Sri Lanka Institute of Information Technology

Cyber Security Assignment (2025)

Server Side Request Forgery (SSRF)

Bug Bounty Report- 06

IT23363366

TABLE OF CONTENT

1. Scope & Objective

Report Title: Server Side Request Forgery (SSRF)
Reported By: Raahim Mahmooth
Platform: https://hackerone.com/
Tested On: https://www.mtn.com/

2. Scope & Objective

The objective of this report is to investigate potential Server Side Request Forgery (SSRF) vulnerabilities on the target web application, **mtn.com**. SSRF vulnerabilities can allow an attacker to send malicious requests from the server to internal resources or other services, potentially leading to data leakage, unauthorized access, or other security risks.

3. Enumeration and Reconnaissance

3.1 Tools Used

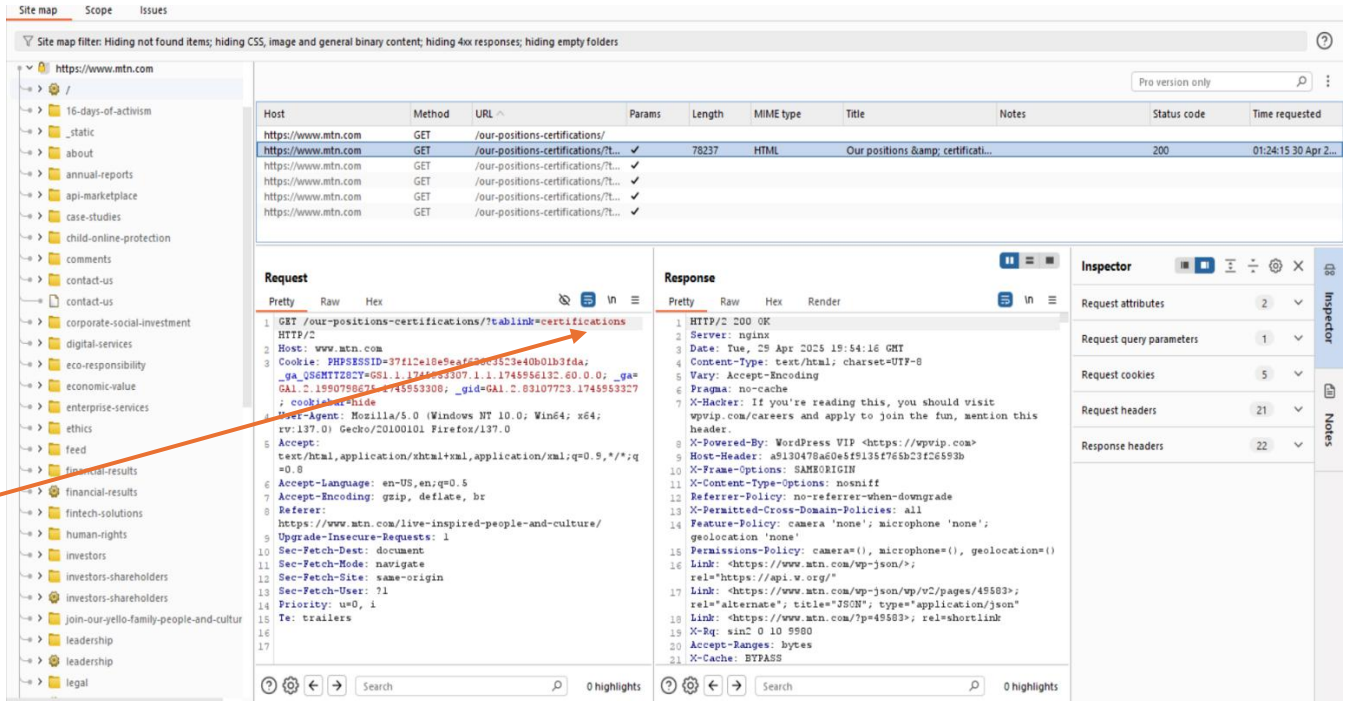
- Manual check using Burp Suite
- OWASP ZAP
- Webhook.site for HTTP request interception
- WaybackURL for accessing historical URLs
- FFuf for fuzzing potential SSRF endpoints

