Monitored machine

Ali BA FAQAS

March 10, 2024

1 About Montiored

Montiored is a Linux machine of medium difficulty, which features a Nagios IX application. The user flag was obtained by exploiting a SQL vulnerability in the database to retrieve the API key. This key was then used to add a new user with administrative privileges to the database. The root flag was simpler to obtain, as it only required utilizing a script that the user could run with sudo privileges. This script had the ability to manage services running on the machine.

2 User Flag

2.1 Nmap

The initial step, as always, involves scanning ports and services. We used the -sV option to identify the software versions associated with the open ports. The scan results are as follows:

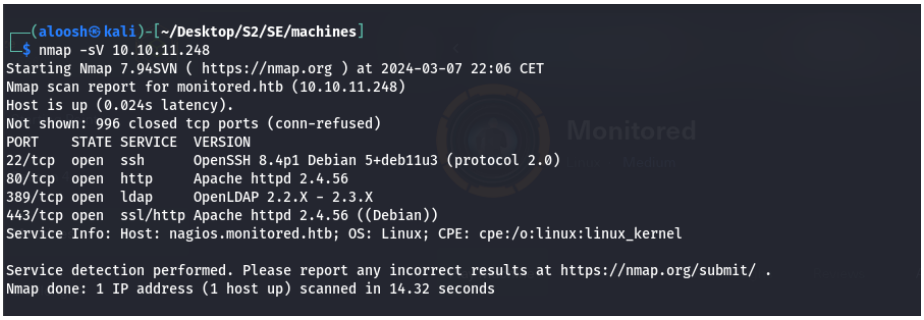


Figure 1: Nmap Scan Results.

The scan revealed four open ports:

1. SSH (Port 22)
2. HTTP (Port 80)

3. LDAP (Port 389)

4. HTTPS (Port 443)

From these results, we inferred that a web application was running using HTTPS. The other ports appeared secure and not exploitable at this stage.

Upon searching for the machine's IP address, we were redirected to http://nagios.monitored.htb/, which displayed an error page. To resolve this, we added the IP and host to our /etc/hosts file on our local machine, which made the webpage accessible.

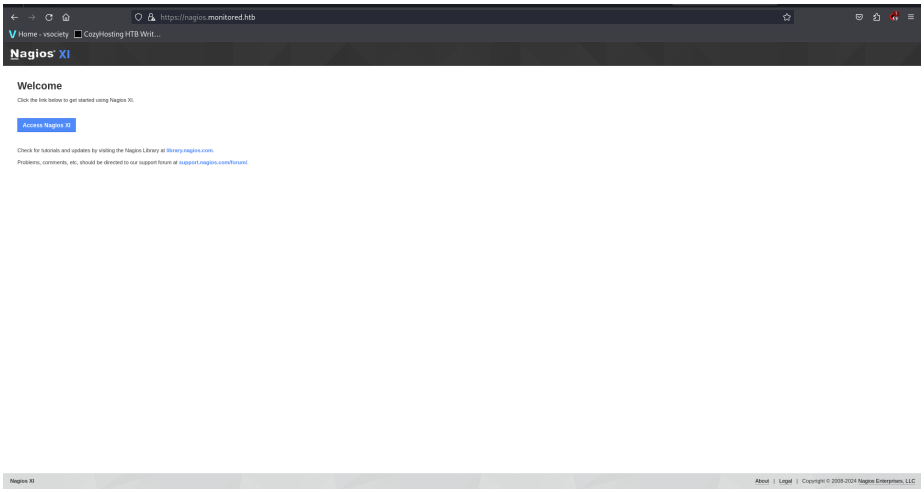


Figure 2: Main Page.

The main page contained three links: a login page link and two links to tutorial and support pages. Upon navigating to the login page, we found no default credentials, no opportunities for brute force, and no applicable CVEs. Therefore, we initiated directory enumeration.

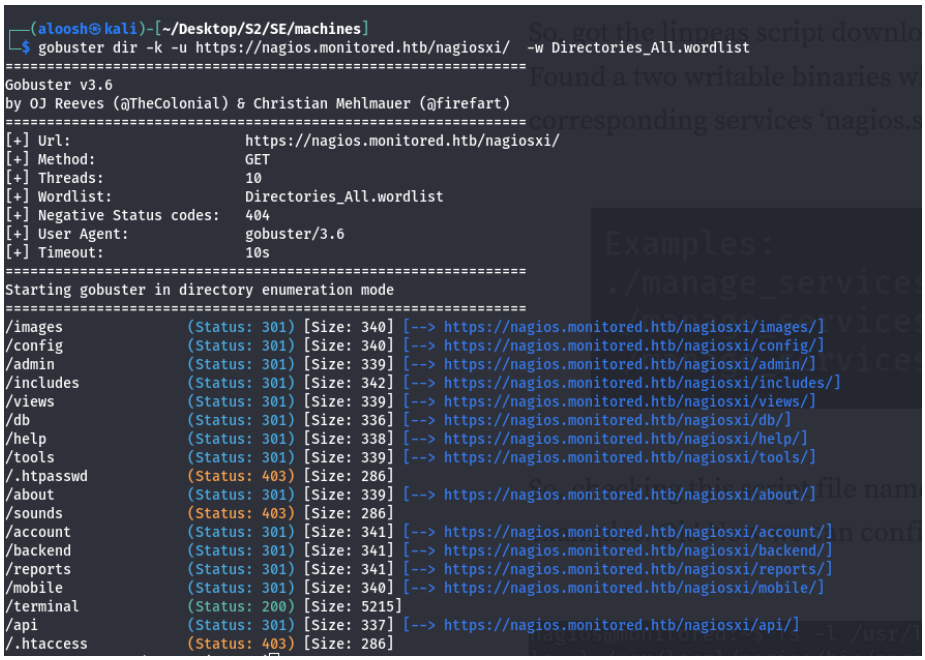


Figure 3: Directory Enumeration.

