Sri Lanka Institute of Information Technology

Bug Bounty - Report 07

Reflected XSS Vulnerability

aiundetect.com

Student Name – Wanasinghe N.K

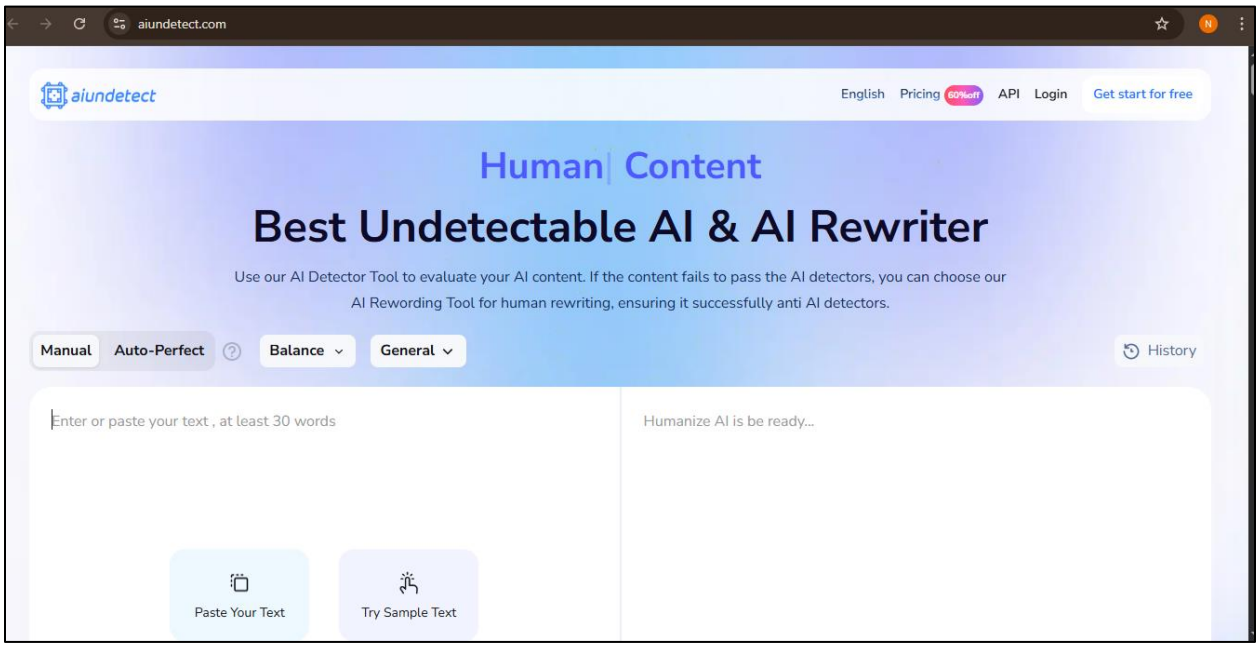
Student ID – IT23221000

IE2062 - Web Security

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

Report 07 – aiundetect.com

Main domain – https://www.aiundetect.com/

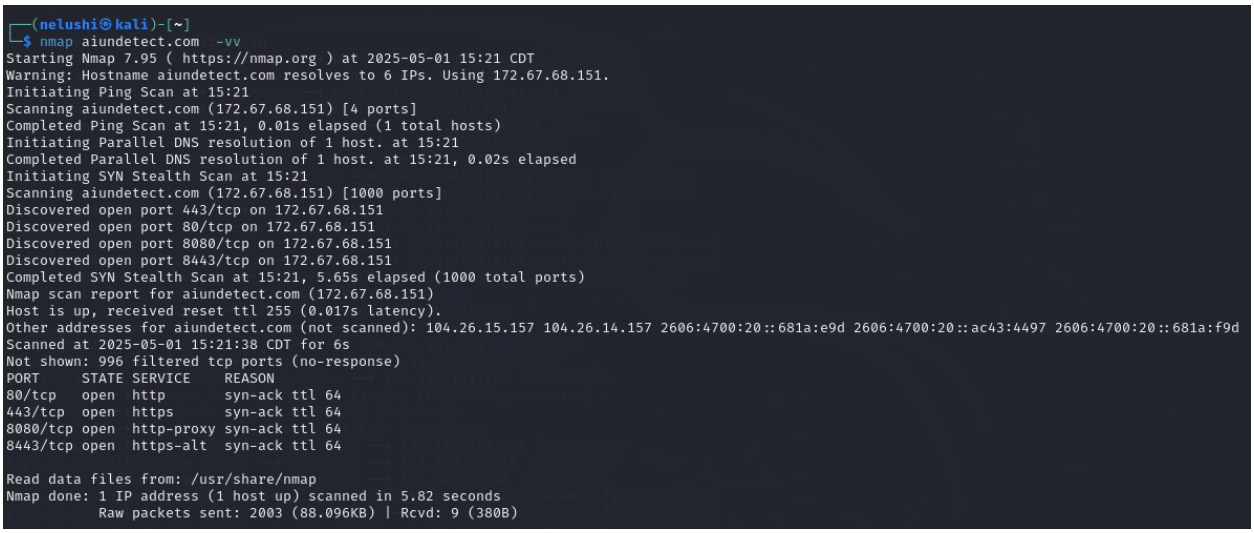


Reconnaissance: Gather information about the target.

I used **sublist3r** tool to find the subdomains of aihumanize.io. and I got an error.

