Penetration Testing CA2
VM Image: South Park

Dean – B0009

Dxxxxxxxxxt of Ixxxxxxxxxxxxs
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxx

Word count: 10,385

BN311 Bachelor of Science in Computing
Digital Forensics & Cyber Security
Professional Penetration Testing
1/1/2019

Contents

1: Executive Summary

This report documents the investigation of the VM **(iii)** Image file 'South Park' as provided by our instructor Mark Cummins. The report will document the tools used, and steps taken to attempt to penetrate this server and where necessary provide recommendations to prevent such penetrations in the future.

2: Introduction

We are provided with some very basic information going into this project. We are given a recommended network IP **(iv)** address of '192.168.176.132/24' to assign to our VM. While this is strictly speaking not necessary, it does provide some clues as to the potential IP address of the VM we shall be attacking. It should at least establish the range we are dealing with.

This should prove useful in our initial probes into the server which we suspect may reveal some exploits or weak points we could use to aggressively attack the server. Our primary Tools will include (without being restricted to):

- Parrot OS
- Kali
- Windows 10
- Google / Google VM
- NetDiscover
- NMap
- FTP**(v)**
- VSFTPD**(vi)**
- OpenSSH
- FCrackzip
- Burp Suite
- Wireshark
- SSH**(v)**
- DirBuster
- Google Chrome
- Mozilla Firefox
- Way Back Machine
- Armitage
- Metasploit framework
- John The ripper
- HashCat
- Eclipse IDE
- Python scripts
- Java scripts

2.1: Scope

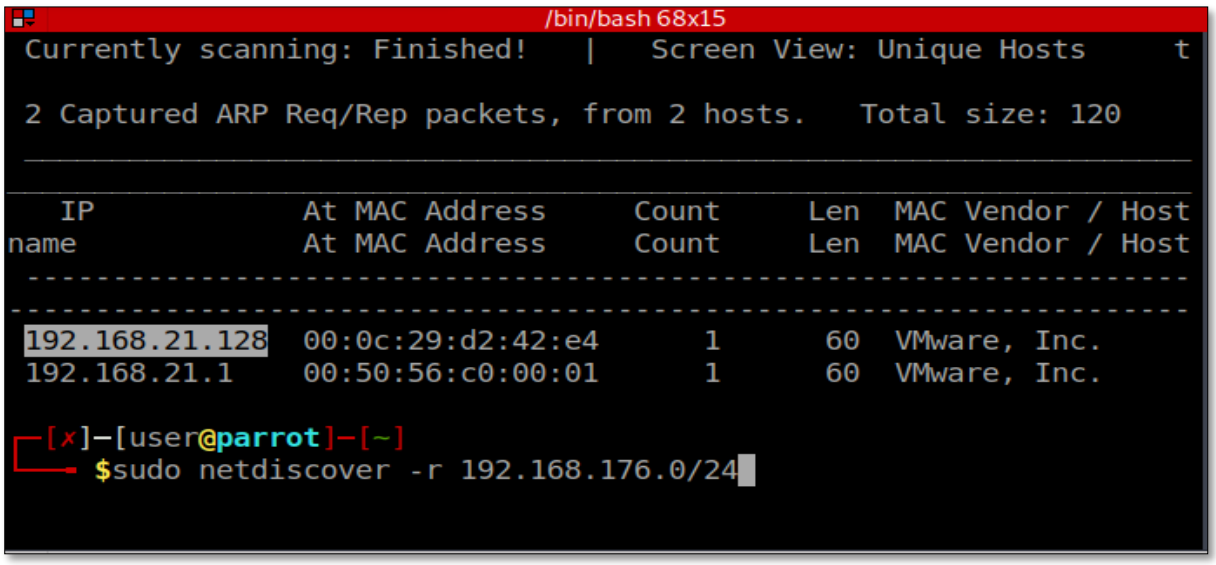
The VM server and any ports discovered therein shall be considered in scope. External links, where they can be used as recon material, via comments, videos, audio files or source code examination shall also be considered in scope. Attacks against any externally link sites or media shall be considered out of scope. Our instructor's laptop, should he leave it unattended shall, most definitely, be considered in scope.

2.2: Objectives

The objective is to establish an administrator level shell access to the server, recover the flags (9 main Flags & 1 bonus recon flag). We must demonstrate our knowledge and skills acquired during the course of our module and produce a professional document detailing the steps we took that were both successful and unsuccessful in a manner which would allow the reader to reproduce these steps, along-side what we learned from the entire process. Where possible, commands and/or screenshots of commands should be provided

3: NetDiscover

As we were provided with a general network range and we knew that the IP's were going to be dynamically assigned on the south park VM we decided to leave our own VM to connect with DHCP too. We assigned both our VMs to host only setting on VMWare. In order to ensure we have a connection to our south park VM we conducted a NetDiscover scan across the entire network range to see if the VM (Virtual Machine) was detected.



Net discovery on local network: ***sudo netdiscover -r 192.168.176/24***

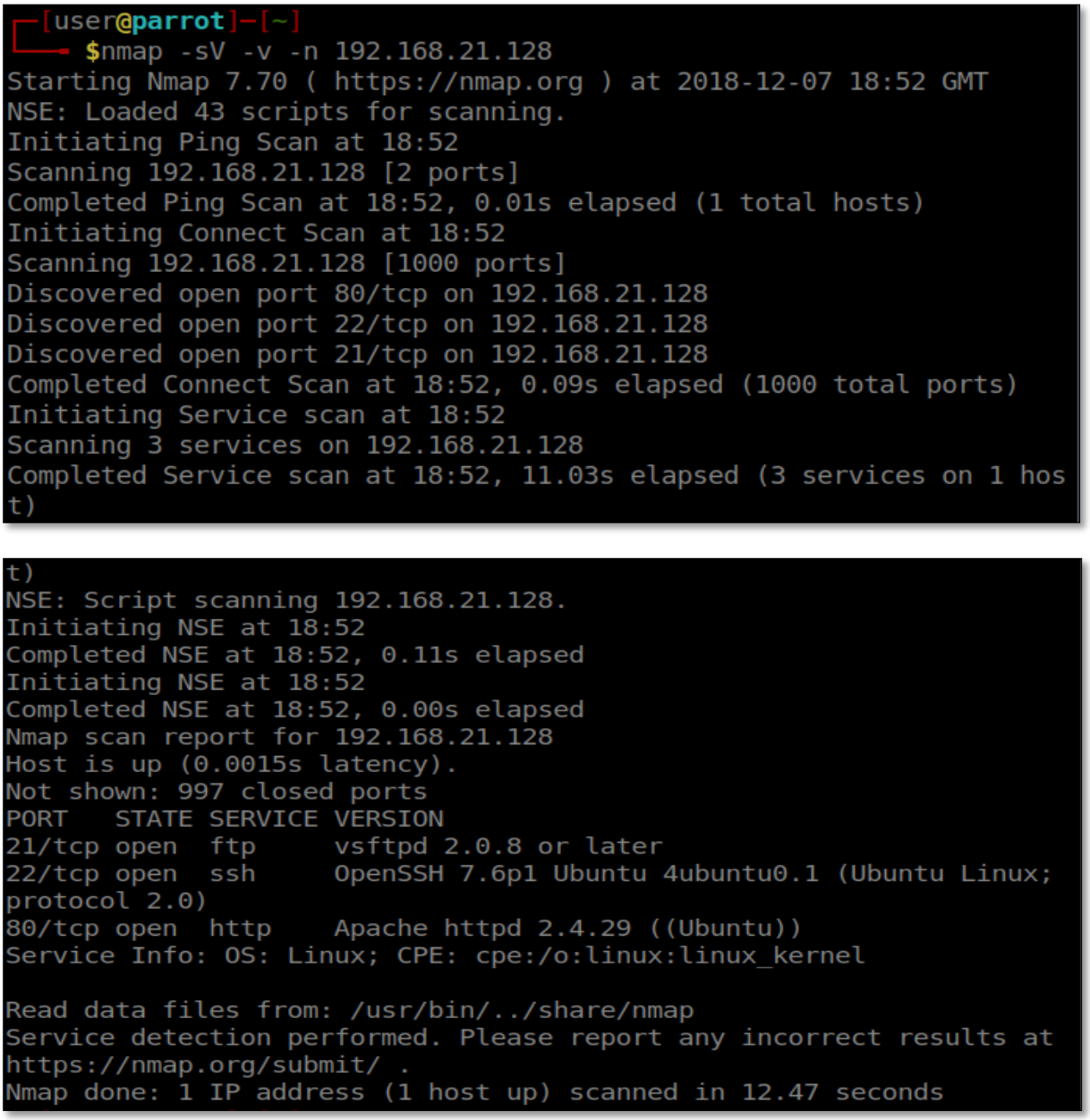
As you can see above, upon using NetDiscover an ARP scan was made and the VM was detected to have 192.168.21.128 as its dynamically assigned IP address.

Our Decision to ignore the recommendation regarding the IP, would prove costly in time later and necessitate us to manually subnet our VM network and have it assign DHCP issued IP addresses within a specific range. This will be covered at a later point.

4: NMap

The first step undertaken once we had discovered the dynamically assigned IP address of the VM was to begin mapping the ports using NMap. This was done in three stages, the first being a service scan of the 1000 most widely used ports. The second was a full TCP port scan. The last was a full UDP port scan.

The service scan of the most widely used ports return several port discoveries. They were as follows:



Service scan of most common TCP ports: ***nmap -sV -v -n 192.168.21.128***

As we see above, we have discovered a VSFTPD service with a version number of 2.0.8 or later running on port 80, an OpenSSH service with a version id of 7.6p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2) also running on port 80; and lastly an Apache web service with a version of 2.4.29((Ubuntu)) running on port 80.

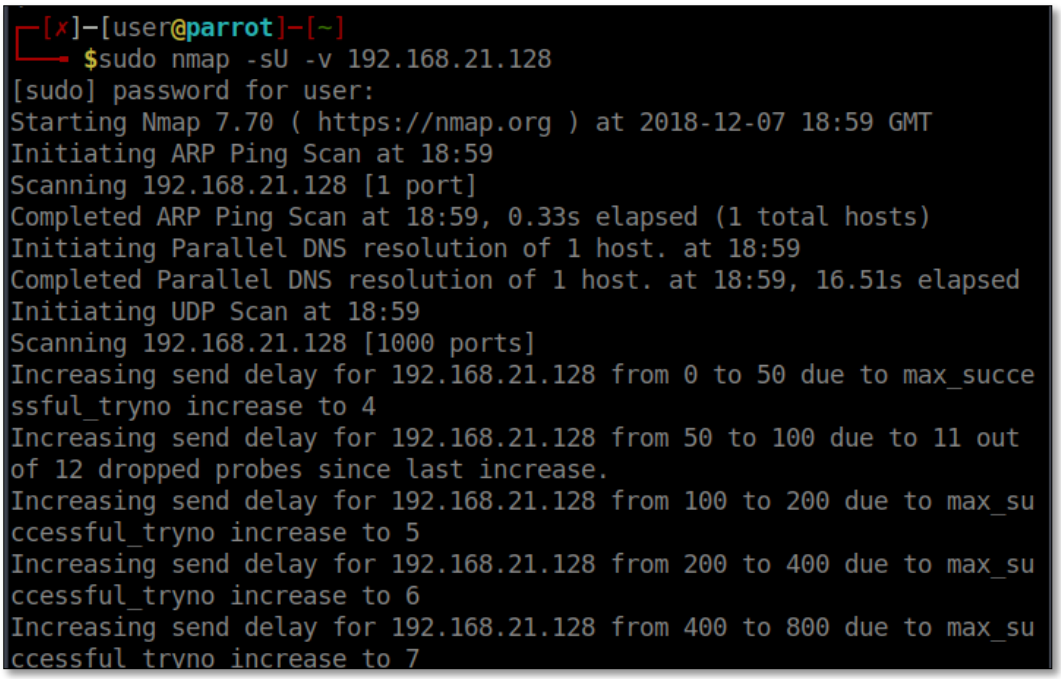
We expect all of these services might provide useful starting points, however we elected to proceed with our intensive port scans before investigating these services further. And so, we began our intensive TCP full port range scan.

```
└─$nmap -sV -p- -n  192.168.21.128
Starting Nmap 7.70 ( https://nmap.org ) at 2018-12-07 18:57 GMT
Nmap scan report for 192.168.21.128
Host is up (0.00036s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE VERSION
21/tcp    open  ftp     vsftpd 2.0.8 or later
22/tcp    open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux;
 protocol 2.0)
80/tcp    open  http    Apache httpd 2.4.29 ((Ubuntu))
33666/tcp open  unknown
1 service unrecognized despite returning data. If you know the service
/version, please submit the following fingerprint at https://nmap.org/
cgi-bin/submit.cgi?new-service :
SF-Port33666-TCP:V=7.70%I=7%D=12/7%Time=5C0AC282%P=x86_64-pc-linux-gnu
%r(N
SF:ULL,3890,"ID3\x03\0\0\0\0\0kCOMM\0\0\0\x1b\0\0\0\0\0\0\0www\.southp
arks
SF:tuff\.comCOMM\0\0\0\x1b\0\0\0XXX\0www\.southparkstuff\.comTIT2\0\0\
0\x1
SF:7\0\0\0www\.southparkstuff\.com\xff\xfb\x90\xc4\0\0\0\0\0\0\0\0\0\0
\0\0
SF:\0\0\0\0\0Xing\0\0\0\x0f\0\0\n\xdb\0\n\x078\0\x03\x06\t\x0b\x0e\x11
\x14
SF:\x16\x19\x1b\x1f!\$&\)\+\.0258;=\?ADFILOQTVX\[\]`begjloqtvy{}\x80\x
82\x
SF:85\x87\x8a\x8d\x90\x92\x95\x97\x9a\x9c\x9f\xa1\xa4\xa6\xa9\xab\xad\
xb0\
SF:xb2\xb5\xb7\xba\xbd\xc0\xc2\xc5\xc7\xca\xcc\xce\xd1\xd4\xd6\xd8\xdb
\xdd
SF:\xe0\xe3\xe6\xe7\xea\xed\xf0\xf2\xf3\xf6\xfa\xfd\0\0\0PLAME3\.99r\x
```

*Full Service scanof all TCP ports:**NMap -sV -p- -n 192.168.21.128***

This full TCP port scan revealed what appeared to be a broken service running on port '33666' which issues a stream of unidentifiable data. This was very odd behaviour. This service announced itself as a TCP service which raised a few questions as to how this service might be exploitable through either a browser or perhaps as an attempt to remotely connect via NetCat. Curiously, what appeared to be a website address was discovered in the data stream returned from the port scan. The website address appeared to be 'www.southparkstuff.com'.