3.2 Steps Taken

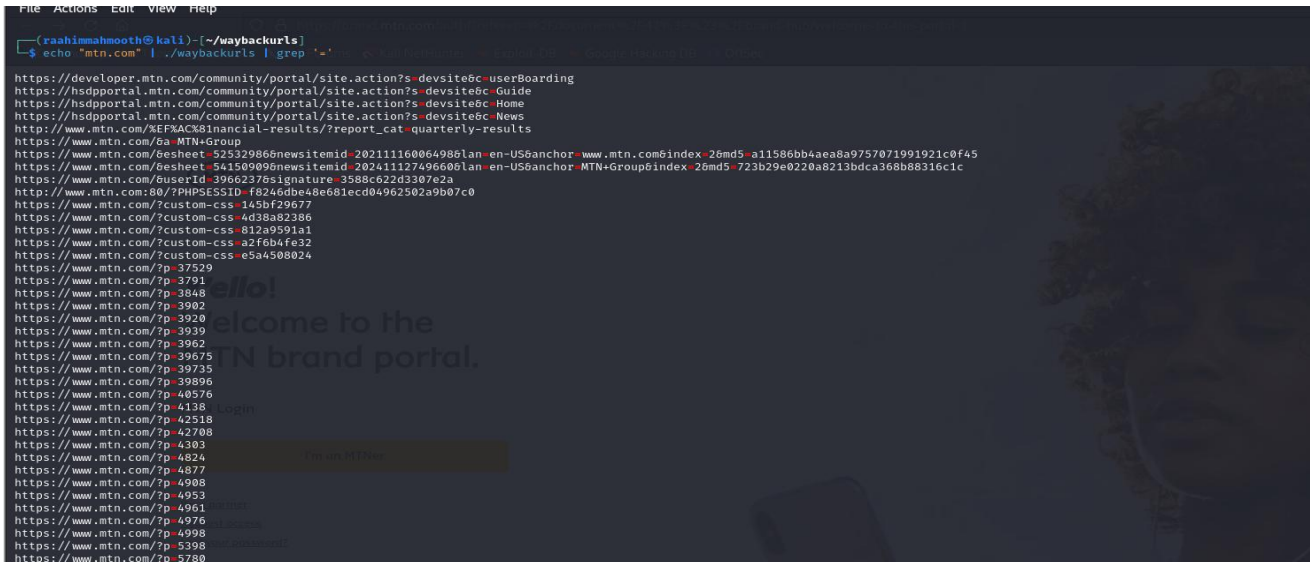
1. **Initial ZAP Scan:** Performed a basic scan with ZAP to detect **SSRF** vulnerabilities in the application.

Nothing found related to SSRF

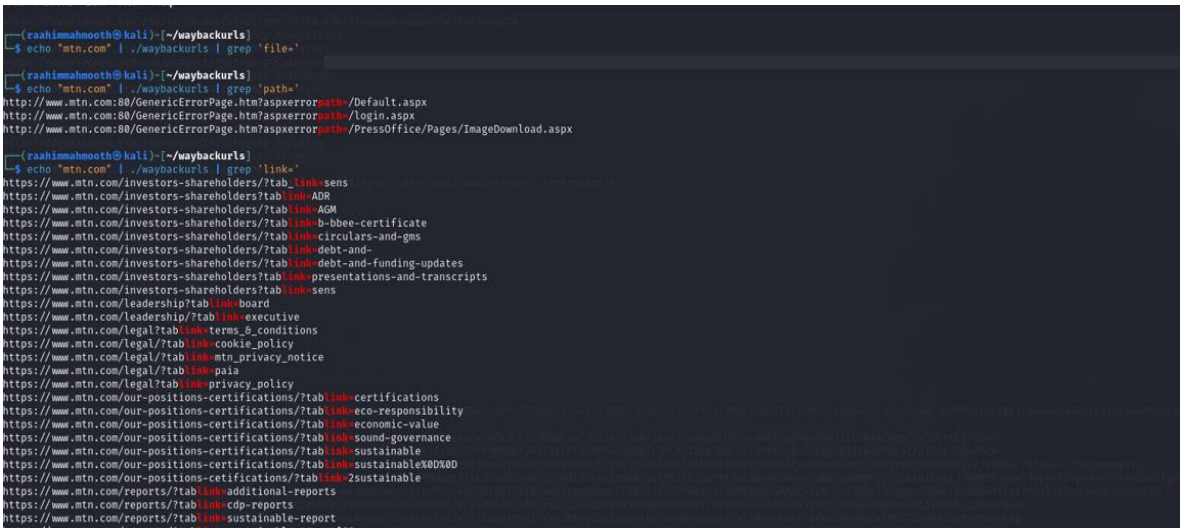
2. **Manual Check:** Manually intercepted requests and examined GET requests for endpoints with potential SSRF vulnerabilities.
    - **Endpoint Discovered:** `/our-positions-certifications/?tablink=certifications` - This **endpoint included a parameter that might be prone to SSRF.**