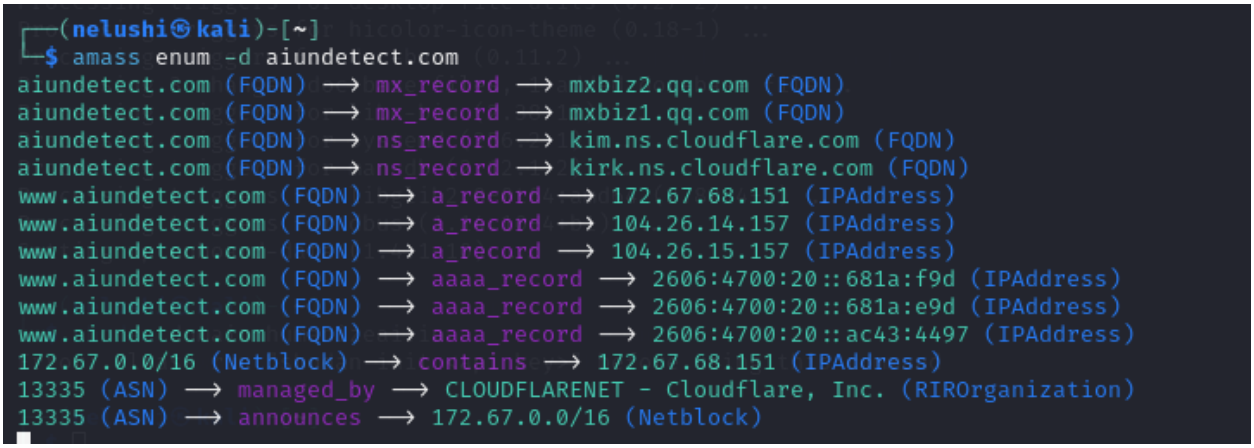
Nmap – Network scanning and enumeration

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



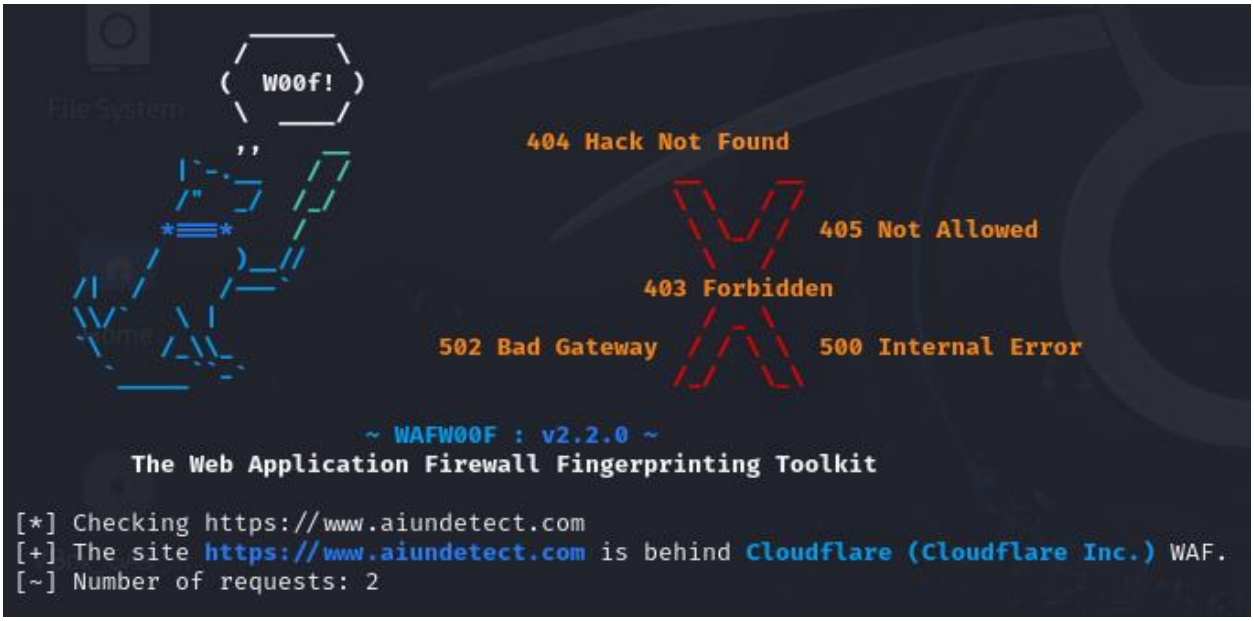
Amass – Subdomain and DNS mapping

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



Wafw00f – Firewall Detection

Command used – wafw00f https://www.aiundetect.com/



Whatweb – to identify the technologies used by the site.

Commans used – whatweb https://www.aiundetect.com/