Finally, we initiated our final full scan. The UDP port scan. This scan revealed some more potentially useful data which can be seen below.



UDP Port scan: ***sudo nmap -sU -v 192.168.21.128***

```
UDP Scan Timing: About 5.68% done; ETC: 19:09 (0:08:35 remaining)
UDP Scan Timing: About 8.54% done; ETC: 19:11 (0:10:53 remaining)
UDP Scan Timing: About 11.43% done; ETC: 19:13 (0:11:45 remaining)
UDP Scan Timing: About 31.00% done; ETC: 19:15 (0:11:03 remaining)
UDP Scan Timing: About 37.49% done; ETC: 19:16 (0:10:12 remaining)
UDP Scan Timing: About 43.50% done; ETC: 19:16 (0:09:22 remaining)
UDP Scan Timing: About 48.96% done; ETC: 19:16 (0:08:31 remaining)
UDP Scan Timing: About 54.31% done; ETC: 19:16 (0:07:40 remaining)
UDP Scan Timing: About 59.77% done; ETC: 19:16 (0:06:47 remaining)
UDP Scan Timing: About 65.03% done; ETC: 19:16 (0:05:55 remaining)
UDP Scan Timing: About 70.39% done; ETC: 19:16 (0:05:02 remaining)
UDP Scan Timing: About 75.77% done; ETC: 19:17 (0:04:08 remaining)
UDP Scan Timing: About 80.82% done; ETC: 19:17 (0:03:17 remaining)
UDP Scan Timing: About 85.97% done; ETC: 19:17 (0:02:24 remaining)
UDP Scan Timing: About 91.01% done; ETC: 19:17 (0:01:33 remaining)
UDP Scan Timing: About 96.07% done; ETC: 19:17 (0:00:41 remaining)
Completed UDP Scan at 19:18, 1084.31s elapsed (1000 total ports)
Nmap scan report for 192.168.21.128
Host is up (0.00048s latency).
Not shown: 997 closed ports
PORT     STATE         SERVICE
68/udp   open|filtered dhcpc
631/udp  open|filtered ipp
5353/udp open|filtered zeroconf
MAC Address: 00:0C:29:D2:42:E4 (VMware)

Read data files from: /usr/bin/../share/nmap
Nmap done: 1 IP address (1 host up) scanned in 1101.28 seconds
           Raw packets sent: 1555 (45.272KB) | Rcvd: 66966 (2.752MB)
```

As we see above, the UDP scan revealed three further services. The first being an open filtered DHCPC service running on UDP port 68, the second being an open filtered IPP service (printer related service) running on UDP port 631; and finally, an open filtered ZEROCONF service running on UPD port 5353. The ZEROCONF and DHPC services are not expected to be highly exploited but will be explored later in the penetration test. The IPP service might prove exploitable.

4.1: TCP (vi) Services

- VSFTPD 2.0.8 TCP
- Open SSH 7.6p1 TCP
- Apache HTTP 2.4.29
- Unknown(33666) TCP

4.2: UDP(vii) Services

- DHCPC Service

- IPP Service
- ZeroConf

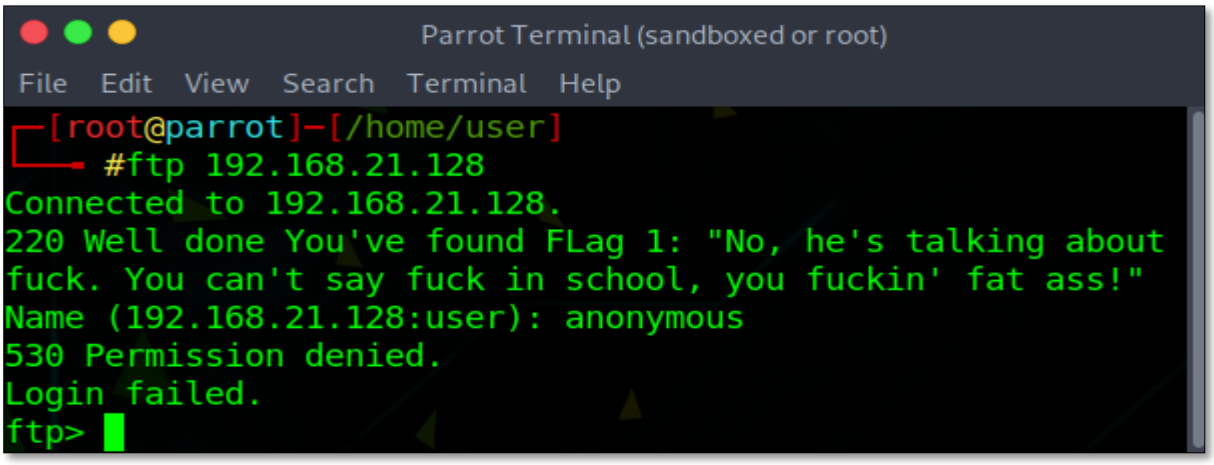
5: OpenVas

Before manually beginning to investigate the services, an aggressive vulnerability scan was conducted against the target at 192.168.21.130(Dynamic IP address has changed) to discover if any high or moderate level vulnerabilities could be found that might be exploited to give a shell access onto the server. Only one very minor vulnerability was found, and it provided no route to exploit the system; instead allowing a remote party to potentially discover the uptime of the server. This TCP timestamp vulnerability mere allowed the attacker to discover the uptime of the server.

Here we see the results of OpenVas scan.

6: Finding Flag 1 Via FTP

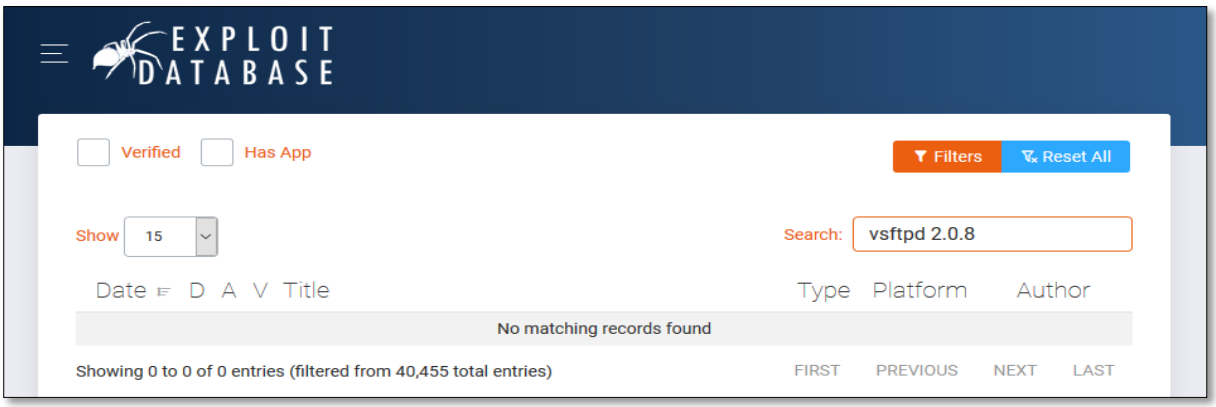
The first thing we attempted to do having moved on to service examination, was connect to the server via FTP as anonymous running on the VM. Before we could log in we noticed we have discovered **Flag 1**. Although we discovered the first flag in a Banner message on the FTP service, we could not gain access to the FTP server using the anonymous account.



***Flag1** = "No, he's talking about fuck. You can't say fuck in school, you fuckin' fat ass!"*

*Attempted connection to ftp Server: **ftp 192.168.21.128***

We decided we would try to find more information using other services before we attempt to brute force our way into FTP. An attempt was made to seek out known vulnerabilities in VSFTPD 2.0.8 however none where found.

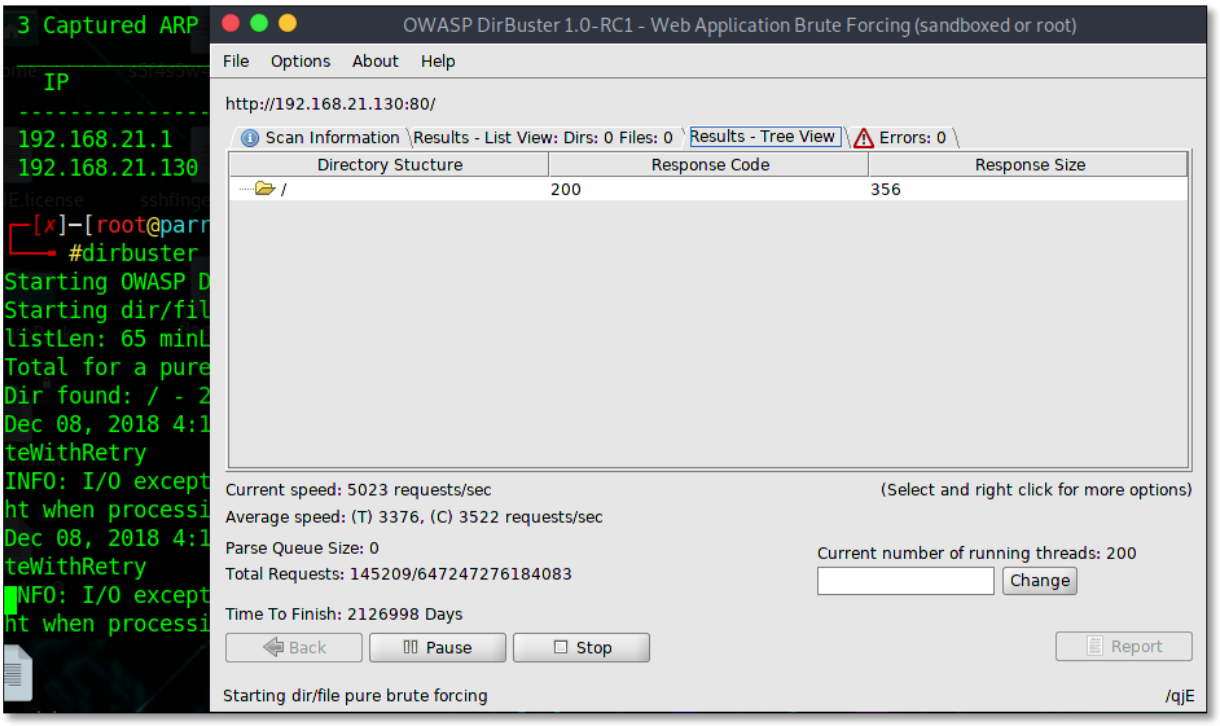


No exploits found for the VSFTPD service.

Next, we decided to investigate the webserver.

7: Web Server Investigation& the Finding of Flags 4 & 5

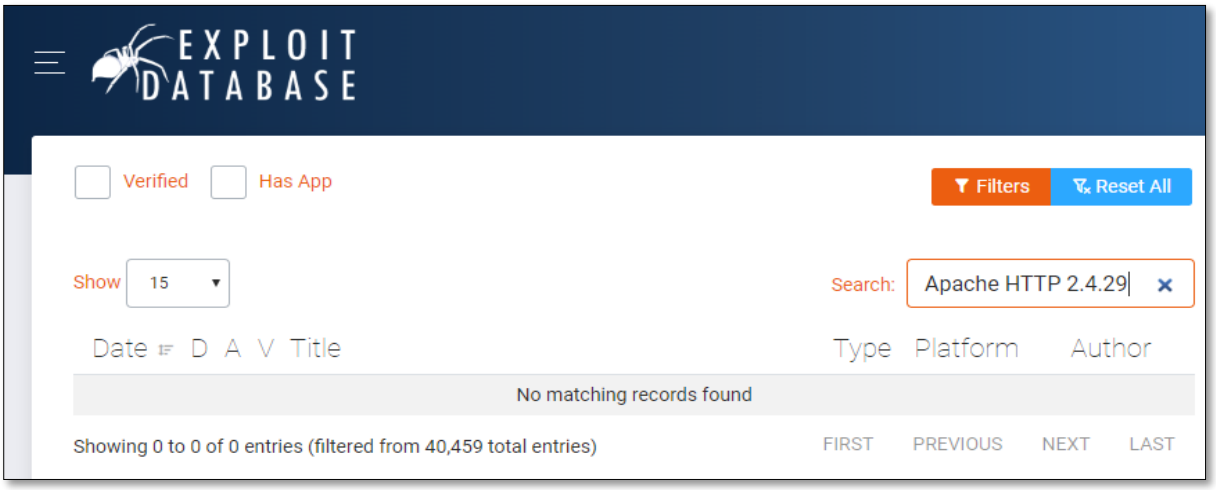
Having discovered the apache web server running on port 80 it was felt that this might potentially yield data points that could be used to access either the FTP or SSH services. We set out to explore the website in depth and our first scan involved the use of DirBuster (Directory Buster) to attempt to map out the directory structure of the website.