3. **WaybackURL Usage:** Utilized WaybackURL to retrieve historical URLs from the site but it results much overwhelming unable to **fetch SSRF prone parameters**



4. focusing on parameters such as `url=`, `path=`, and `link=`, `file=` which can be targets for SSRF.
    - Found a URL endpoint: `/wp-json/oembed/1.0/embed?url=https://www.mtn.com/2africa-now-the-longest-subsea-cable-in-the-world/&format=xml`, which is a WordPress oEmbed endpoint, potentially vulnerable to SSRF if the server fetches and embeds external URLs.



4. Vulnerability Description

SSRF occurs when a web application is tricked into sending requests to unintended resources, potentially internal systems or third-party services, on behalf of the attacker. The vulnerability was explored in two main parts:

1. **GET /our-positions-certifications/?tablink=certifications**: This endpoint takes a URL parameter and might fetch remote resources, opening the door for SSRF.
2. **WordPress oEmbed Endpoint**: `/wp-json/oembed/1.0/embed?url=https://www.mtn.com/2africa-now-the-longest-subsea-cable-in-the-world/&format=xml` - This endpoint fetches external URLs to embed content, making it a likely candidate for SSRF attacks if the input isn't properly validated.

5. Affected Component

1. **Endpoint :** *GET /our-positions-certifications/?**tablink**=certifications*
2. **Endpoint :** */wp-json/oembed/1.0/embed?**url**=https://www.mtn.com/2africa-now-the-longest-subsea-cable-in-the-world/&format=xml*

6. Impact Assessment

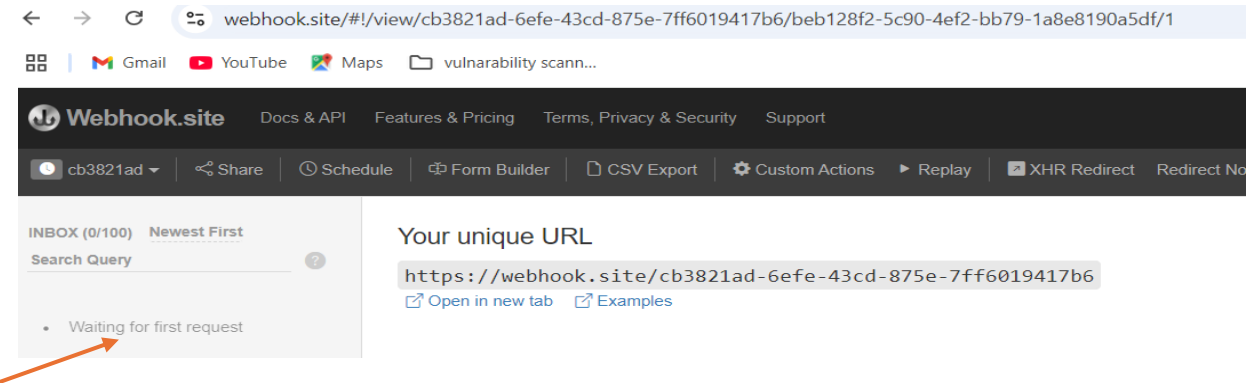
1. **SSRF Vulnerability:** If successful, an SSRF attack could allow an attacker to access internal services or resources that are not meant to be publicly accessible. This can lead to data leaks, unauthorized access to internal systems, or even the ability to interact with sensitive internal resources.
2. **WordPress oEmbed Endpoint:** This endpoint, when misused, could potentially allow attackers to fetch arbitrary resources from the internal network, which could then be exploited to gain unauthorized access or cause disruptions in service.