The enumeration revealed one page that responded with a 200 status code, but it was a terminal page that required login credentials. We continued enumerating the pages that responded with a 301 status code. All of them led to an endpoint, except for the API page, which revealed two additional pages: includes and v1.

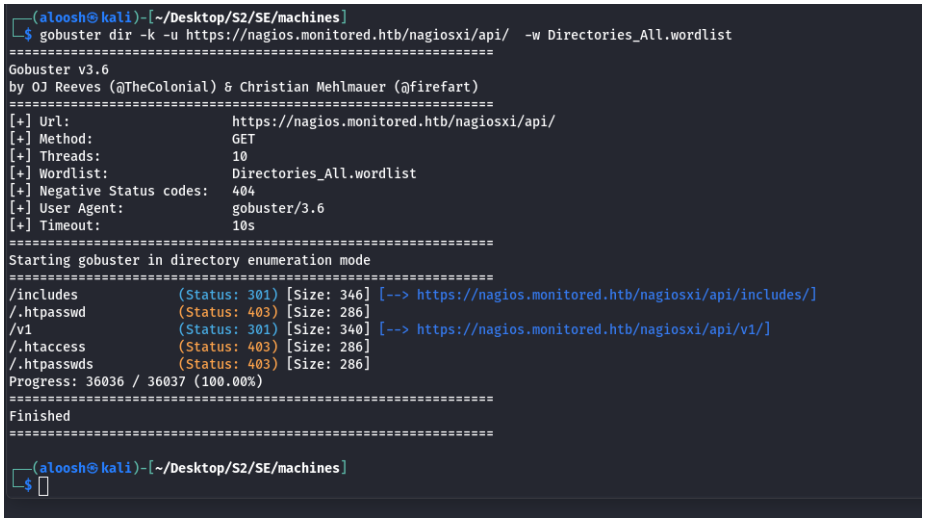


Figure 4: Directory Enumeration Continued.

Since the includes page was an endpoint, we focused our investigation on v1.

Figure 5: Investigating v1.

This led us to another endpoint, but it revealed that we needed a key. The v1 page also revealed an authenticate page, which seemed to require the API key for authentication.

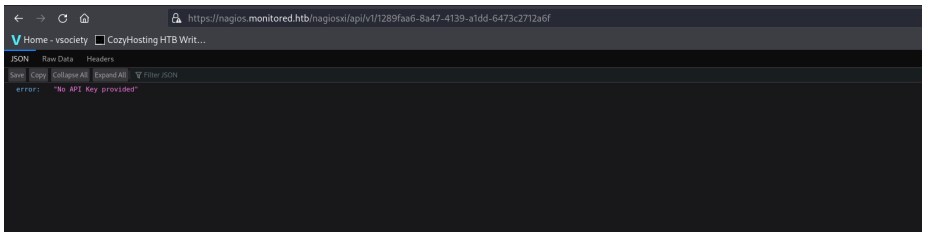


Figure 6: v1 Page.

The authenticate page accept only POST requests.

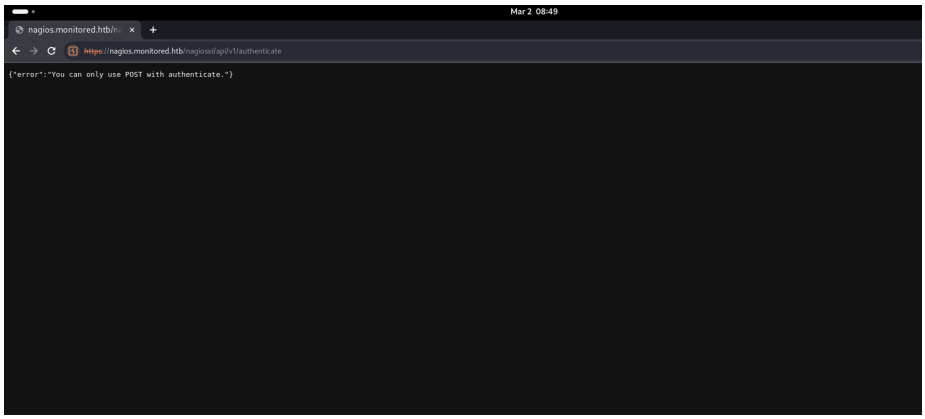


Figure 7: Authenticate Page.

At this point, we had exhausted all options in the directory enumeration section. We then attempted to locate any subdomains or vhosts, but found none.

Following the initial steps, we proceeded with a UDP Nmap scan. While the majority of popular services on the Internet operate over the TCP protocol, UDP services are also widely used. DNS, SNMP, and DHCP (registered ports 53, 161/162, and 67/68) are among the most common. However, because UDP scanning is generally slower and more challenging than TCP, it is often overlooked.