DirBuster attempting to map out the website Directory structure: ***dirbuster****.*

Having left this scan running while we manually investigated the site, it became clear to us that the directory structure had been obfuscated or masked in some manner as we could manually find folders through given clues while the program seemed to be struggling. It was decided to abandon this search in favour of manual searches.

Before our manual investigation of the webserver, a check was undertaken to see if any vulnerabilities could be found for the version of Apache the server is running. None were found.



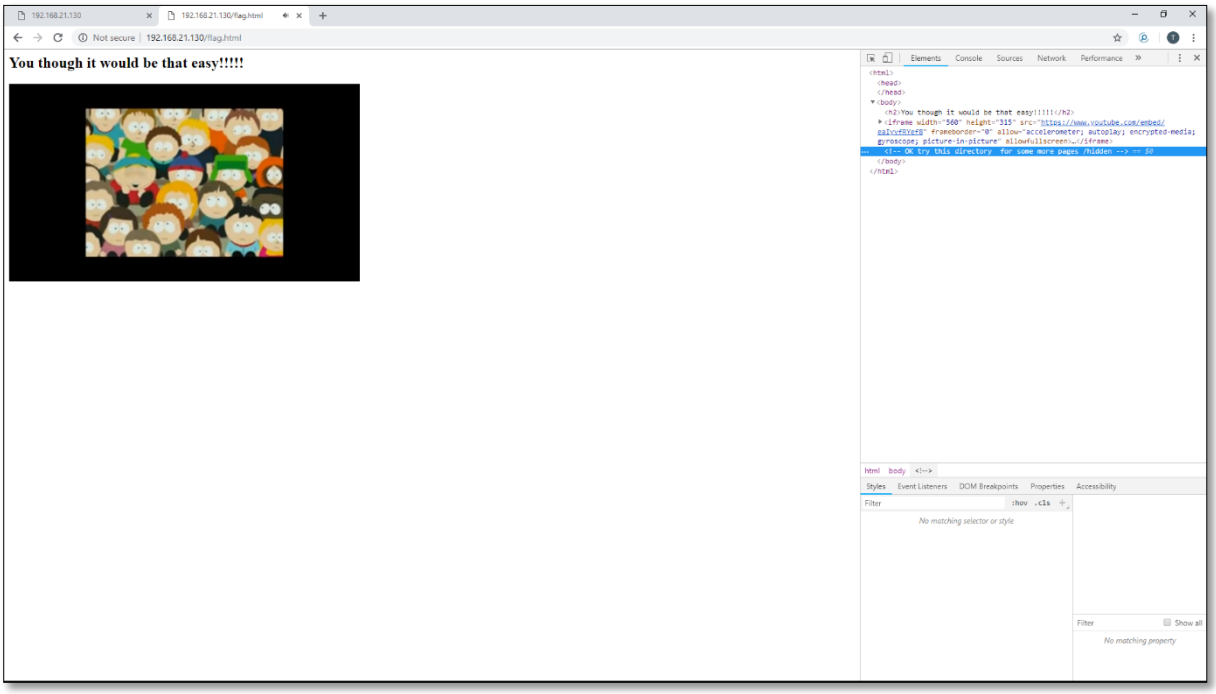
Vulnerability check performed on Apache version running on server.

And so, we began our manual investigation by visiting the servers address via our browser. This displayed an image. Having first attempted to select any hidden white text on the page, we began inspecting elements and discovered the revelation of another page in a html comment which led us to a page called 'flag.html'. No useful information was found in the Network or Application Sections.

Index page at 192.168.21.130 reveals flag.html

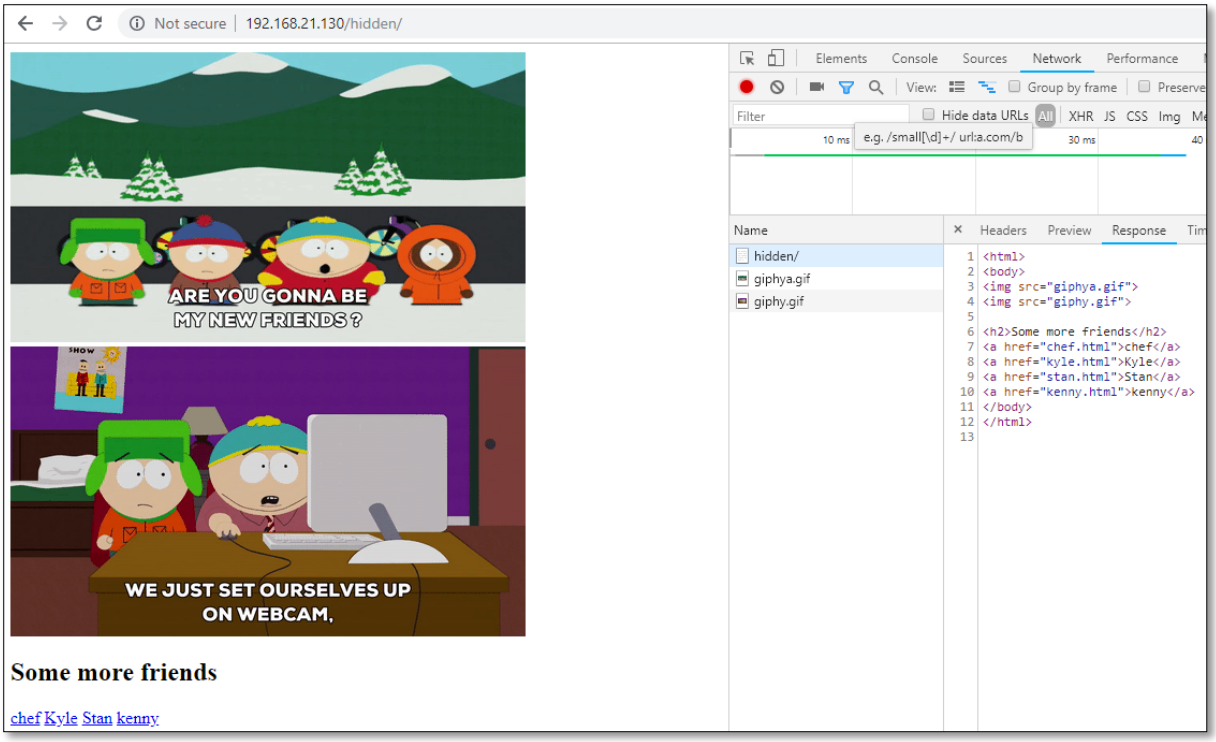
Next, we moved on to '192.168.21.130/flag.html'. This page contained a video starring Eric Cartman which hilariously mocked the user with the text 'You though it would be that easy!'. No hidden white text was found on the page. However, an examination of the source revealed a folder called '/hidden'.

The embedded video source code was examined to see if a flag might have been hidden therein, but none was found. No useful information found in the Network or Application sections.



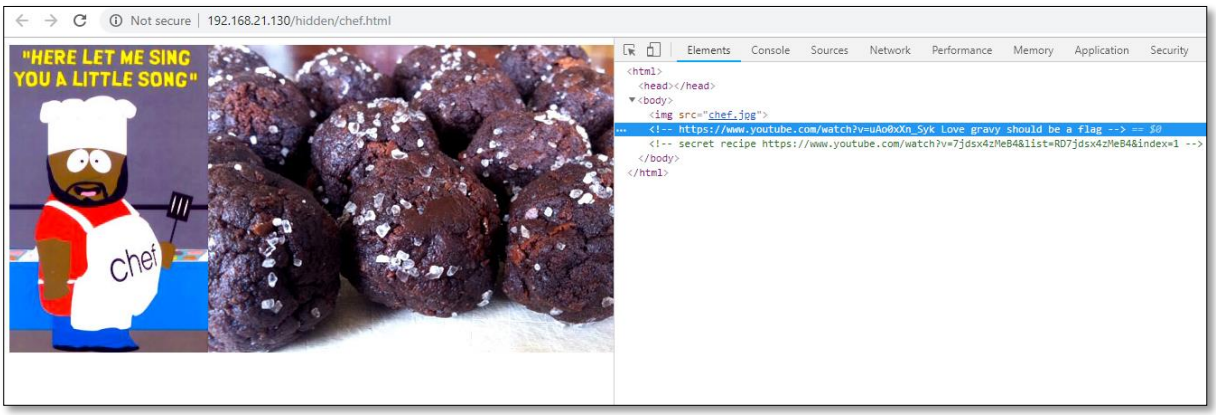
flag.html reveals the hidden folder.

The hidden folder at '192.168.21.130/hidden' was examined and displayed two GIFs', it also revealed four new pages: chef, Kyle, Stan and Kenny. No useful information was found in the Network or application sections of the source code.



/hidden reveals the four new pages: chef, Kyle, Stan & Kenny.

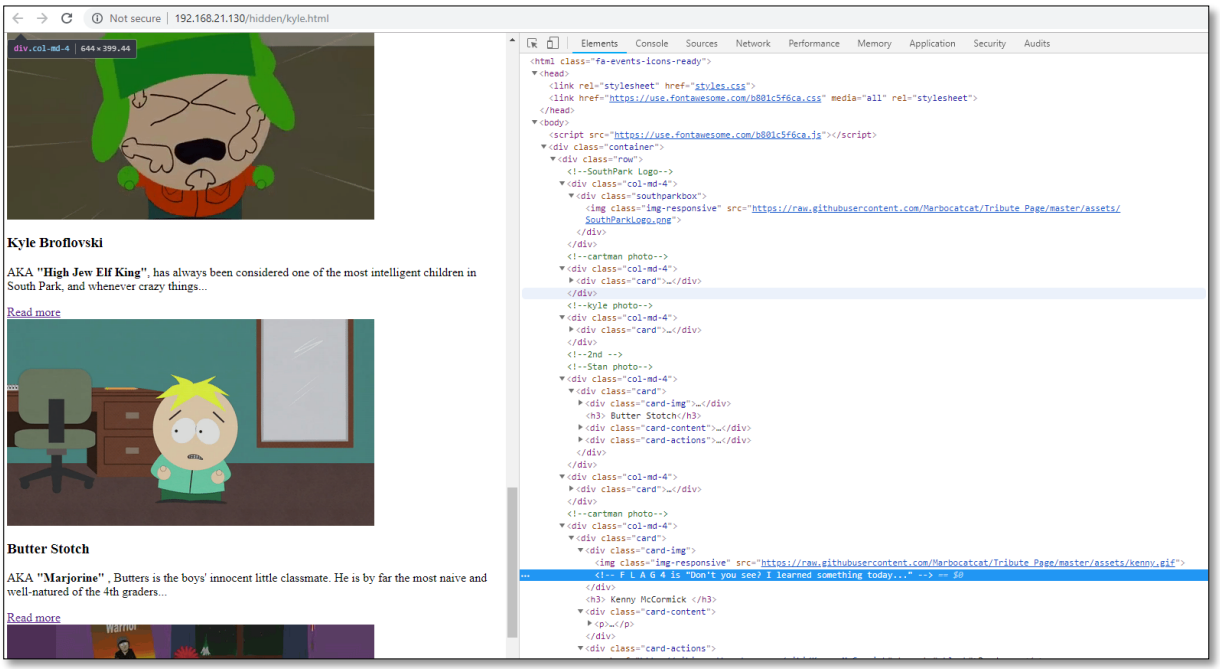
The chef page was found to contain two links (https://www.youtube.com/watch?v=7jdsx4zMeB4&list=RD7jdsx4zMeB4&index=1, https://www.youtube.com/watch?v=uAo0xXn_Syk) in the comment section with one link hinting at a possible flag via this comment: 'Love gravy should be a flag'. These pages where investigated and the entire comment sections where screened manually and although the key phrase 'Love Gravy' did appear 9 timeson the page, no flag was discovered. We may have to investigate these links further.



Hidden links revealed in html comments when source code of '192.168.21.130/chef.html' is viewed.

The Kyle page at ‘192.168.21.130/kyle.html’revealed some interest data points. Firstly, the page contained a series of further external links (http://wiki.southpark.cc.com/wiki/Eric_Cartman, http://wiki.southpark.cc.com/wiki/Kyle_Broflovski, http://wiki.southpark.cc.com/wiki/Butters_Stotch, http://wiki.southpark.cc.com/wiki/Stan_Marsh, http://wiki.southpark.cc.com/wiki/Kenny_McCormick). Each of those links leads to a wiki page related to the characters displayed on the page. These pages and the information contained on them may prove useful, alongside the keyword phrases ‘The Con, High Jew Elf King, Marjorine, Ranger Stan Marshwalker and Mysterion’ found on the parent page; in making a wordlist for brute force attacks further in the penetration test.

Most importantly, the Kyle page reveals **flag 4** in the source code. No further useful information was found in the Network or Application sections.