7. Proof of Concept (PoC)

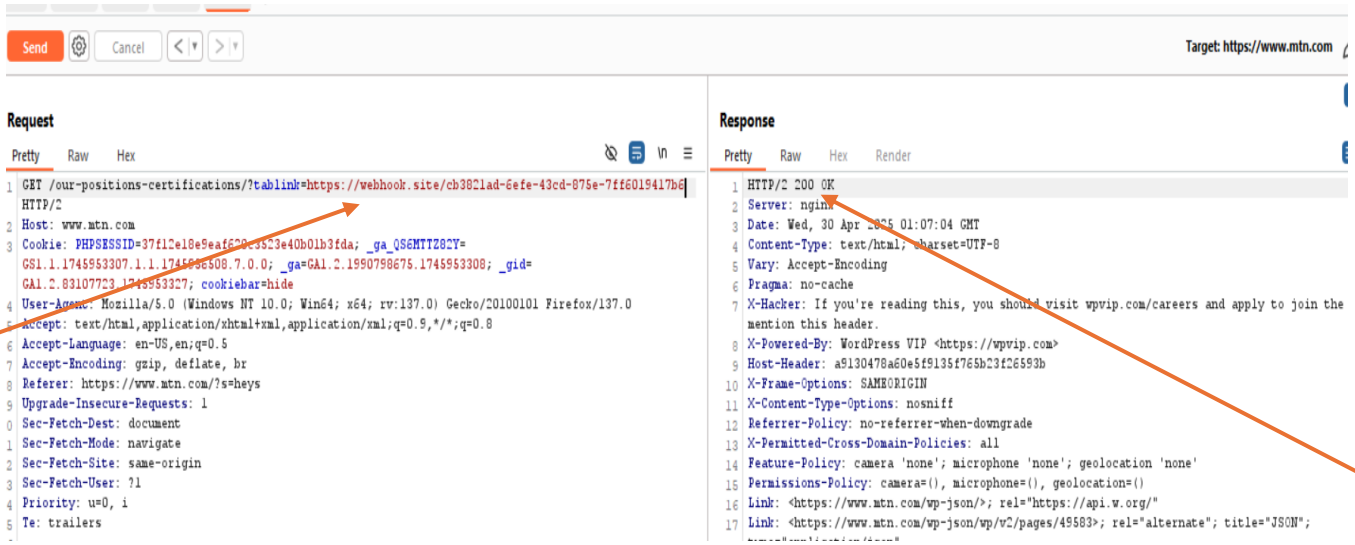
7.1 Manual Testing

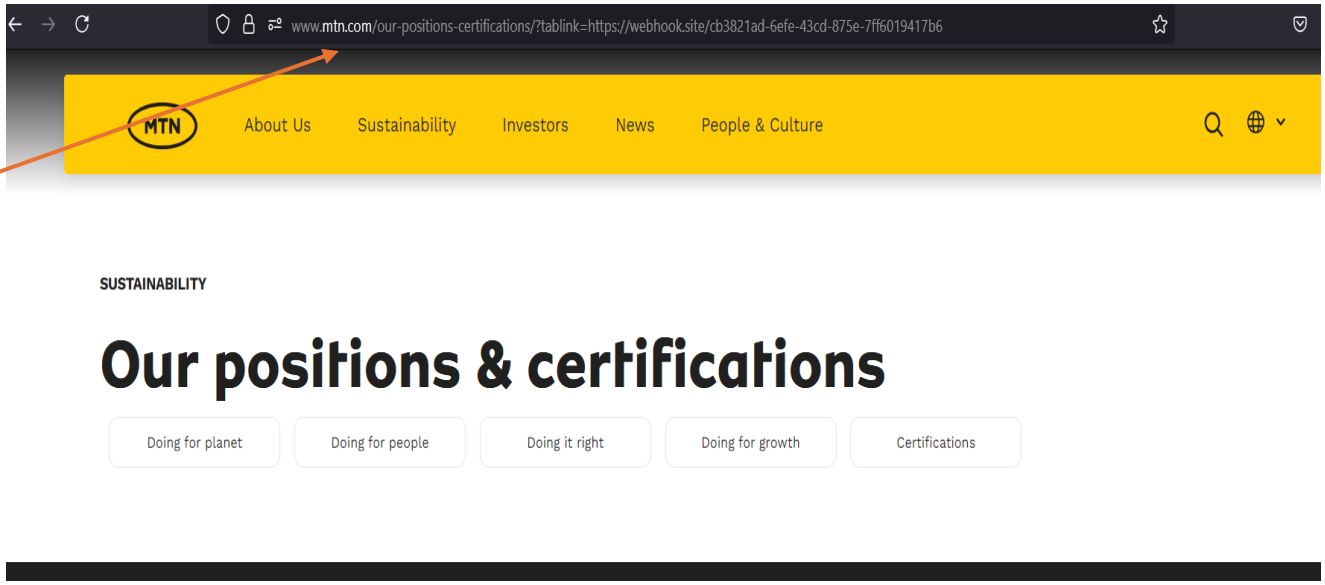
- **Testing *`/our-positions-certifications/?tablink=certifications`*:** Replaced the `tablink` parameter with a webhook URL obtained from Webhook.site. However, although the response was 200 OK, the request was sanitized by the backend, preventing SSRF.