Vulnerability 01

Domain

https://www.aiundetect.com/

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>

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

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

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 components

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

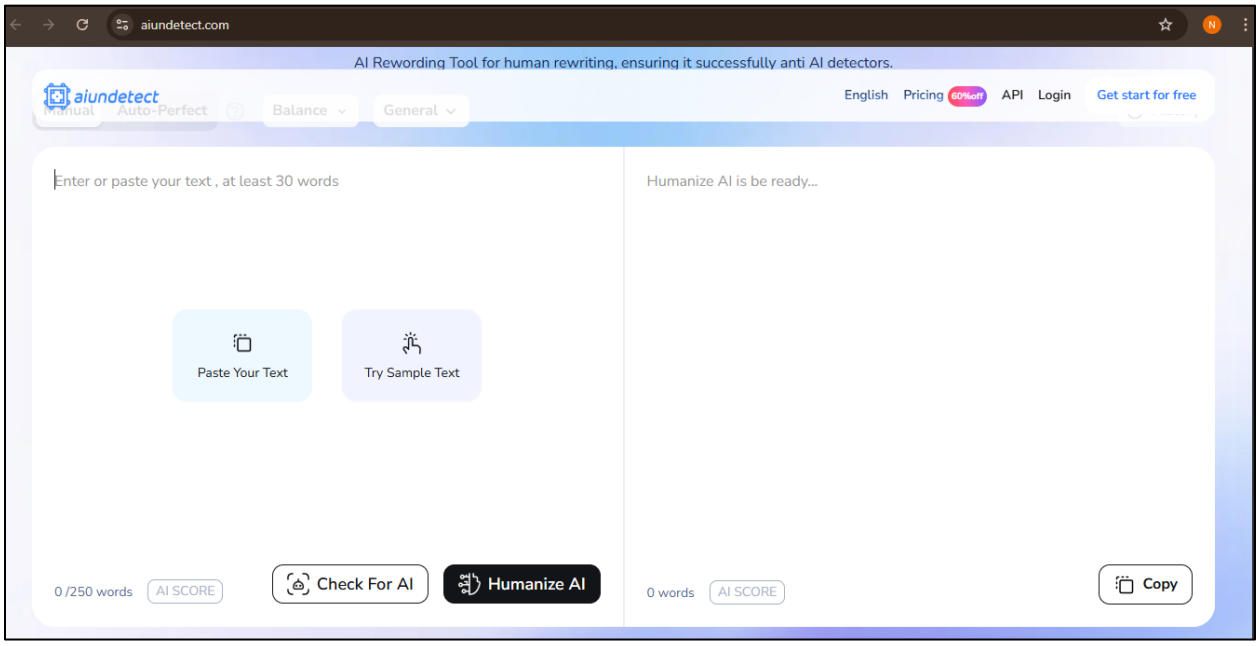
Attackers will manipulate client-side checks that can lead to security or logic being bypassed.