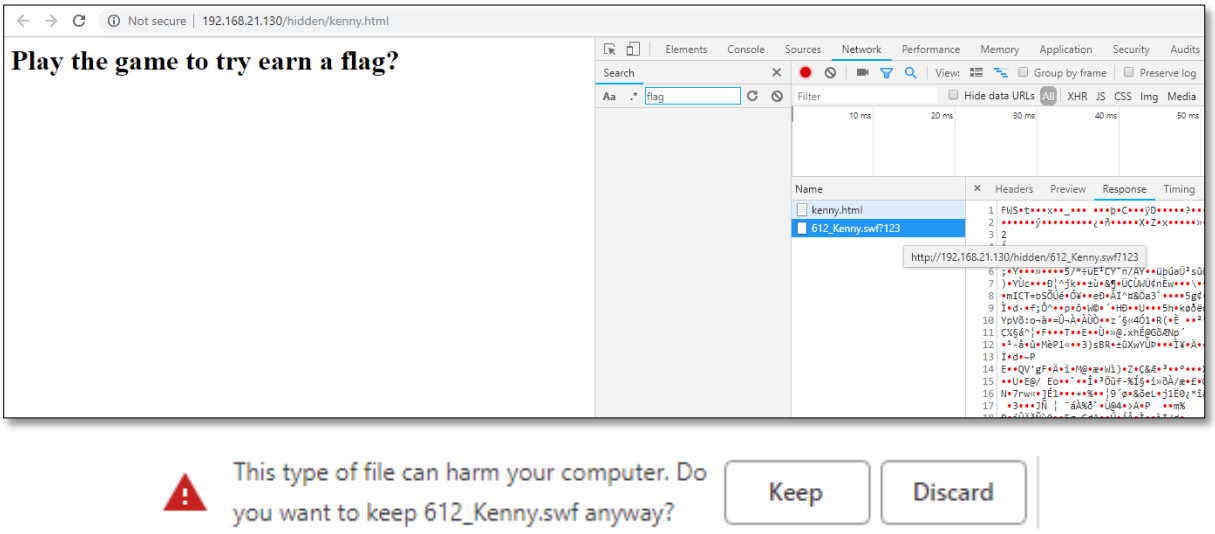
Flag 4= *“Don’t you see? I learned something today…”.*

Flag 4*found in the source code of 192.168.21.130/Kyle.html*

The Stan page at ‘192.168.21.130/Stan.html’is almost a direct copy of the Kyle.html page, however, it does not contain flag 4 in the source code. No other useful Information was found in the network or Application sections.

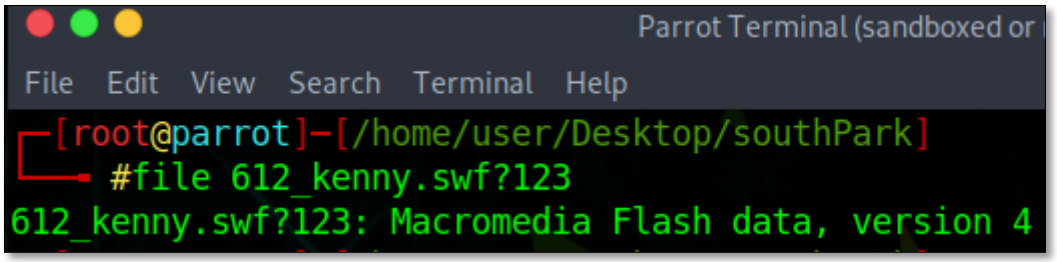
The Kenny page at ‘192.168.21.130/Kenny.html’ revealed some useful information. The page displayed a plain text message ‘Play the game to earn a flag’. This is a classic capture the flag trap and any relatively experienced player knows, you never play the game unless you absolutely have too. We suspected immediately the source code would hold some kind of important data. Our suspicions were rewarded almost immediately. Although the source code itself did not contain any flags, it also did not appear to contain any JavaScript or other flash-based code. Upon inspect the network data we discovered a curious response containing a file called ‘612_Kenny.swf?123’.

A quick recon search tells us that .swf files are flash files. However, it is odd that the file is being sent and not run when the page loads. Right clicking on this file and selecting open in new tab allowed us to download and inspect this file.



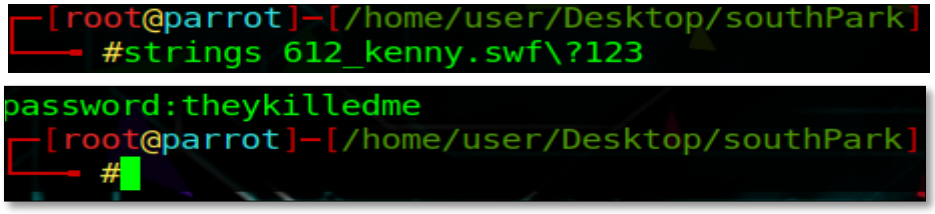
Kenny page reveals a flash file called '612_kenny.swf?123'

A cursory attempt to run this flash failed. It was suspected that the file extension might be obfuscated or change so we ran the 'file' command to examine the file in depth. This confirms the file is a flash file. Yet the file remains unplayable.



File run on '612_kenny.swf?123'.

A last-ditch attempt to obtain some data from this file was made by running 'strings' on the file to see if any useful information could be gained in plain text from the file. This proved to a useful Hail-Mary scan and a password '**theykilledme**' was recovered from the file.

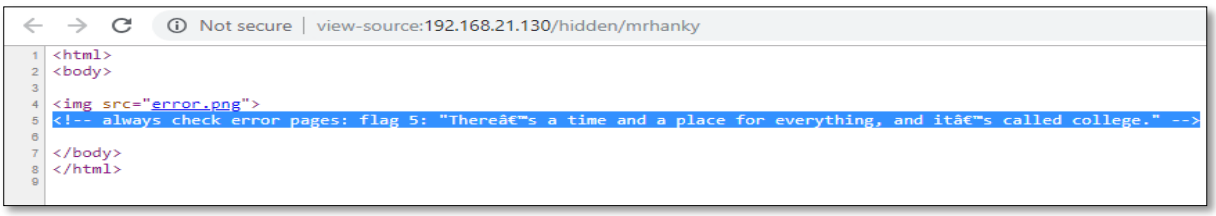


*Strings run on '612_kenny.swf?123' and password '**theykilledme**' recovered.*

While this appeared to be useful information we did not as yet have a place to use this information. However, we expected the password to be crucial in good time.

At this stage we had run out of pages to examine so the time seemed right to begin to experiment with random possible names using character names from the show. Our first attempt was '192.168.21.130/hidden/mrhanky.html'.

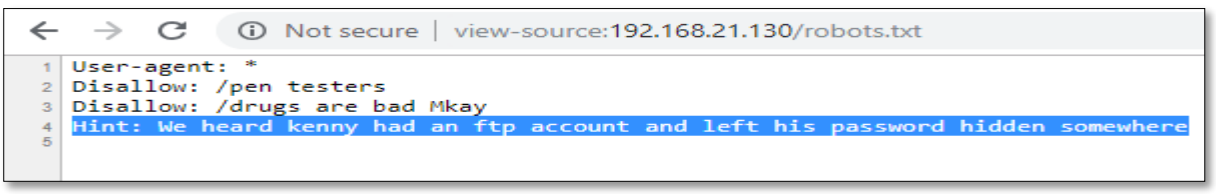
This returned an error page. This page was almost dismissed, however; an experienced member of the team pointed out that no stone should be left unturned and we examined the source code. Sure enough, f**lag 5**was discovered in the source code.



Flag 5= *"There's a time and a place for everything, and it's called college."*

Flag 5 *found in error response.*

Many more of these searches were undertaken, however it soon became apparent that with no more useful info and nothing but errors being returned, far too much time was being wasted. As a final attempt to discover hidden pages we attempted to find a robots.txt page, and one was indeed found. Furthermore, the robots text page revealed some interesting information.



Robots.txt page reveals the existence of a Kenny FTP account, along with a clue revealsthat further passwords may be found in images.

This information proved a lucky break. We had at this point been suspecting that we might need to create a user list from perhaps character names in order to attempt to enumerate user accounts in FTP login attempts. This negated the need for that for now and gave us a starting point.

8: FTP Revisited 1

Armed now with both an account, and a potential password; this allowed us to attempt to gain access to the FTP server once more. We began by using the account name provided by robots.txt and knowing that the password 'theykilledme' likely referred to kenny, we attempted to use this. We expected this attempt to succeed and sure enough it did.

```
┌─[root@parrot]─[/home/user/Desktop/southPark]
└──╼ #ftp 192.168.21.130
Connected to 192.168.21.130.
220 Well done You've found FLag 1: "No, he's talking about fuck. You can't say f
uck in school, you fuckin' fat ass!"
Name (192.168.21.130:user): kenny
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

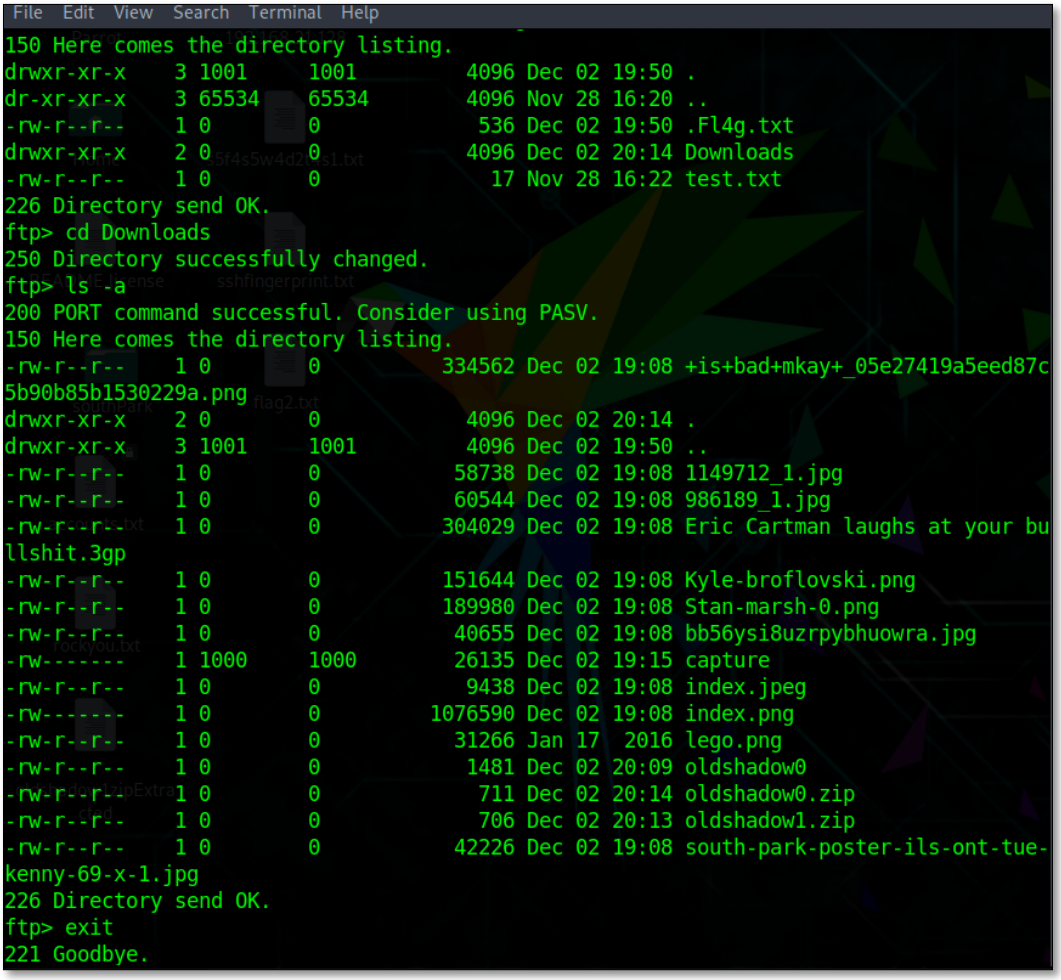
Successful log in to FTP server using account name 'kenny' and password 'theykilledme'.

Our first instinct upon gaining our first access to the sever was to go right after the passwd and shadow files. However, this proved impossible. It appeared that the server file structure and/or permissions had been set up in such a way that we had limited permissions and access.

```
File  Edit  View  Search  Terminal  Help
ftp> cd /etc
550 Failed to change directory.
ftp> put test.txt
local: test.txt remote: test.txt
200 PORT command successful. Consider using PASV.
553 Could not create file.
ftp> ls -a
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
dr-xr-xr-x    3 65534    65534        4096 Nov 28 16:20 .
dr-xr-xr-x    3 65534    65534        4096 Nov 28 16:20 ..
drwxr-xr-x    3 1001     1001         4096 Dec 02 19:50 files
226 Directory send OK.
ftp> cd files
250 Directory successfully changed.
ftp> ls -a
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
drwxr-xr-x    3 1001     1001         4096 Dec 02 19:50 .
dr-xr-xr-x    3 65534    65534        4096 Nov 28 16:20 ..
-rw-r--r--    1 0        0             536 Dec 02 19:50 .Fl4g.txt
drwxr-xr-x    2 0        0            4096 Dec 02 20:14 Downloads
-rw-r--r--    1 0        0              17 Nov 28 16:22 test.txt
226 Directory send OK.
ftp> cd Downloads
250 Directory successfully changed.
```

Here we attempt to access the /etc folder, followed by an attempt to upload a file.

The rejection of the attempt to access the /etc folder coupled with the refusal to put a file, told us that we had severely restricted permissions. This being the case privilege escalation was impossible.



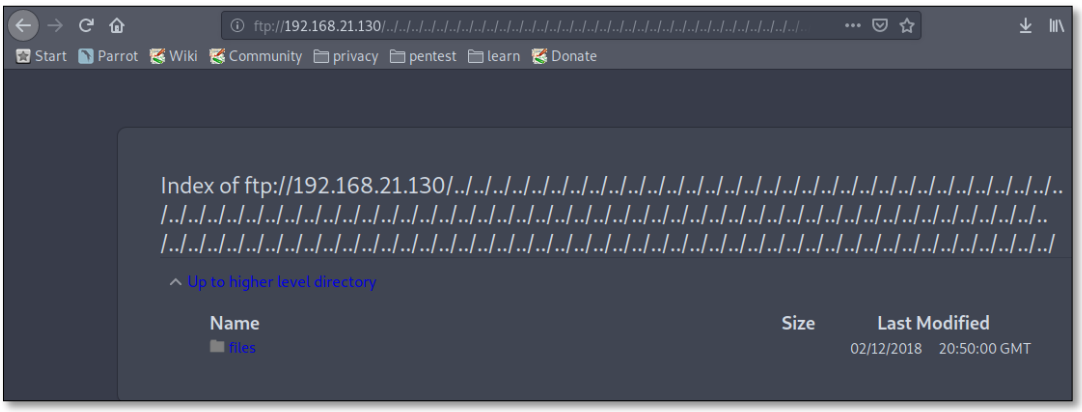
Here we view the file listings of the folders /files and /Downloads. These are the only folders we can currently access.

Above we see the directory listings of the folders we could accessat this stage. The next step was to download these files and begin to investigate them individually. However due to our restricted permissions, there are two files ‘capture’ and ‘index.png’ that we could not get at this stage. We believed an escalated shell access will be needed to get these files.