webhook URL obtained from Webhook.site(act as my server)



Replaced the `tablink` parameter with a webhook URL

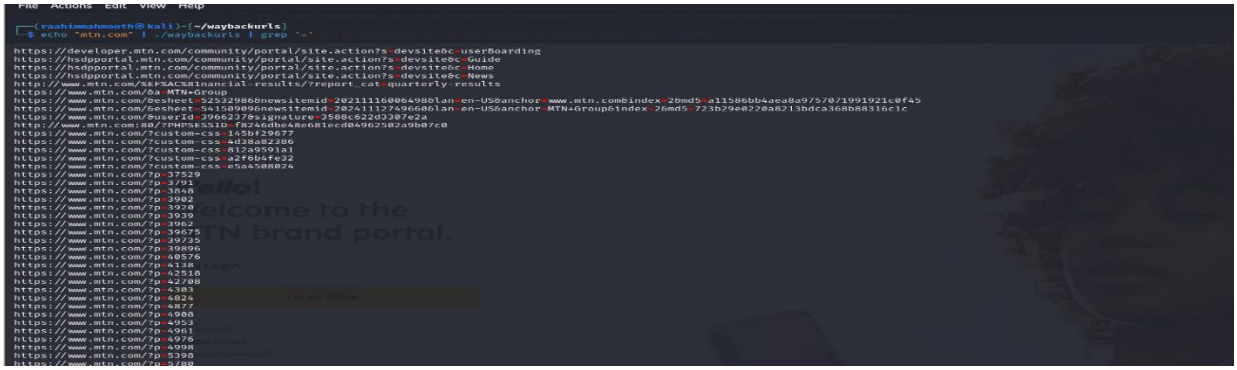




As a result no request was present in the site which tells the request was sanitized by the backend, preventing SSRF. Although the response was 200 OK