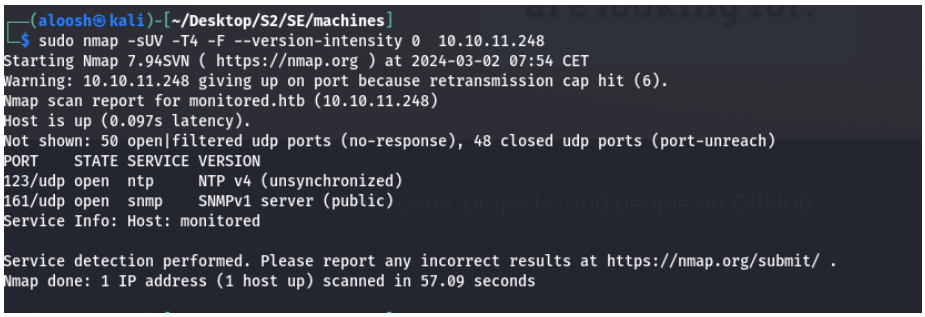


Figure 8: UDP Nmap Scan Results.

We discovered that the SNMP port was open. The Simple Network Management Protocol (SNMP) is a network protocol used for monitoring and managing devices on a network. It can reset a password on a network device, change its baseline configuration, or request a report on the network's bandwidth usage.

We used the snmpwalk command to retrieve a tree of information from the SNMP-enabled device and redirected the output into a file. This revealed a username, 'svc', and its corresponding password.

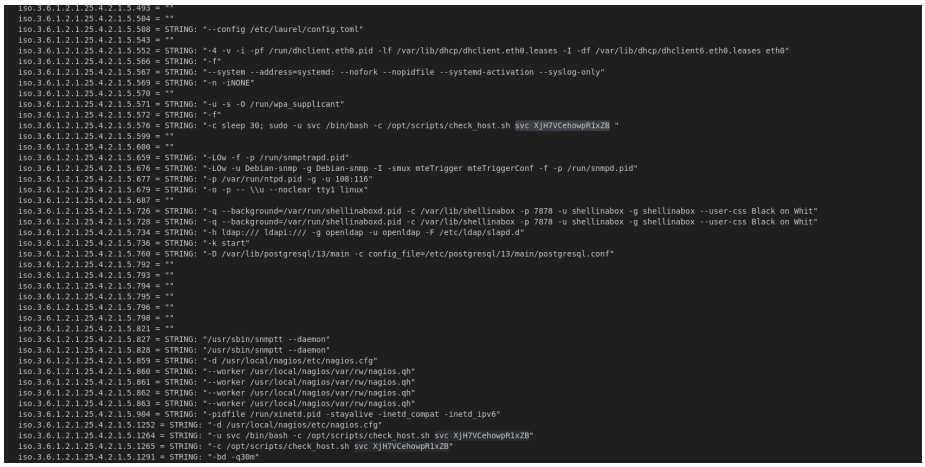


Figure 9: SNMP Output.

We attempted to log in using these credentials, but they did not work for the web application. We then considered the possibility of them being API credentials.

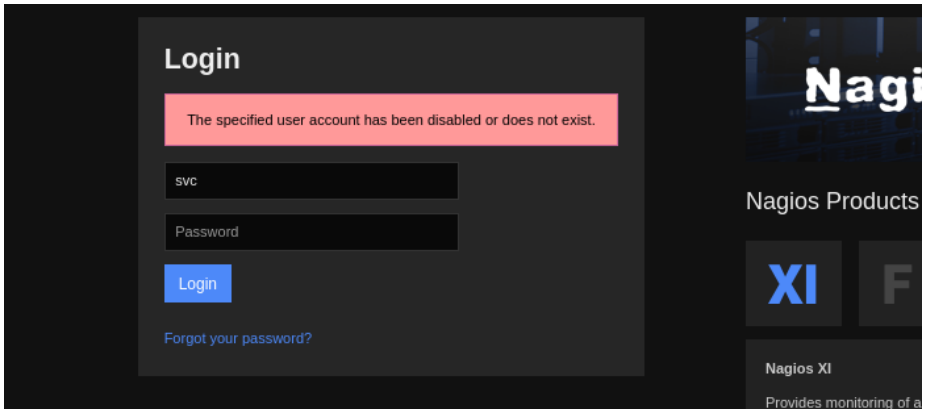


Figure 10: Attempt to Use 'svc' Credentials.

We needed to authenticate these credentials, but as we had observed earlier, only POST requests were permitted. Using Burp Suite, we sent a POST request with the 'svc' credentials. Although the connection failed, we received an authentication token, which we suspected could be the API token.

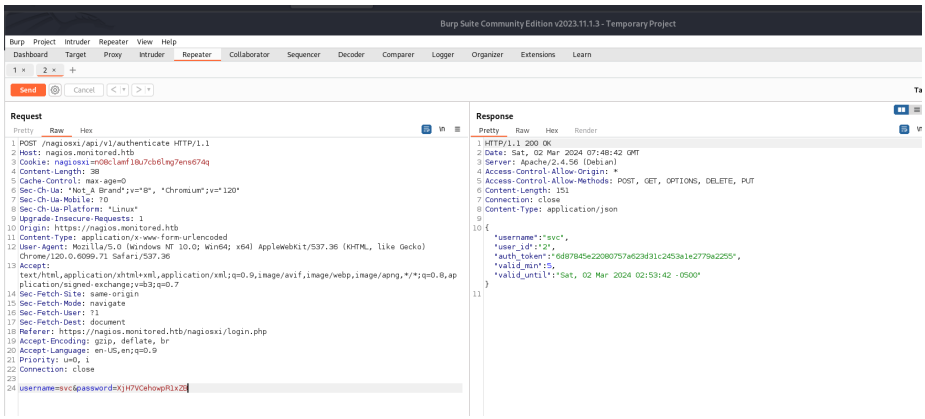


Figure 11: POST Request with 'svc' Credentials.