```
┌─[root@parrot]─[/home/user/Desktop/southPark]
└──#wget -r ftp://kenny:theykilledme@192.168.21.130/files
--2018-12-08 20:57:36--  ftp://kenny:*password*@192.168.21.130/files
           => '192.168.21.130/.listing'
Connecting to 192.168.21.130:21... connected.
Logging in as kenny ... Logged in!
```

Here we download the entire /files folder containing all the files.

As a final step in our examination via FTP, before beginning to examine the server via a browser to see if any other folders could be found.

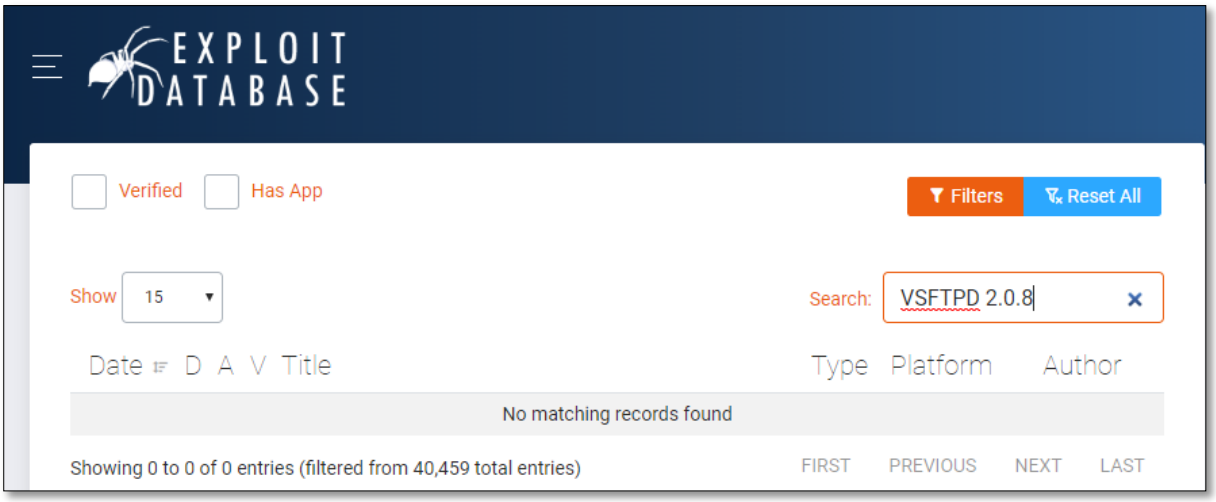


Here we are, tumbling down the rabbit hole. This confirms the file structure has been obfuscated.

Above you can see that the file structure has indeed been obfuscated. One of our group members spent almost hour confirming this. We further confirmed that the folder structure we have previously examined is all we could access. Interesting to note that we cannot view potential hidden files and folders via the browser view by default.

9: FTP File Transfer Folder Investigated & The Finding of Flag 6

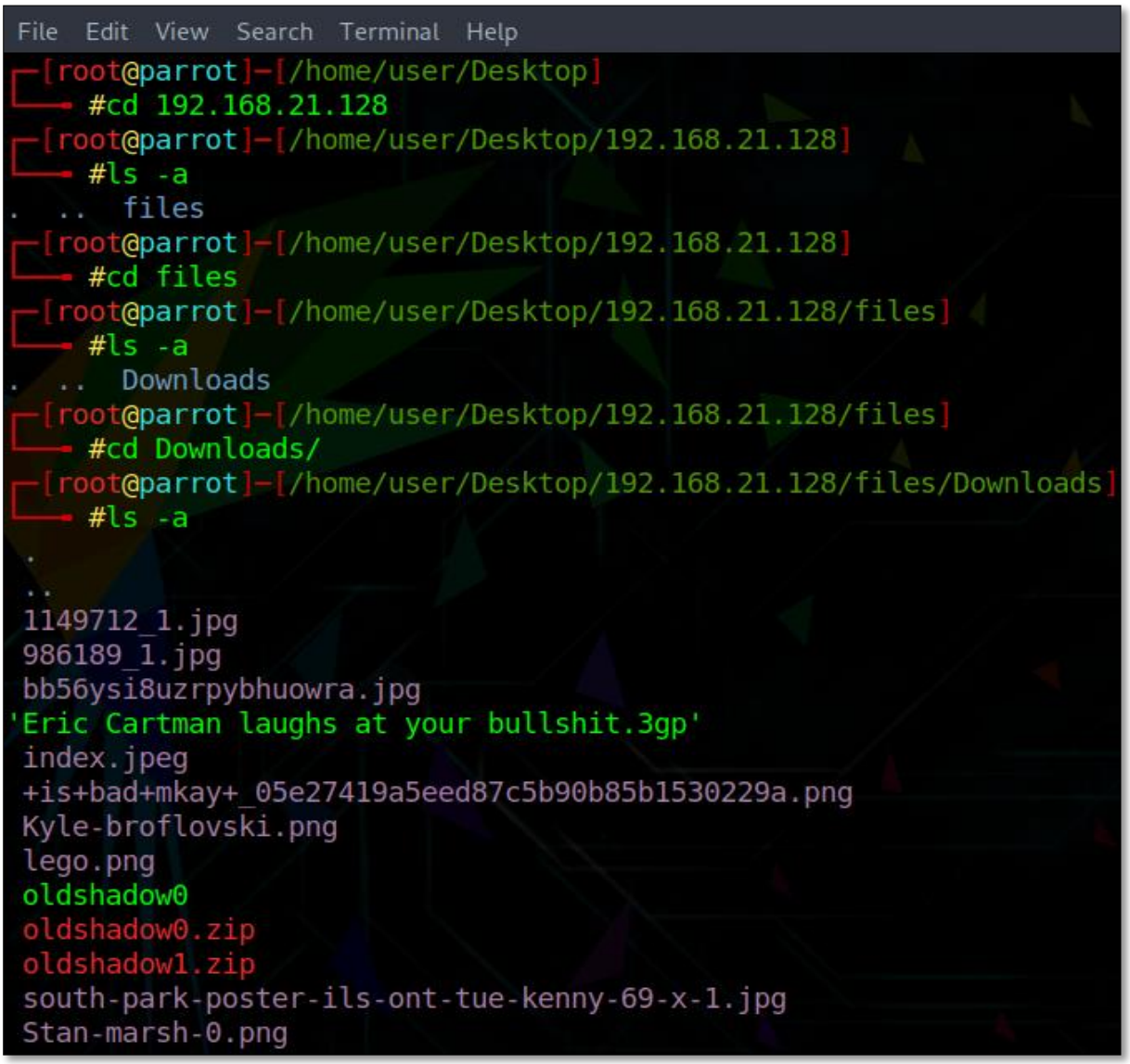
A vulnerability scan of the VSFTPD server running on the server was performed. No exploits were found.



Vulnerability check performed on VSFTPD version running on server.

9.1: FTP File Transfer Folder

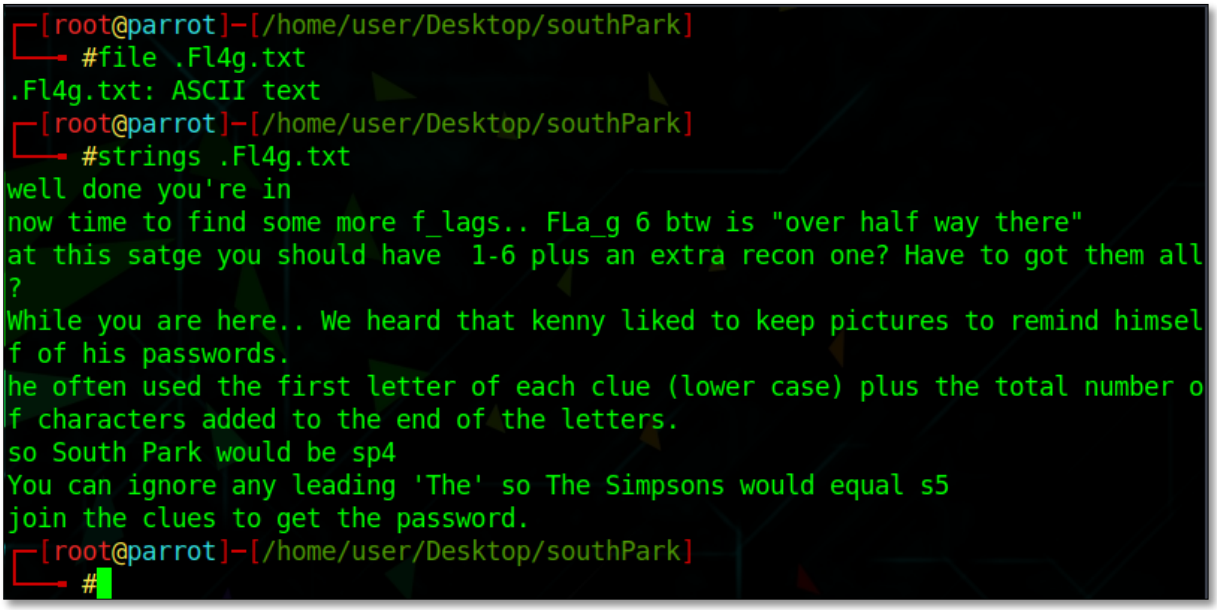
Our File investigation began with a cursory inspection of all available files to be examined, both visible and hidden.



Files transferred via FTP viewed. Two files (/files/.Fl4g.txt&./files/test.txt) appear to be missing. These where grabbed individually.

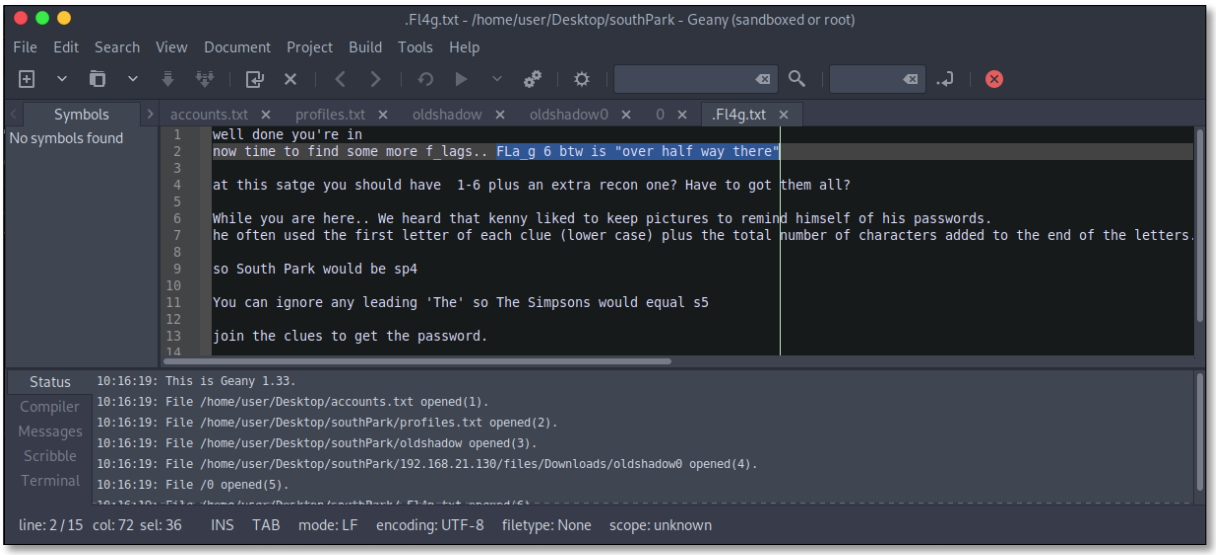
This inspection of the files retrieved, when compared to our earlier viewing of the file structure on the server, informed us that two files we expected to find have failed to transfer. These files (/files/.Fl4g.txt& .files/test.txt) were downloaded individually to complete the file/folder structure as it appeared on the server, minus the two restricted files we cannot retrieve (capture & index.png).

These files, being contained within the file folder alongside the sub folder Downloads became the first files to be investigated. So, we attempted to run file and strings on this files before attempting to view them manually.



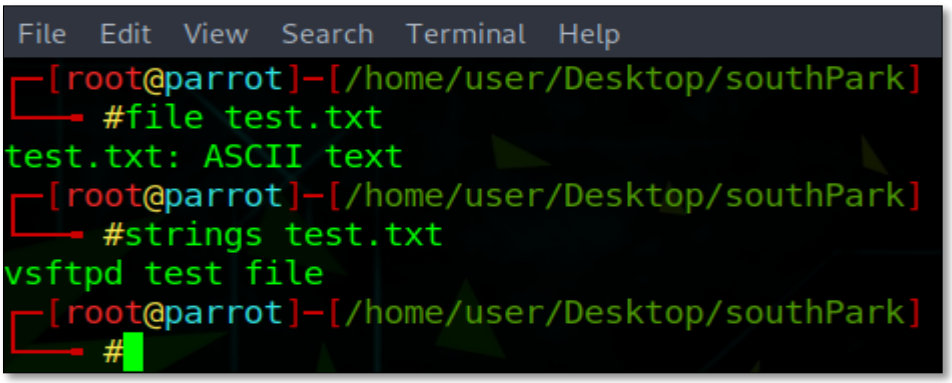
Flag 6 *=revealed to be "over half way there" in .Fl4g.txt.*

As we see above, file reveals the file is a txt file and Strings reveals contents. There also appeared to be a cryptic clue as to how to retrieve a password from a picture kenny may have hidden. This clue stated that "he often used the first letter of each clue (lower case) plus the total of characters added to the end of the letters. So South Park would be sp4 You can ignore and leading 'The' so The Simpsons would equal s5 join the clues to get the password.' No picture has yet been discovered which might related to the solution.



Flag 6 *confirmed manually to be "over half way there" in .Fl4g.txt.*

Having view ./Fl4g.txt manually we confirm **Flag 6** to be "over half way there". The process was then applied to the file 'test.txt'. However, test.txt did not reveal any new useful data points.



Test.txt examined with no relevant data found.

Having examined all files in the 'files' folder, we extended our examination to the 'Downloads' sub folder. The same examination process was carried out with all files contained in /Downloads sub folder, however; as there are quite a number of image files we will elected not to display screenshots of any irrelevant files found in this folder having already demonstrated the process used. We did of course, continue to document relevant files in this folder. It is worth mentioning that should it be deemed necessary we may return to the image files contained in this folder to strip out any plain text words displayed to add to any wordlists created for brute force attacks.

The irrelevant files examined via file and string included '1149712_1.jpg', '986189_1.jpg', 'bb56ysi8uzrpybhuowra.jpg', 'Eric Cartman laughs at your bullshit.3gp', 'index.jpeg', '+is+bad+mkay+_05e27419a5eed87c5b90b85b1530229a.png', 'Kyle-broflovski.png', and 'Stan-marsh-0.png'.

Four files were found to be of interest. They were as follows: 'oldshaow0', 'oldshadow0.zip', 'oldshadow1.zip' and lego.png. We began investigating these files with the shadow files. It should be noted that great difficulty was encountered in cracking these files. A brute force attack was initiated on the unzipped oldshadow0 file however, despite numerous scans running across 4 VM's using every hash format we could not successfully crack this shadow file.

Brute force attacks where immediately undertaken to crack both 'oldshadow0.zip' and 'oldshadow1.zip', however, neither of these where successful despite being run several times on two different VM's.

After much head scratching, we switched from using 'John The Ripper' to HashCat. Again, to no avail. Despite almost 150 hours of combined attacks across multiple VM's and a google VM our attacks repeatedly failed, and we simply did not understand why. Finally, if came down to a deep mistrust of everything. So, we endeavoured to use the process of elimination.

We began by moving outside our VM's and installed both John and HashCat on a windows system.

Having run both using the same commands, and files and failing again, we replace our uncracked password file. Interestingly we discovered a major oversight at this juncture. Upon closely examining the data in our shadow files, both originally unzipped (oldshadow0) and zipped (oldshadow) we discovered that we had missed a potential password. One hash differed between both files. This extra hash was added to our hash file to be cracked.

It must be pointed out here while John will accept the full string retrieved from the shadow file, HashCat will not. The hash must be stripped out to be cracked in HashCat.