- 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 (aiundetect.com) and found the text input field.

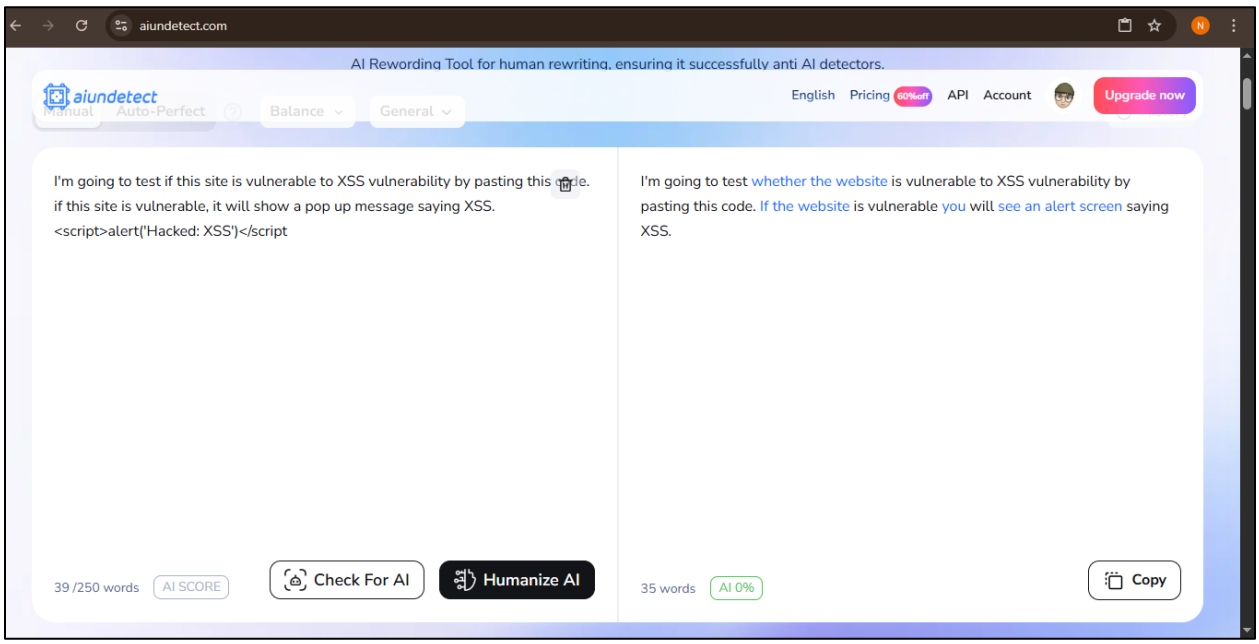


Reflection

The site loads normally, and it has a large input text field and a button “Humanize AI” 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>