We attempted to authenticate the page using this token, but it was unsuccessful. According to the tutorial found on the main page, this token could be used to authenticate the user's webpage as follows:

https://nagios.monitored.htb/nagiosxi/login.php?token=...

As shown in the following figure, we were logged in as 'svc'.

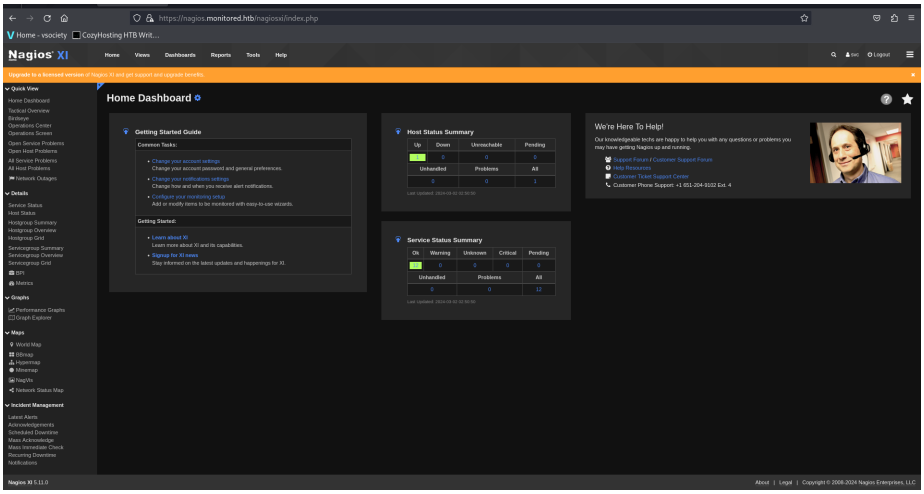


Figure 12: Logged in as 'svc'.

Upon navigating the page, we found that 'svc' was a regular user and we could not obtain a shell with our current user privileges. It was time to search for other CVEs that could potentially elevate our privileges. We found an article discussing the three most known CVEs for privilege escalation.

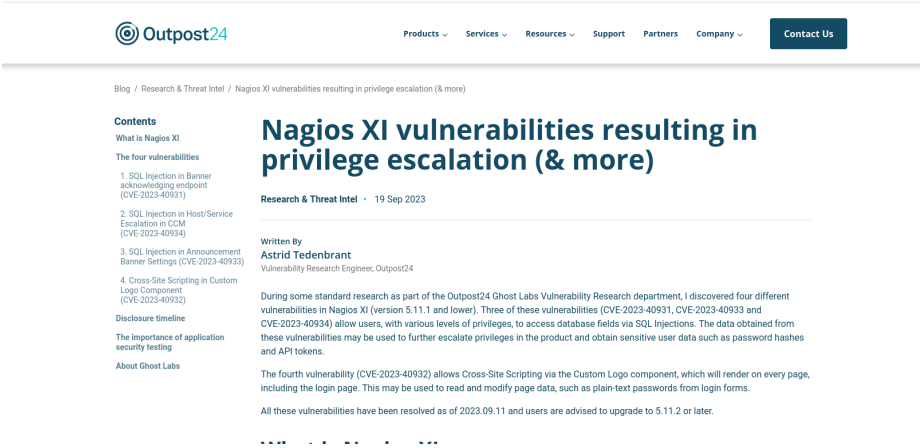
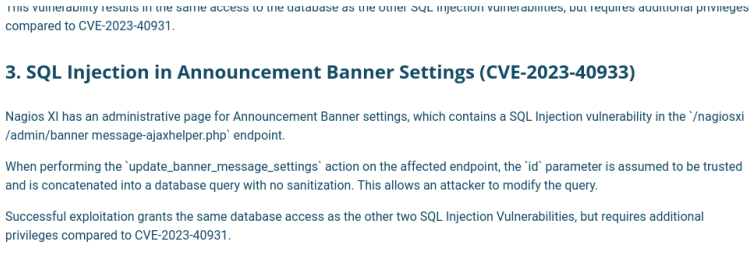


Figure 13: Known CVEs for Privilege Escalation.

One vulnerability that caught our attention was a SQL Injection vulnerability in the Announcement Banner settings of Nagios XI, identified as CVE-2023-40933. This vulnerability exists in the /nagiosxi/admin/banner message-ajaxhelper.php endpoint. When performing the update_banner_message_settings action on this endpoint, the 'id' parameter is assumed to be trusted and is directly concatenated into a database query without any sanitization. This lack of sanitization allows an attacker to modify the query. If an attacker successfully exploits this vulnerability, they would gain the same level of database access as the other two SQL Injection vulnerabilities mentioned.


This vulnerability results in the same access to the database as the other SQL Injection vulnerabilities, but requires additional privileges compared to CVE-2023-40931.

3. SQL Injection in Announcement Banner Settings (CVE-2023-40933)

Nagios XI has an administrative page for Announcement Banner settings, which contains a SQL Injection vulnerability in the `/nagiosxi/admin/banner message-ajaxhelper.php` endpoint.

When performing the `update_banner_message_settings` action on the affected endpoint, the `id` parameter is assumed to be trusted and is concatenated into a database query with no sanitization. This allows an attacker to modify the query.

Successful exploitation grants the same database access as the other two SQL Injection Vulnerabilities, but requires additional privileges compared to CVE-2023-40931.

Figure 14: CVE Details.

Next, we decided to delve into the database. We executed a specific code with our current cookie session, which allowed us to extract data from the database.