```
cartman:$1$ebjntvDS$5X9oFpg/sMOJGe8QY8jpv/:17863:0:99999:7:::
kenny:$6$g71s2gMV$/M2kfgiOd4/KmHBeyLQ4wDBC1BxfFkjx161N8n8I7RVEqezzRHt5pY12qKWN7S4gQyub.nRID2UvhdK.djxjO0:17864:0:99999:7:::
chef:$6$SuNBnaqZ$p3T8Rqxa5VcSDyUHhLoqgoaiqXiiZZoTZW7ItCj36.FnDrCS.5nASAWVdNypgD4fIqxFbEzZXtj.AIiaCly6.0:17868:0:99999:7:::
chef:$6$wVEx/4tR$OaTlSRIKC0i9aAqIRbTvN9ZF0tqFPJJiPZyoXU25zSGJyEFlXxpz.FJoWdCx2C36CM9ogF5vJJAOU.a9qRVNx/:17868:0:99999:7:::
```

Hashes as they are entered into 'John The ripper'.

```
$1$ebjntvDS$5X9oFpg/sMOJGe8QY8jpv/
$6$g71s2gMV$/M2kfgiOd4/KmHBeyLQ4wDBC1BxfFkjx161N8n8I7RVEqezzRHt5pY12qKWN7S4gQyub.nRID2UvhdK.djxjO0
$6$SuNBnaqZ$p3T8Rqxa5VcSDyUHhLoqgoaiqXiiZZoTZW7ItCj36.FnDrCS.5nASAWVdNypgD4fIqxFbEzZXtj.AIiaCly6.0

$6$wVEx/4tR$OaTlSRIKC0i9aAqIRbTvN9ZF0tqFPJJiPZyoXU25zSGJyEFlXxpz.FJoWdCx2C36CM9ogF5vJJAOU.a9qRVNx/
```

Hashes as they are entered into 'HashCat'.

The process of investigating how to correctly investigate the data also showed us how to investigate the hash types. For example, in the image above you'll notice that the first password begins with 1 and the rest begin with 6. 1 denotes md5crypt hash type and 6 denotes sha512crypt hash type. This was very useful information.

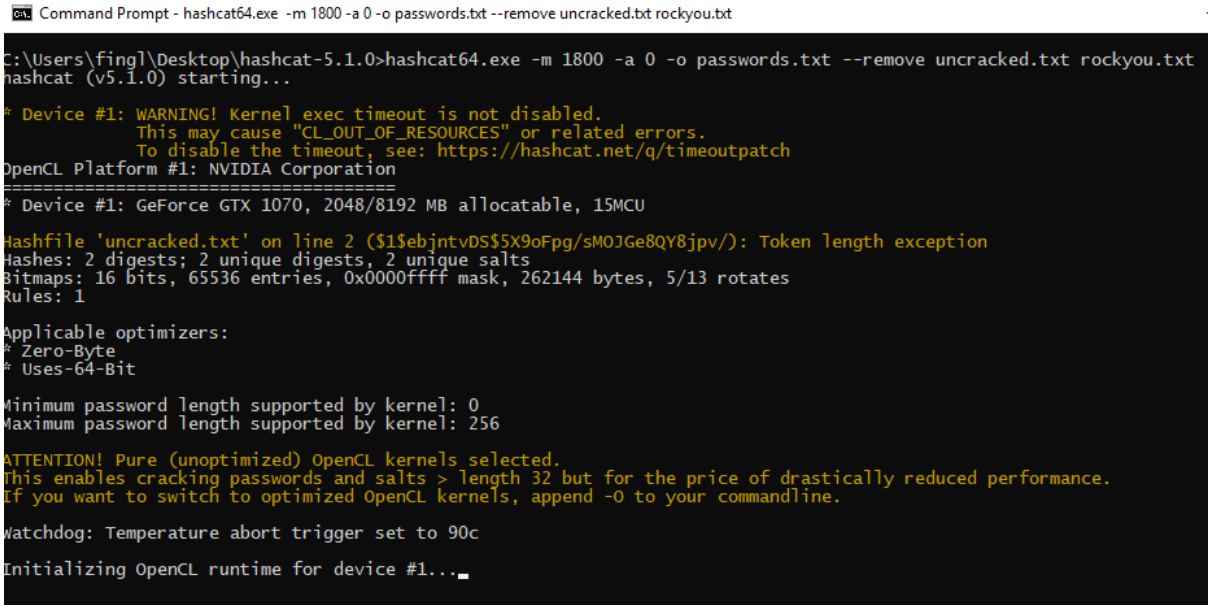
Now having ruled out our Machines, our operating systems and our password files, this left only our word list, and having downloaded a fresh rockyou.txt file from GitHub, we were surprised to find they did not match.

Armed with our updated wordlist, we set out to attempt to crack these passwords again, and sure enough it took only a few mins to crack both.

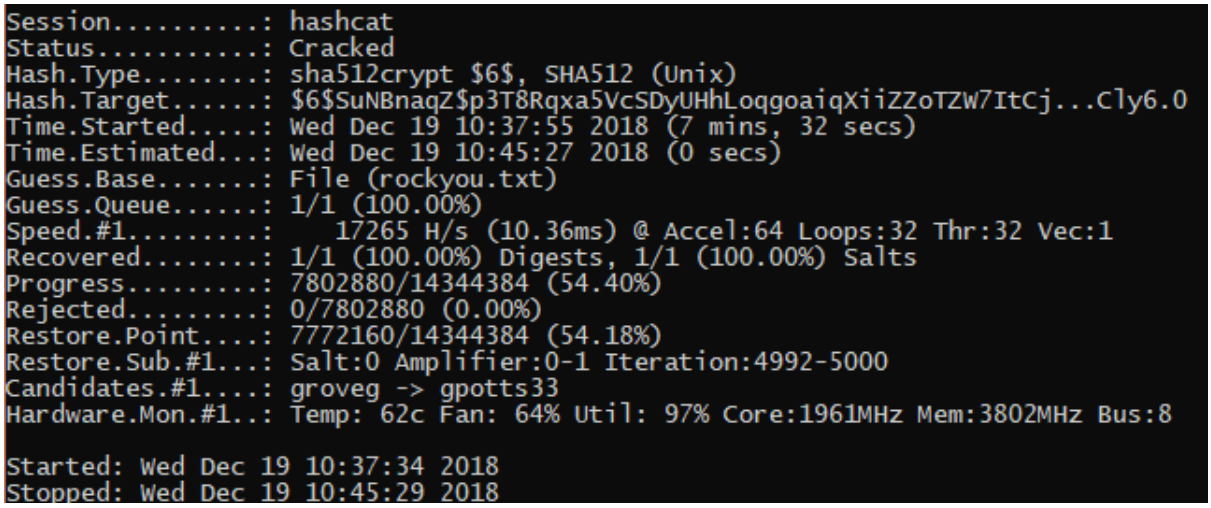
Using this command: hashcat64.exe -m 1800 -a 0 -o passwords .txt –remove uncracked.txt rockyou.txt

- Were passwords.txt = our output file
- Uncracked.txt = our stored hashes
- Rockyou.txt = our chosen updated wordlist
- 1800 = sha512crypt hash type

One Final word of note, with this syntax, ‘HashCat’ stopped after its first successful crack, so we entered only one hash per pass into our file and re ran it multiple times.



Here we see our first ‘HashCat’ attack beginning with our updated wordlist.



Here we see out first successful password crack result.

```
Session..........: hashcat
Status...........: Cracked
Hash.Type........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: $6$wVEx/4tR$OaTlSRIKC0i9aAqIRbTvN9ZF0tqFPJJiPZyoXU2...qRVNx/
Time.Started.....: Wed Dec 19 10:48:20 2018 (7 mins, 29 secs)
Time.Estimated...: Wed Dec 19 10:55:49 2018 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    17374 H/s (10.79ms) @ Accel:64 Loops:32 Thr:32 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 7802880/14344384 (54.40%)
Rejected.........: 0/7802880 (0.00%)
Restore.Point....: 7772160/14344384 (54.18%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:4992-5000
Candidates.#1....: groveg -> gpotts33
Hardware.Mon.#1..: Temp: 63c Fan: 66% Util: 95% Core:1961MHz Mem:3802MHz Bus:8

Started: Wed Dec 19 10:48:03 2018
Stopped: Wed Dec 19 10:55:51 2018

C:\Users\fingl\Desktop\hashcat-5.1.0>
```

Here we see our second successful password crack result

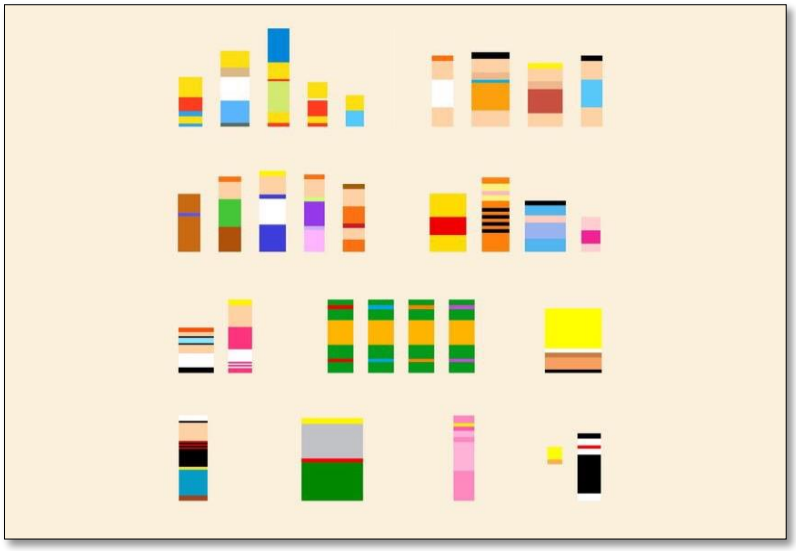
```
$6$SuNBnaqZ$p3T8Rqxa5VcSDyUHhLoqgoaiqXiiZZoTZW7ItCj36.FnDrCS.5nASAWVdNypgD4fIqxFbEzZXtj.AIiaCly6.0:gravy123
$6$wVEx/4tR$OaTlSRIKC0i9aAqIRbTvN9ZF0tqFPJJiPZyoXU25zSGJyEF1Xxpz.FJoWdCx2C36CM9ogF5vJJAOU.a9qRVNx/:gravy121
```

Here we see our successfully cracked passwords in both hash and plain text form. The ':' acts as a divider.