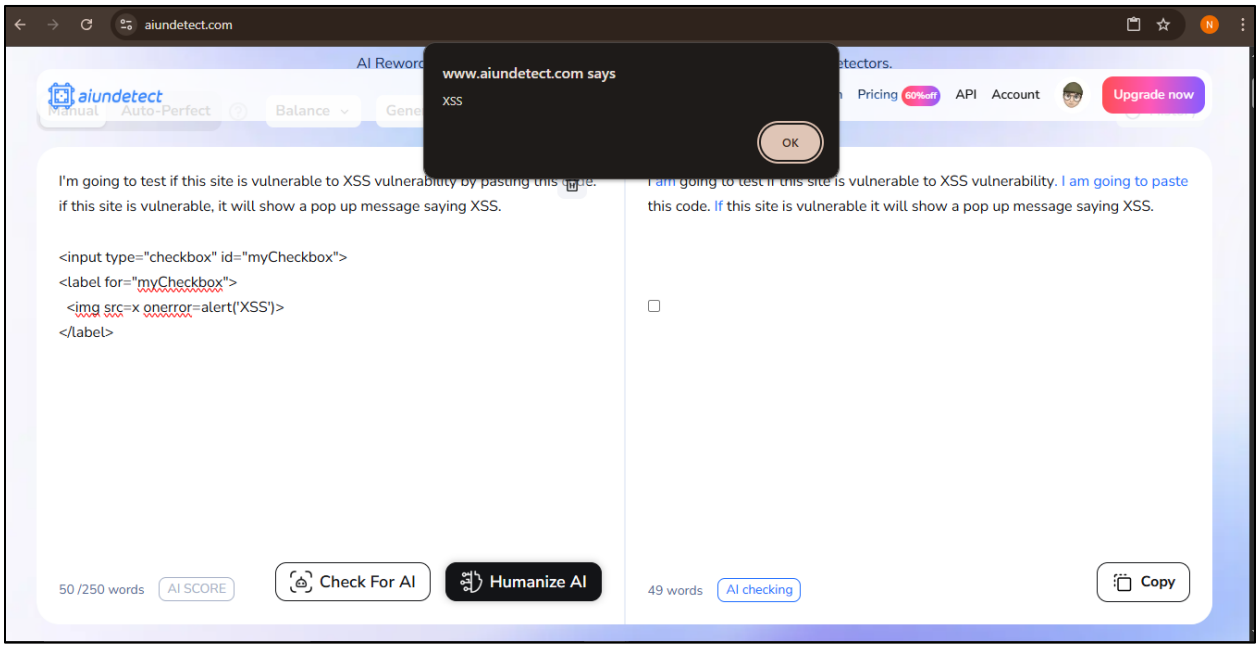


Reflection
No pop of shown, meaning that the site is sanitizing and filtering <script> tags.

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

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

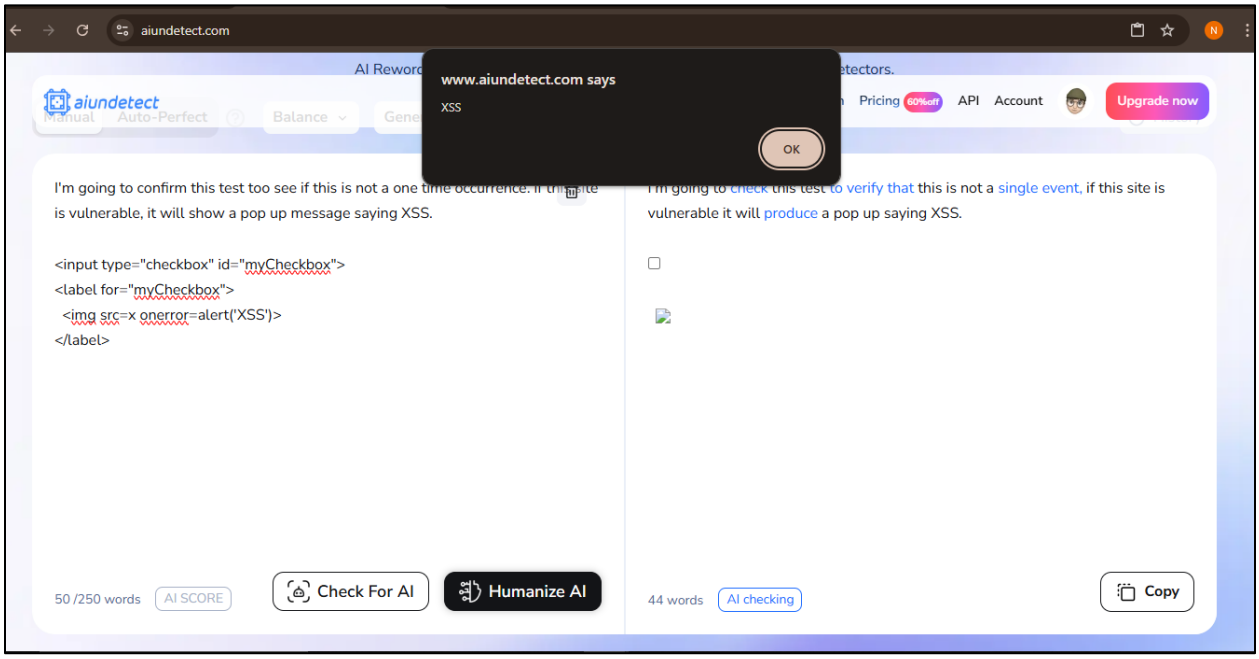
```
<input type="checkbox" id="myCheckbox">
<label for="myCheckbox">
  <img src=x onerror=alert('XSS')>
</label>
```