Figure 15: Database Exploitation.

As can be seen, there are API tokens present.

Figure 16: API Tokens.

We now had an admin API token, but we were unsure of its use. We revisited the support and tutorial documents and found something intriguing.

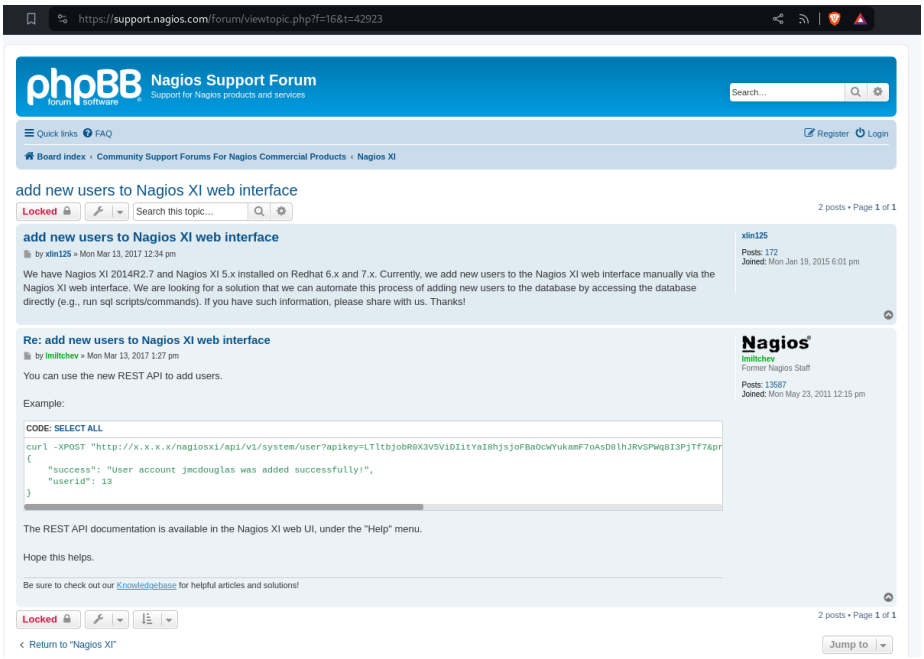


Figure 17: Adding a New User.

We then considered adding a new user with admin privileges. We ran the code found in the support page and successfully added a new user named "Ali" with admin privileges.

Figure 18: Adding User 'Ali'.

We logged in as Ali.

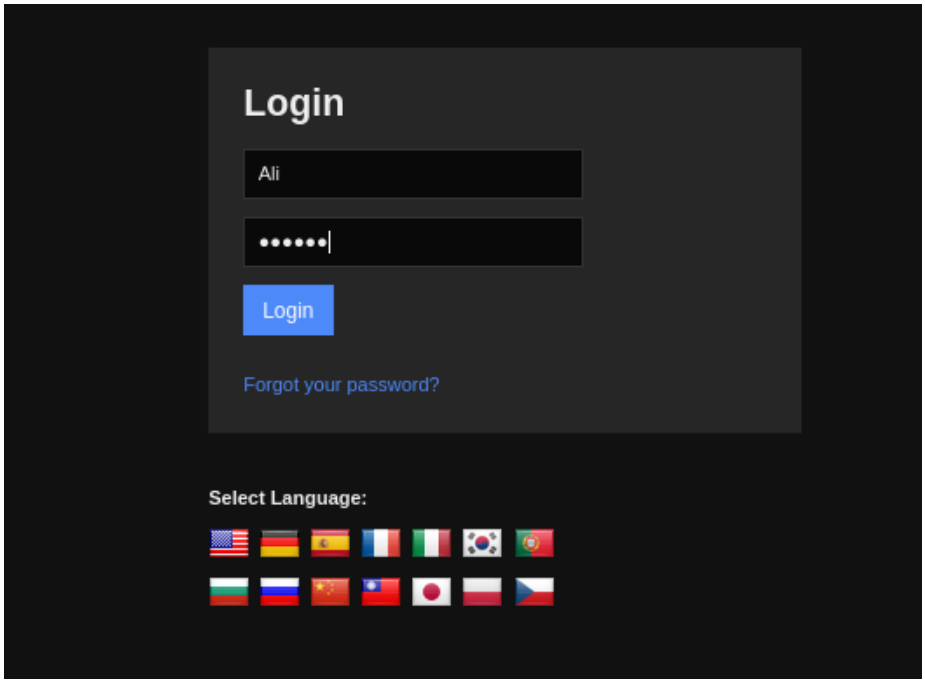


Figure 19: Logging in as 'Ali'.

We confirmed that we were logged in.

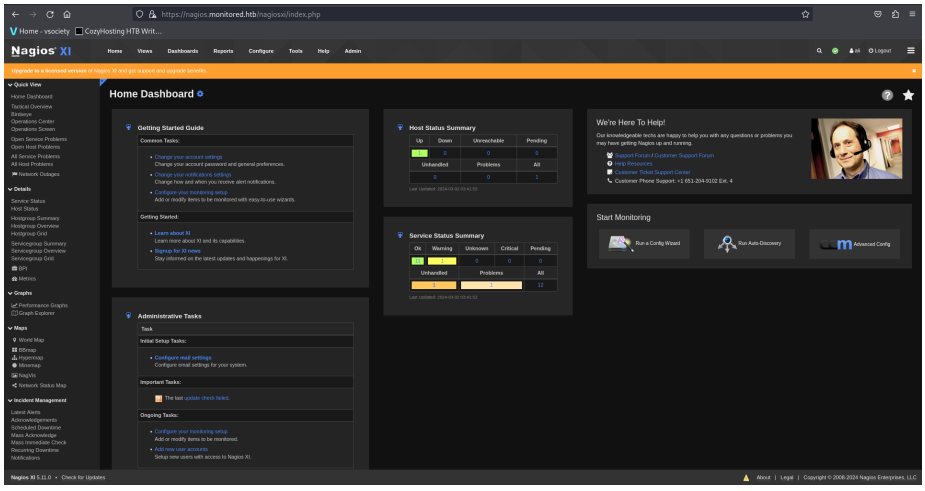


Figure 20: Logged in Confirmation.