```
┌─[✗]─[root@parrot]─[/home/user/Desktop/192.168.21.128/files/Downloads]
└──╼ #fcrackzip -b -c aA1! -u oldshadow0.zip
┌─[root@parrot]─[/home/user/Desktop/192.168.21.128/files/Downloads]
└──╼ #fcrackzip -b -c Aa1! -u oldshadow1.zip
```

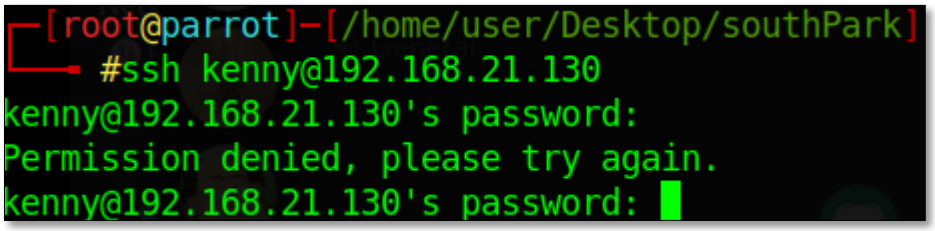
FCrackzip brute force attempts on 'oldshadow0.zip' & 'oldshadow1.zip'.

The final file of interest was the 'lego.png file'. It is important to point out that this file was initially overlooked and considered to be irrelevant. However, after some thought on the images contained in the folder we realized the lego.png image contained crude representations of cartoon characters we recognized. Putting this together with our earlier clue received from flag 6, we began to attempt to piece together the password using the steps recommended in the clue.



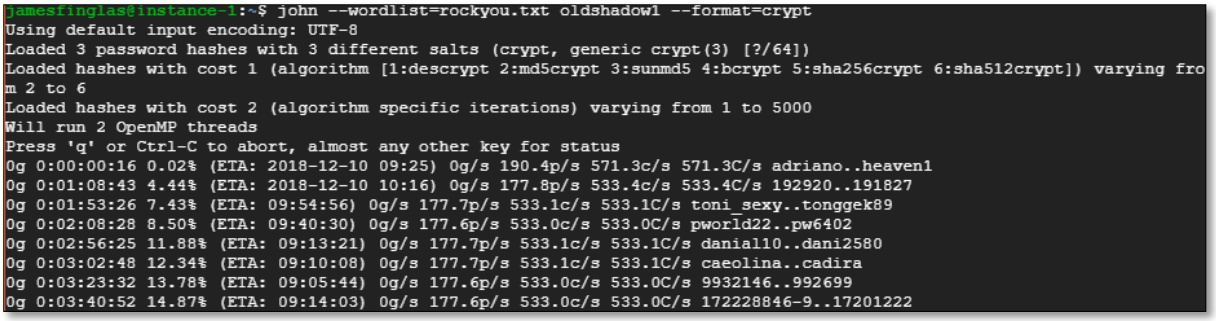
Lego.png contained crude pixel art cartoon character representations.

The process of gaining this password was a lengthy process of trial and error. The was a very clever way obfuscate access to a password because no tools could be used to data mine the file. It being purely visual based we had to manually make up passwords. The process took a very long time and approximately 150 to 200 passwords were attempted before we struck pay dirt with 's5f4sd5wtp4dl2tmnt4ss1p1b1pp1tas2'. We initially assumed that this would be an SSH password and so we attempted to log into SSH as kenny using this password.



SSH log in failure with password retrieved from lego.png and flag 6 clue.

As we see above, this log in attempt failed. It took some time, as it had been a long time since our team discovered the .zip files, for one of our team to suggest going back to the .zip files to attempt to unzip these files. We first attempted to unzip 'oldshadow0.zip', but this proved unsuccessful. However, an attempt to unzip 'oldshadow1.zip' proved successful and this gave us a second shadow file which we renamed 'oldshadow1' to attempt to brute force attack. This attack was undertaken immediately on a team members Google VM server.

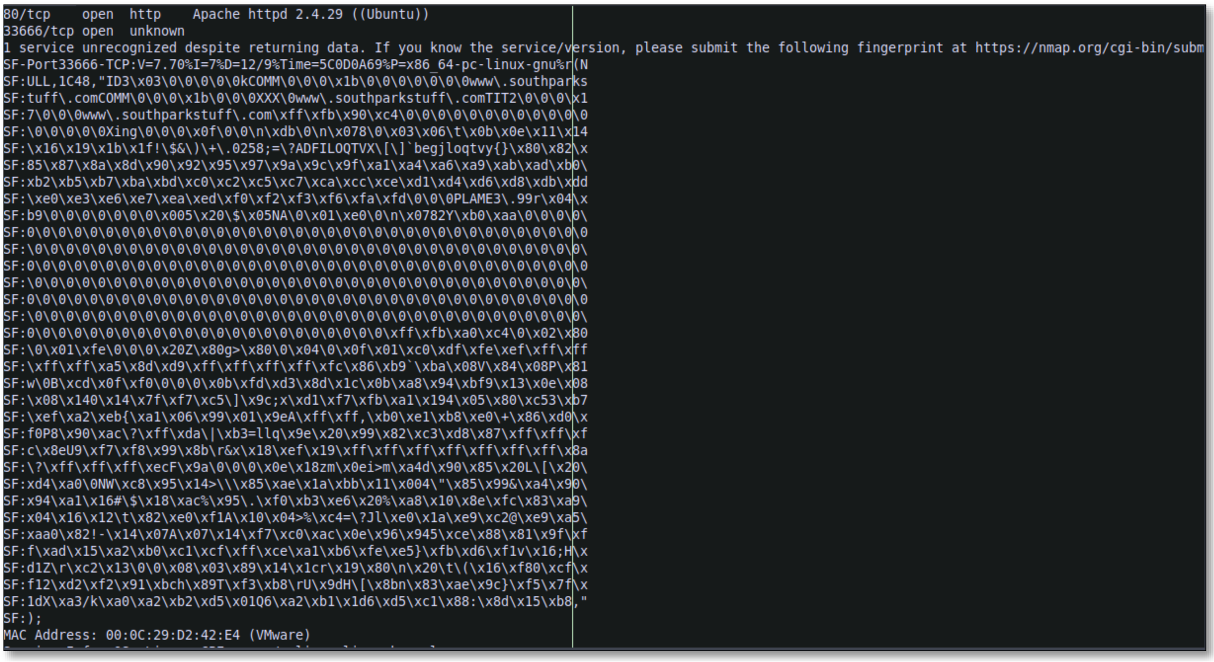


Brute force attempts to crack the oldshadow1 file via google VM.

10: Unknown Service (33666) & Finding Flag 2

Having exhausted almost all avenues of data extraction and awaiting the results of brute force attacks the team revisited our NMap scans. We are left with SHH, an unknown service and three UDP services, one of which is a printer service; to investigate. We decided next to investigate the unknown service to rule it out.

We expected this service to be a broken service of little relevance. However, upon closer inspection we again noticed the output of what appeared to be a website in the NMap scan. We re-ran the NMap scan and sent the data to a text file for easier viewing.

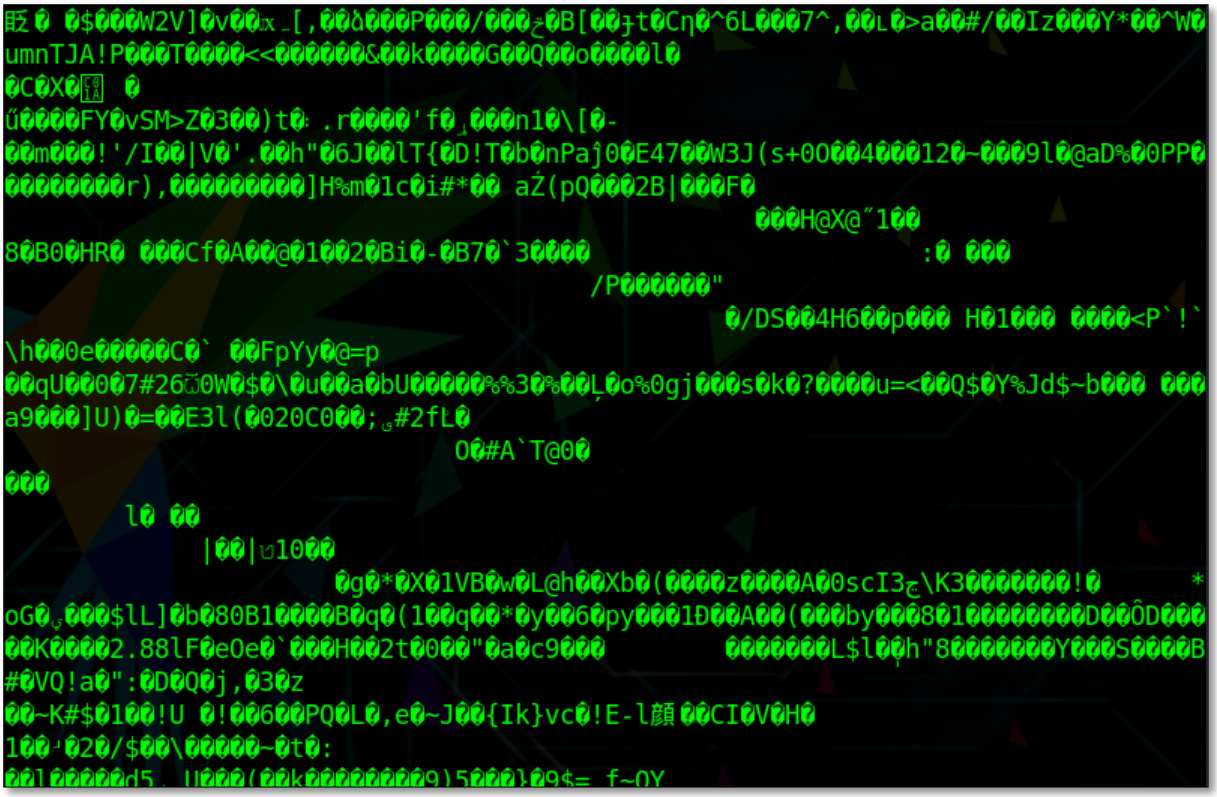