7.2 Automated Testing

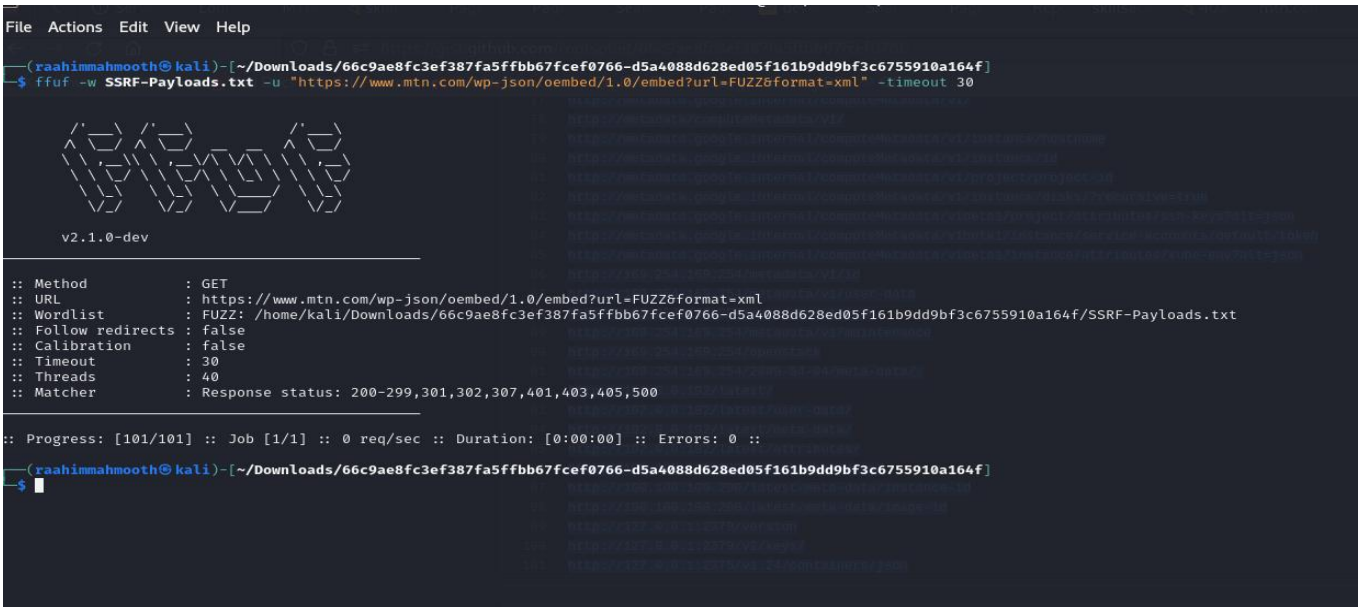
- **WaybackURL & FFuf Usage:**
  - **Tested** *https://www.mtn.com/wp-json/oembed/1.0/embed?url=<>* :
  - Used WaybackURL to retrieve **historical URLs** and found a **potentially vulnerable endpoint.**



- But testing all endpoint it takes some months but in this assessment I was randomly choosing a "*url=" parameter* that seems to **high prone to SSRF**

- Conducted fuzz testing using FFuf with the payload list *`SSRF-Payloads.txt`*:

```
ffuf -w SSRF-Payloads.txt -u "https://www.mtn.com/wp-json/oembed/1.0/embed?url=FUZZ&format=xml" -timeout 30
```



- However, all attempts to trigger SSRF failed, indicating proper sanitization and no SSRF vulnerability in this endpoint.

8. Proposed Mitigation

- **Implement Input Validation:** Ensure all parameters that accept URLs or external resources are strictly validated before making requests. Implement proper whitelisting or URL sanitization.
- **Use Firewall and Network Segmentation:** Block outgoing requests to internal resources or sensitive domains to prevent SSRF attacks.
- **Adopt Security Mechanisms:** Employ additional protections like allowing only specific external services to be fetched or using network-level controls to limit access.

9. Conclusion

After testing multiple endpoints over some time, **no SSRF vulnerabilities were found in the application**. However, proper URL sanitization should still be implemented in case new endpoints are introduced in the future. Furthermore, additional testing with Burp Suite Professional may yield more thorough results. But testing all might possibly give valuable hook.

10. References

- OWASP SSRF : *https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery*
- CVE-2020-15778: **SSRF vulnerability in WordPress oEmbed**