The page had a configure section that was not available when we were logged in as the user 'svc'. We discovered that we could add commands, so we added our reverse shell command.

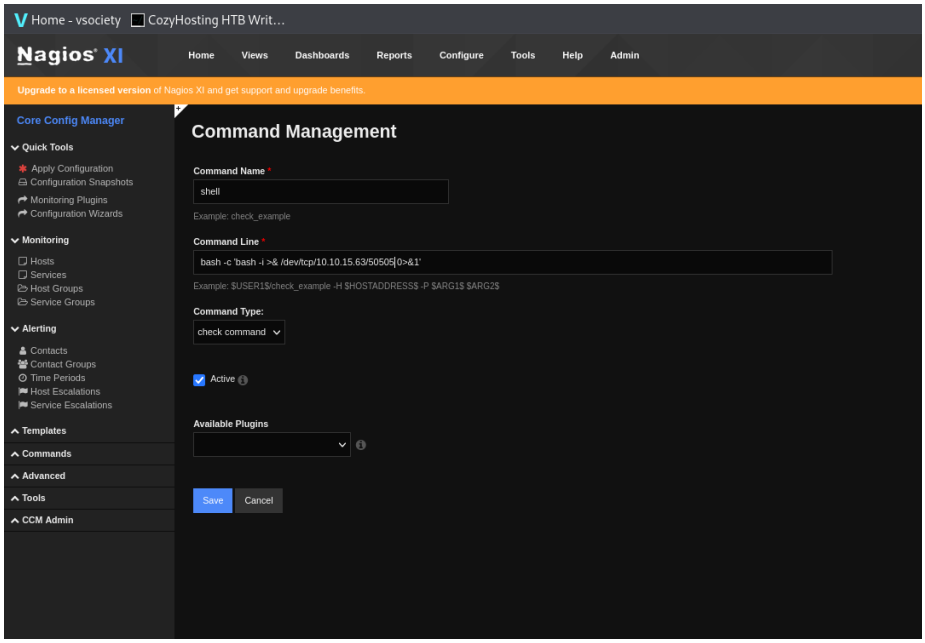


Figure 21: Adding Reverse Shell Command.

To execute the command, we needed to create a service for it and run it from there.

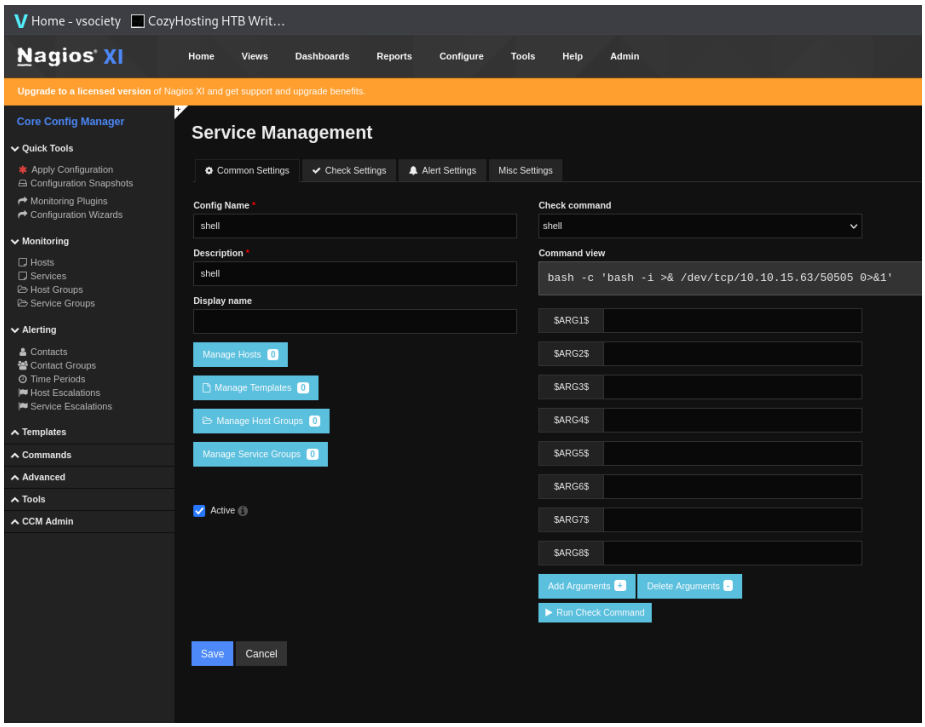


Figure 22: Adding Service for Command.

Finally, we obtained shell access for the user 'nagios'.