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 <img> tag, executes JavaScript alert('XSS') when the image fails to load because src=x is invalid.

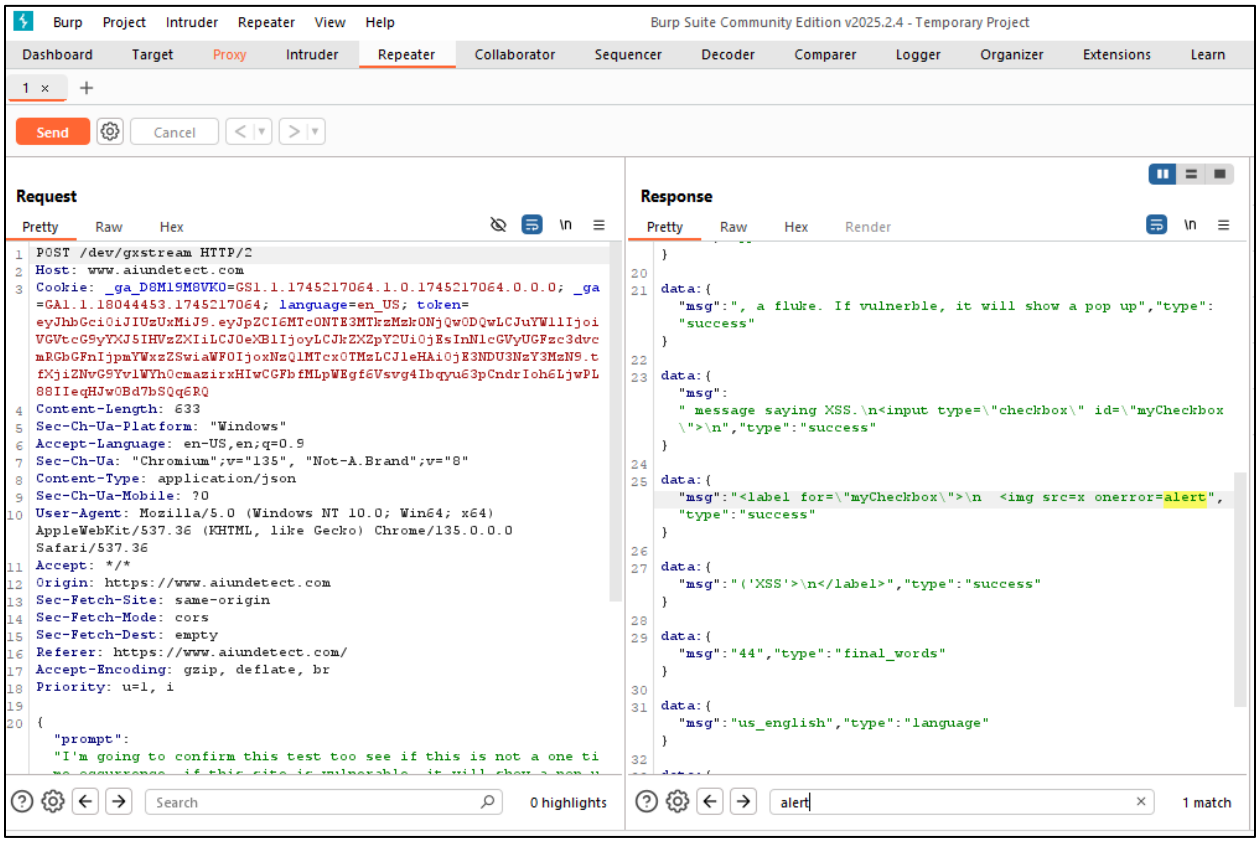
4. I repeated the same procedure to confirm that it's not a one-time occurrence.



