providing the input - It is not persistent (it is not stored).
- ✓ My payload appears back in the HTTP response - The server sends it back unsanitized.
- ✓ The script runs in the browser - XSS is triggered.
- ✓ The script vanishes on refresh - It is not stored in either the database or server session.
- ✓ The script is sent in the HTTP response - The server is reflecting the input.

IS NOT:

Stored XSS → because it does not persist after reloading a page.

DOM-based XSS → because here the execution is due to server reflection, not only front-end JavaScript logic.

Reflected XSS is classified under **OWASP 2021 A03: Injection** and **WSTG-v42-INPV-01**, as it appears when the server responds immediately and reflects user input without processing any security filtering mechanism. The payload can be reflected in an HTTP response, interpreted in the victim's web browser often through the crafted link. It can be used by attackers to steal credentials, impersonate users, or perform unauthorized actions.

Proposed mitigation or fix

All the inputs from users must be validated and sanitized on the server side. The inputs should only conform to the formats that were expected.

Output should be encoded based on the context (HTML, air conditioning, URL) before it is displayed on the browser.

Disable Inline JavaScript: Avoid eval(), innerHTML, and similar unsafe methods unless required.

An appropriate library should be used to encode outputs. For example, OWASP Java Encoder.

Enforce a strong CSP header to make it possible to reject all unauthorized scripts.

Cookies should have HttpOnly and Security tags to minimize theft through XSS.

Modern frameworks such as React, Angular, automatically escape outputs. This will reduce XSS possibilities.


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.

Potential XSS Vulnerability – Responsible Disclosure (Educational Purpose)

 **Nelushi Wanasinghe** <nelushiwanasinghe70@gmail.com>
to mail ▾

21:32 (0 minutes ago) ☆ 😊 ↶ ⋮

Dear [humanizeai.io](#) Team,

I hope you're doing well. My name is Nelushi Wanasinghe, and I am currently a student learning about web application security as part of a university assignment.

While testing how different websites handle HTML and JavaScript input, I came across a potential **Reflected Cross-Site Scripting (XSS)** vulnerability on your website's text input field.

I submitted the following harmless JavaScript payload for testing:

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

When processed by the site, this payload triggered a browser alert, which strongly indicates that user input is being returned unsanitized, leading to a client-side script execution.

Please rest assured this was discovered accidentally during basic testing, and **no harm was done**. I did not extract any data, attempt to access accounts, or automate any scans. I'm sharing this with you **privately and respectfully**, so your team can review and resolve it if necessary.

This test was performed **purely for academic and educational purposes**, in a **non-malicious** and **non-intrusive** manner.


For your reference, I have attached my Academic report which includes a proof of concept (PoC) describing the vulnerability and its behavior.

Please let me know if you would like more details or if there's a specific security policy or submission channel you'd prefer I use. Again, this was for **educational purposes only** and not part of any malicious activity.

Thank you for your time, and I appreciate the important work your team does to maintain a secure and functional platform.

Best regards,
Nelushi Wanasinghe.
nelushiwanasinghe70@gmail.com

One attachment • Scanned by Gmail

 XSS_PoC_Report_...

↶ Reply ↷ Forward 😊