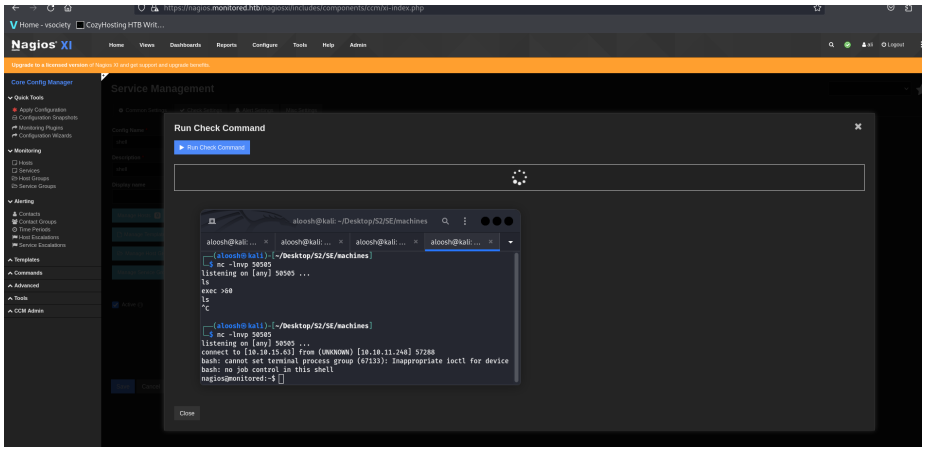


Figure 23: Shell Access for 'nagios'.

And here is the user flag!

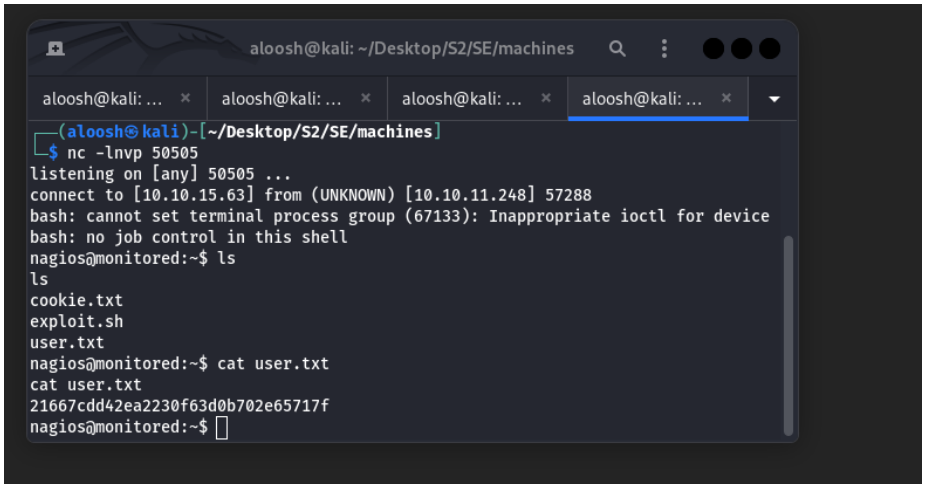


Figure 24: User Flag.

3 Root Flag

We began by examining the current repository, where we found a script named exploit.sh. However, this script was not useful.

Figure 25: Listing Files in Current Repository.

Next, we listed the sudo privileges for the current user.

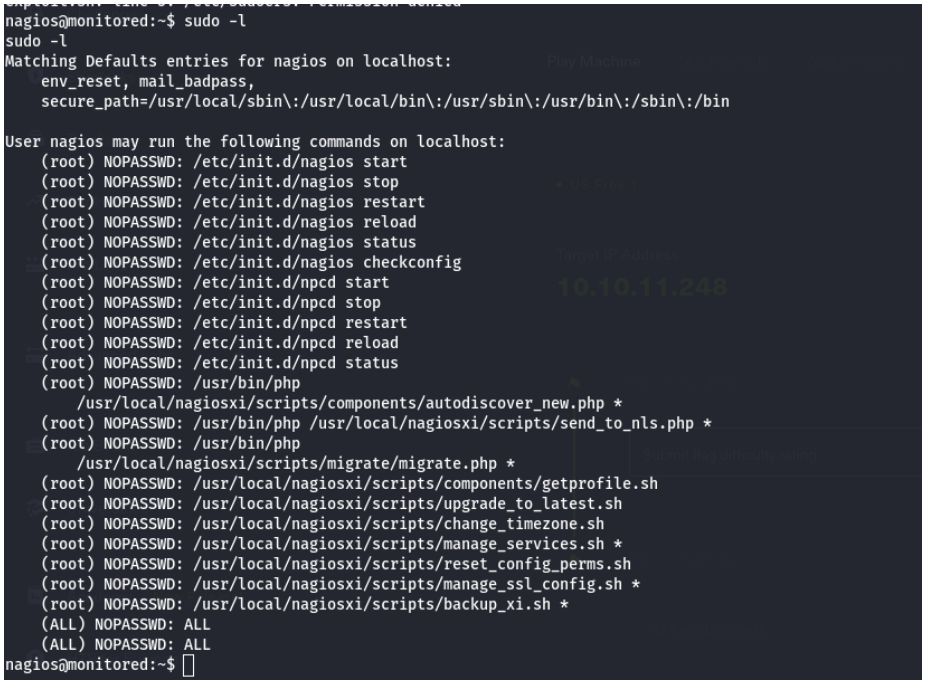


Figure 26: Listing Sudo Privileges.

We found that this user could run some scripts as sudo. Upon examining these scripts, we found the manage-services.sh script to be interesting. This script allowed us to start, stop, and restart listed services.