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.

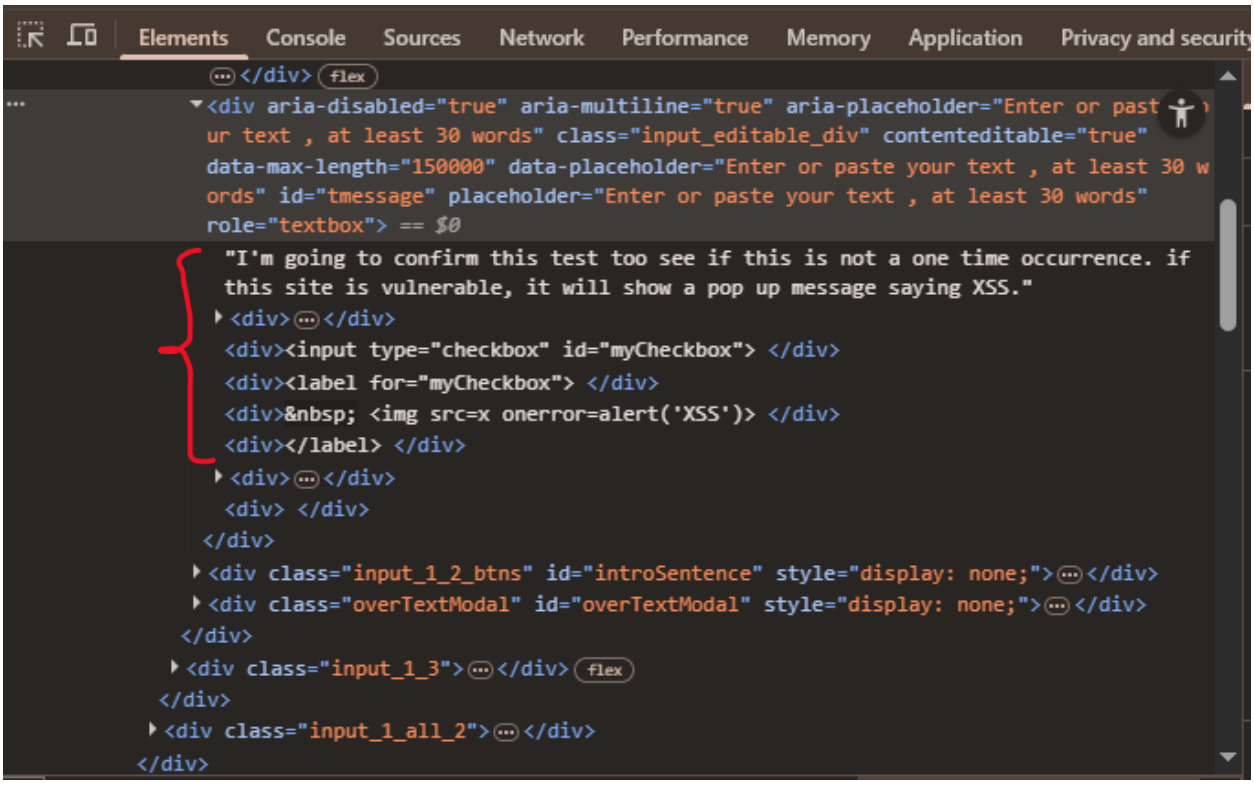
5. I used **Burp Suite** checked the HTTP response of the request by sending the request to the repeater.