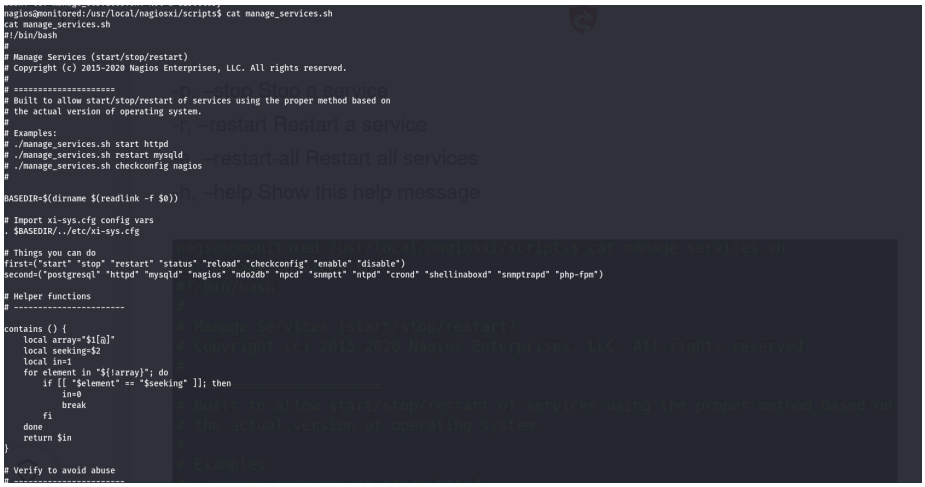


Figure 27: Manage Services Script.

We then used linpeas to perform a full scan and identify which services among those listed we could use. We found that the npcd service could be modified by us.

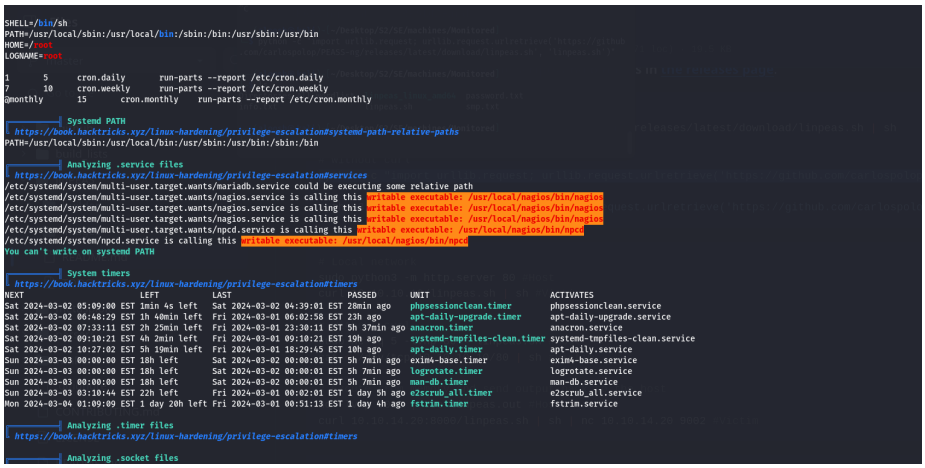


Figure 28: Linpeas Scan Results.

The straightforward approach was to delete npcd and replace it with a file of the same name containing our reverse shell code.

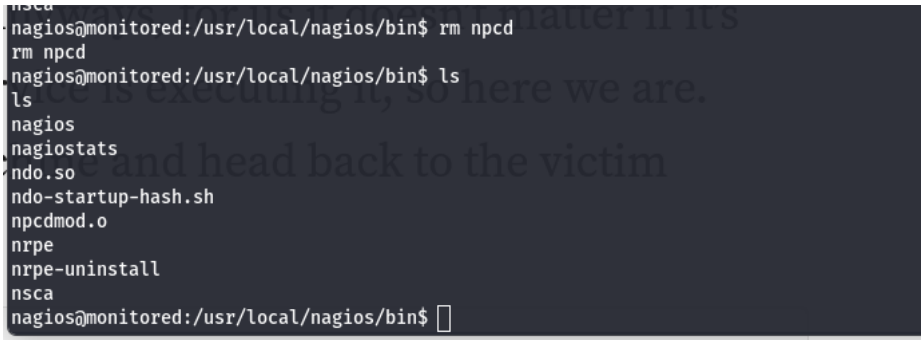


Figure 29: Deleting npcd.

here is the reverse shell used.

Figure 30: Gaining User Access.

By restarting the service using manage-services.sh, we gained root access. And here is the root flag!

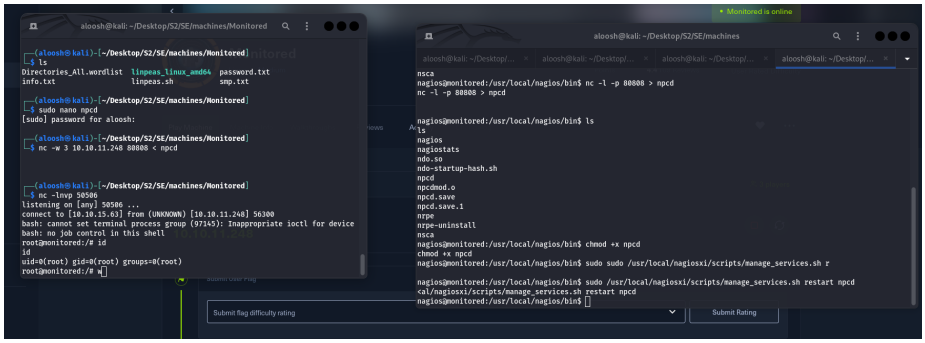


Figure 31: Root Flag.