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 tmessage.**
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 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.

It's 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.

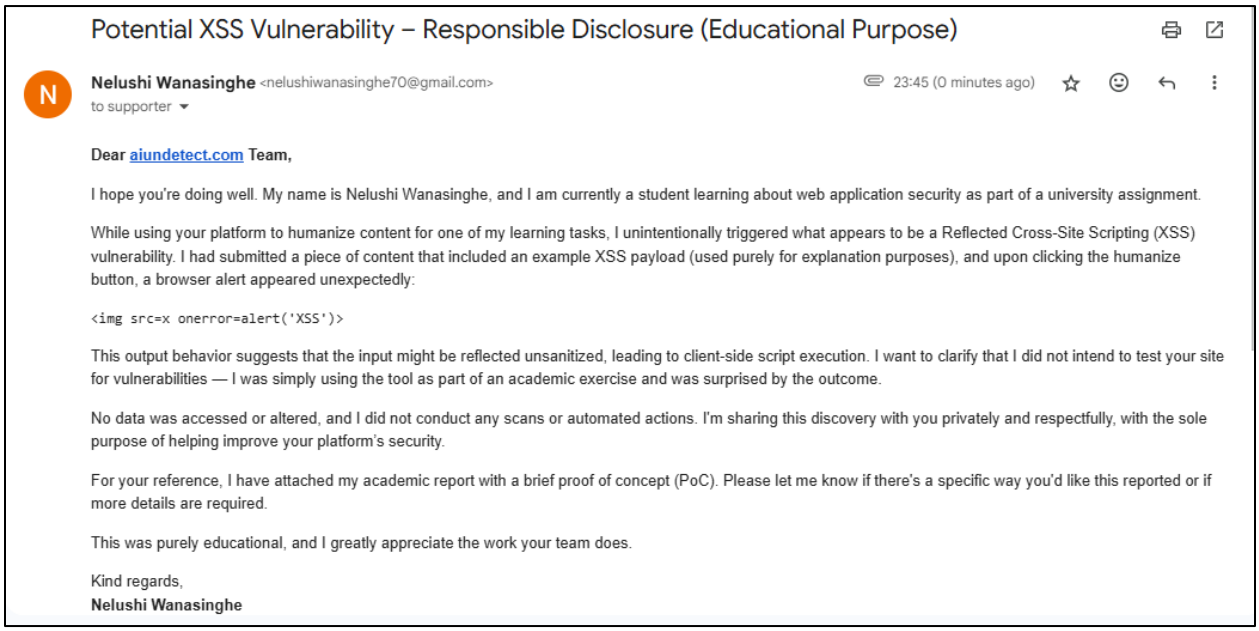
Modern frameworks such as React, Angular, automatically escape outputs. This will reduce XSS possibilities.

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 supporter

23:45 (0 minutes ago)

Dear aiundetect.com 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 using your platform to humanize content for one of my learning tasks, I unintentionally triggered what appears to be a Reflected Cross-Site Scripting (XSS) vulnerability. I had submitted a piece of content that included an example XSS payload (used purely for explanation purposes), and upon clicking the humanize button, a browser alert appeared unexpectedly:

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

This output behavior suggests that the input might be reflected unsanitized, leading to client-side script execution. I want to clarify that I did not intend to test your site for vulnerabilities — I was simply using the tool as part of an academic exercise and was surprised by the outcome.

No data was accessed or altered, and I did not conduct any scans or automated actions. I'm sharing this discovery with you privately and respectfully, with the sole purpose of helping improve your platform's security.

For your reference, I have attached my academic report with a brief proof of concept (PoC). Please let me know if there's a specific way you'd like this reported or if more details are required.

This was purely educational, and I greatly appreciate the work your team does.

Kind regards,
Nelushi Wanasinghe

Kind regards,
Nelushi Wanasinghe
nelushiwanasinghe70@gmail.com

One attachment • Scanned by Gmail

Reply Forward