```
80/tcp    open  http    Apache httpd 2.4.29 ((Ubuntu))
33666/tcp open  unknown
1 service unrecognized despite returning data. If you know the service/version, please submit the following fingerprint at https://nmap.org/cgi-bin/subm
SF-Port33666-TCP:V=7.70%I=7%D=12/9%Time=5C0D0A69%P=x86_64-pc-linux-gnu%r(N
SF:ULL,1C48,"ID3\x03\0\0\0\0\0kCOMM\0\0\0\x1b\0\0\0\0\0\0\0www\.southparks
SF:tuff\.comCOMM\0\0\0\x1b\0\0\0XXX\0www\.southparkstuff\.comTIT2\0\0\0\x1
SF:7\0\0\0www\.southparkstuff\.com\xff\xfb\x90\xc4\0\0\0\0\0\0\0\0\0\0\0\0
SF:\0\0\0\0\0Xing\0\0\0\x0f\0\0\n\xdb\0\n\x078\0\x03\x06\t\x0b\x0e\x11\x14
SF:\x16\x19\x1b\x1f!\$&\)\+\.0258;=\?ADFILOQTVX\[\]`begjloqtvy{}\x80\x82\x
SF:85\x87\x8a\x8d\x90\x92\x95\x97\x9a\x9c\x9f\xa1\xa4\xa6\xa9\xab\xad\xb0\
SF:xb2\xb5\xb7\xba\xbd\xc0\xc2\xc5\xc7\xca\xcc\xce\xd1\xd4\xd6\xd8\xdb\xdd
SF:\xe0\xe3\xe6\xe7\xea\xed\xf0\xf2\xf3\xf6\xfa\xfd\0\0\0PLAME3\.99r\x04\x
SF:b9\0\0\0\0\0\0\0\x005\x20\$\x05NA\0\x01\xe0\0\n\x0782Y\xb0\xaa\0\0\0\0\
SF:0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
SF:\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
SF:0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
SF:\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
SF:0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
SF:\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\
SF:0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\xff\xfb\xa0\xc4\0\x02\x80
SF:\0\x01\xfe\0\0\0\x20Z\x80g>\x80\0\x04\0\x0f\x01\xc0\xdf\xfe\xef\xff\xff
SF:\xff\xff\xa5\x8d\xd9\xff\xff\xff\xff\xfc\x86\xb9`\xba\x08V\x84\x08P\x81
SF:w\0B\xcd\x0f\xf0\0\0\0\x0b\xfd\xd3\x8d\x1c\x0b\xa8\x94\xbf9\x13\x0e\x08
SF:\x08\x140\x14\x7f\xf7\xc5\]\x9c;x\xd1\xf7\xfb\xa1\x194\x05\x80\xc53\xb7
SF:\xef\xa2\xeb{\xa1\x06\x99\x01\x9eA\xff\xff,\xb0\xe1\xb8\xe0\+\x86\xd0\x
SF:f0P8\x90\xac\?\xff\xda\|\xb3=llq\x9e\x20\x99\x82\xc3\xd8\x87\xff\xff\xf
SF:c\x8eU9\xf7\xf8\x99\x8b\r&x\x18\xef\x19\xff\xff\xff\xff\xff\xff\xff\x8a
SF:\?\xff\xff\xff\xecF\x9a\0\0\0\x0e\x18zm\x0ei>m\xa4d\x90\x85\x20L\[\x20\
SF:xd4\xa0\0NW\xc8\x95\x14>\\\x85\xae\x1a\xbb\x11\x004\"\x85\x99&\xa4\x90\
SF:x94\xa1\x16#\$\x18\xac%\x95\.\xf0\xb3\xe6\x20%\xa8\x10\x8e\xfc\x83\xa9\
SF:x04\x16\x12\t\x82\xe0\xf1A\x10\x04>%\xc4=\?Jl\xe0\x1a\xe9\xc2@\xe9\xa5\
SF:xaa0\x82!-\x14\x07A\x07\x14\xf7\xc0\xac\x0e\x96\x945\xce\x88\x81\x9f\xf
SF:f\xad\x15\xa2\xb0\xc1\xcf\xff\xce\xa1\xb6\xfe\xe5}\xfb\xd6\xf1v\x16;H\x
SF:d1Z\r\xc2\x13\0\0\x08\x03\x89\x14\x1cr\x19\x80\n\x20\t\(\x16\xf80\xcf\x
SF:f12\xd2\xf2\x91\xbch\x89T\xf3\xb8\rU\x9dH\[\x8bn\x83\xae\x9c}\xf5\x7f\x
SF:1dX\xa3/k\xa0\xa2\xb2\xd5\x01Q6\xa2\xb1\x1d6\xd5\xc1\x88:\x8d\x15\xb8,"
SF:);
MAC Address: 00:0C:29:D2:42:E4 (VMware)
```

Data stream emitted by port 33666 viewed in test form.

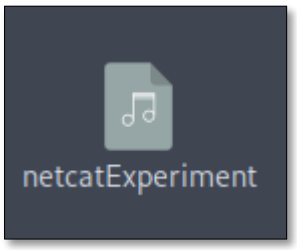
Upon a close inspection, while we did not know exactly what we were looking at, it did seem reminiscent of a data steam. We could not understand why a service would return a stream of data during an NMap scan. However, with no real idea what this data was, we simply decide to attempt to NetCat into all known TCP ports. While a basic NetCat connection can be established via ports 80, 21 and 22, we could achieve nothing through these ports without privilege escalation. Our attempts to send files failed.

Lastly, we attempted to NetCat into port 33666. This returned a data stream of unknown data.



Data stream of unknown data returned from our NetCat attempt to connected to port 33666(TCP).

This was very intriguing. We reconnected and sent this data to a file labelled ‘netcatExperiment’ via the command ‘nc 192.168.21.130 33666 > netcatexperiemnt’. Immediately the file appeared to be an audio file.



netcatExperiment audio file captured from port 33666(TCP).

When played, this file was confirmed as audio file containing the voice of Eric Cartman singing a well-known song from South Park titled ‘Kyles mom’s a bitch!”. However, it also revealed **Flag 2** via audio format to be “Kyle’s mom is a lovely person”.

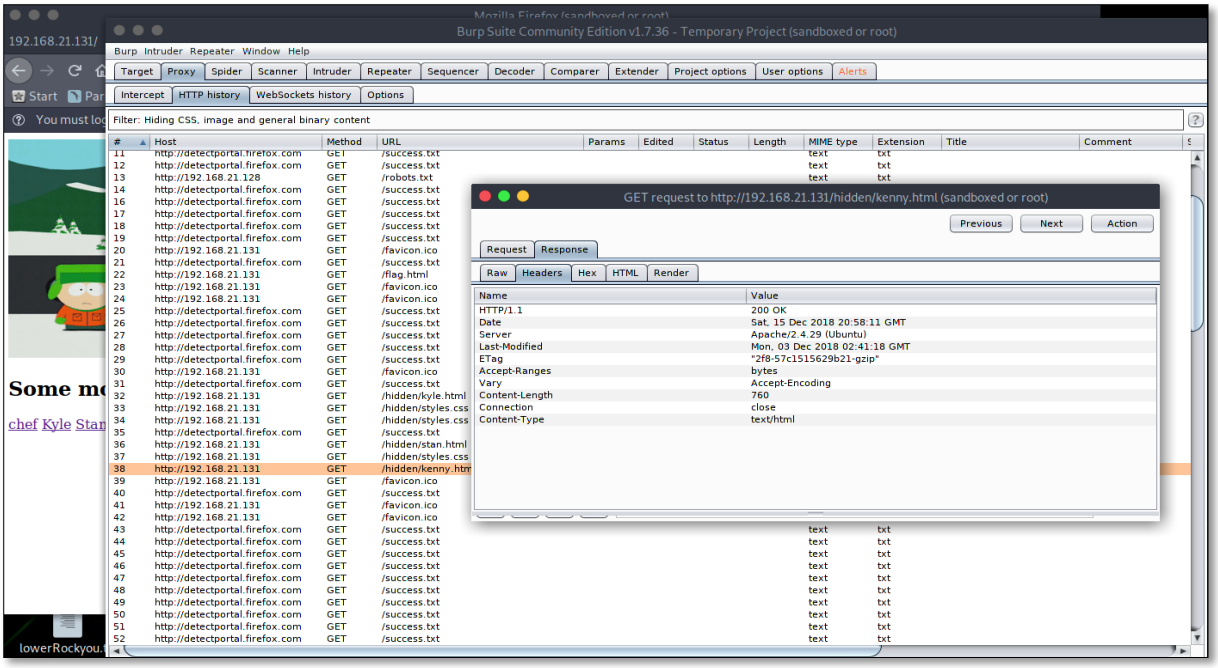
11: Wireshark& The Finding of Flag3

While our brute force attempts to crack either the Shadow files retrieved or the SHH log in directly continued, we turned out attention to the remaining flags.

We ventured over to the hidden links revealed by examining elements on the /hidden/chef.html page. After examining all of comments and finding little of relevance we found ourselves really stuck for ideas. So, we decided to change our attack plan, and rather than focus on clues given, or data mined, we approached the problem from a tools perspective. What tools have we covered this year, and not used? And what areas might these tools cover that we have yet to examine in any great details?

After a long time, we stumbled on a possible answer. Networking! While strictly speaking networking was not a part of the material we covered in any real meaningful way, it is also true that when it comes to penetration testing no stone should be left unturned that is in scope.

That being said, we were not hopeful that a network analysis would turn up much in the way of useful information. Our first thought was to attempt to use Burp Suite, and so; we set out to run a man in the middle sweep across every page we had discovered so far on the website.



Burp Suite results being examined.

No new information was discovered here. However, it was noticed that some of the flags cold have discovered here very easily. In has been noted that in future attacks all webserver probes should be conducted in conjunction with Burp Suite for a more in depth thorough analysis. This in itself is a very useful lesson.

We then moved onto Wireshark intercepts. We shall preface this section by pointing out that in the area of networking, as it pertains to penetration testing, we have not had much time to develop our skills in this area, and this process was long, painstaking and very difficult.

We began with simple network traffic captures, which we continued for to watch for several hours in an effort to learn what we were seeing. This led to some fundamental questions. Why are seeing traffic from the South Park VM, when we are not engaging with this VM. Why would we see traffic between this VM and its network address? Until finally it dawned on us that we were seeing all traffic related to the internal virtual adapter. This didn't further our investigation until one of group re read the brief during a break and wondered aloud... "Why is it suggested that we use the IP address of '192.168.176.132/24'"? Could this in and of itself be a clue?

We began to investigate the assigned IP addresses and discovered that each of our VM's was assigned different DHCP address ranges. This seemed odd, given that the ranges seemed random. Again, we asked ourselves why the recommended IP address? So, we set our IP to this address and immediately lost connectivity with our VM. A quick examination of our networks revealed we were on different subnets, so the loss of connection made sense. It seemed in order to have the recommended IP, and still communicate with our VM we had to ensure that it would be on the same network. Understanding how to achieve this we not easy. The process of attempting to achieve a subnet which would place our South Park VM in the network of '192.168.176.0/24' was long. We did not fully understand what we were doing and thus we set out to attempt to use the entire range of the '192.168.176.0/24' network.

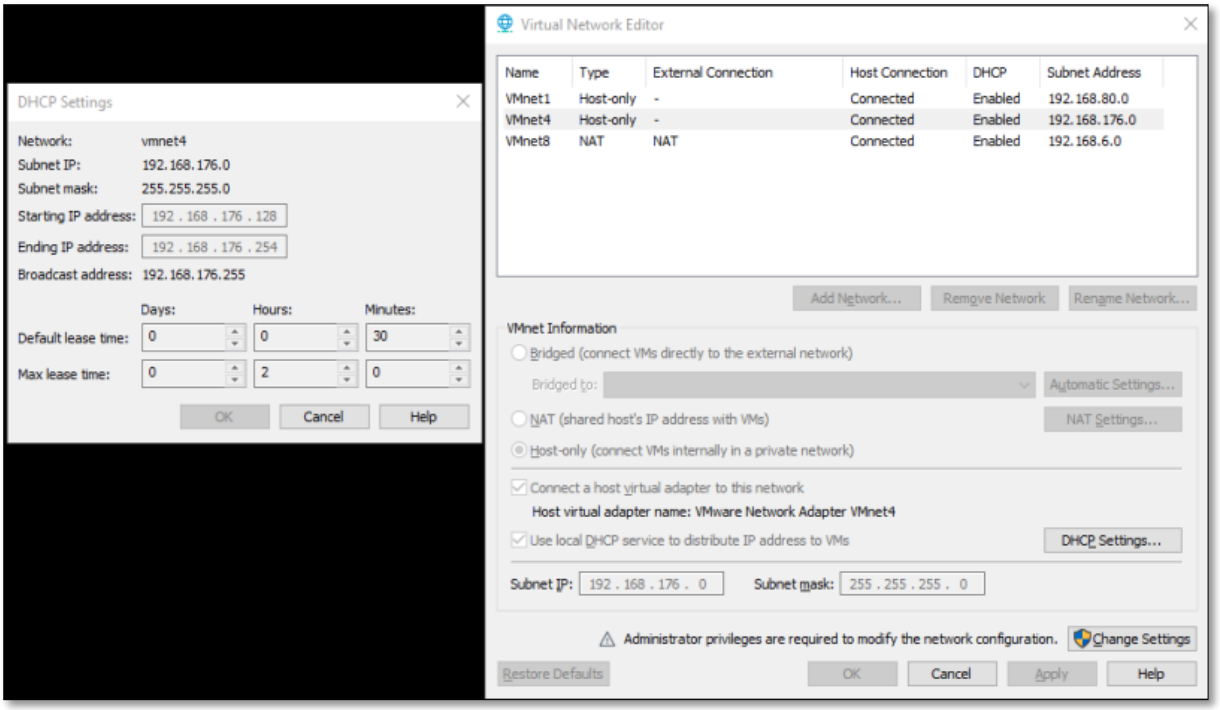
We ran into great difficulty in having the South Park VM be assigned a DHCP IP in this range. However, we stumbled across some very useful information. Which was as follows…

DHCP Conventions for assigning IP addresses in 'Host-Only' adapters in Virtual Machines:

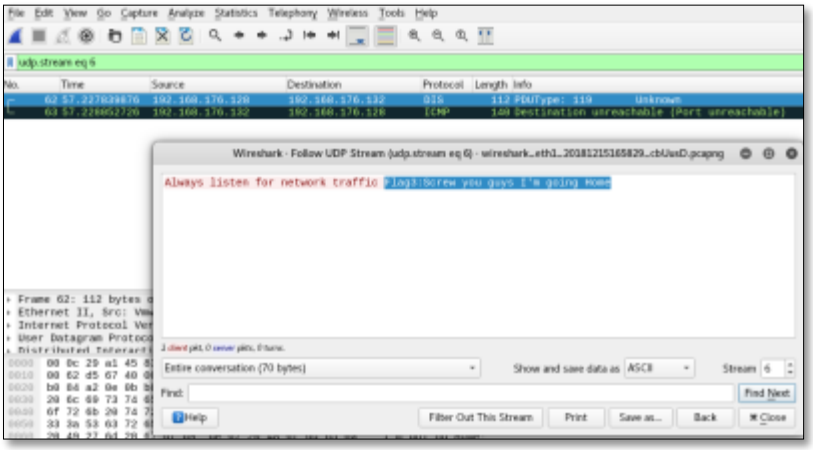
- Network address: always **.0**
- Host Machine: always **.1**
- Static Addresses: always **.2 -.127**
- DHCP Addresses: always **.128** - **.253**
- DHCP Server: always **.254**
- Broadcast = **.255**

We studied this for some time, and while conventions are not rules, we posited that it was likely that the South Park VM was not being assigned an IP as we were using the full range of the network. And so, we edited our settings to restrict the network to the range that convention states is generally assigned for DHCP addresses as seen above (upper half of the given range .128 - .253). While we cannot categorically why state that this fixed the issue, we did get an IP addressed assigned to the South Park VM within our desired range almost instantly. We then set our VM manually to the recommended IP of '192.168.176.132'.

Our Intention was to begin to rerun all of our previous steps, and intercept traffic in the hopes of revealing some previously unseen data. This proved unnecessary as within 60 seconds of establishing a connection we saw some very strange network traffic with some strange protocols. We jumped right in to analyse this traffic.



Here you see above that we altered the subnet to use only the upper half (DHCP) of the network range.



*Here we see **Flag 3** Revealed to be: "screw you guys I'm going home".*

As we see above, the DIS protocol reveals F**lag 3**to be "Screw you guys I'm going home". This was highly unexpected. We had begun to suspect we might have a previously unknown exploit route revealed to us, but we did not expect a plain text Flag in network traffic.

12: Bonus Recon Flag

Our investigation into this flag incorporated elements of our previous methods. First, we visited all links on the web server containing external links and we ran these URL's past Burp Suite but did not discover any new information.

We then collated all external links we had retrieved. They were as follows:

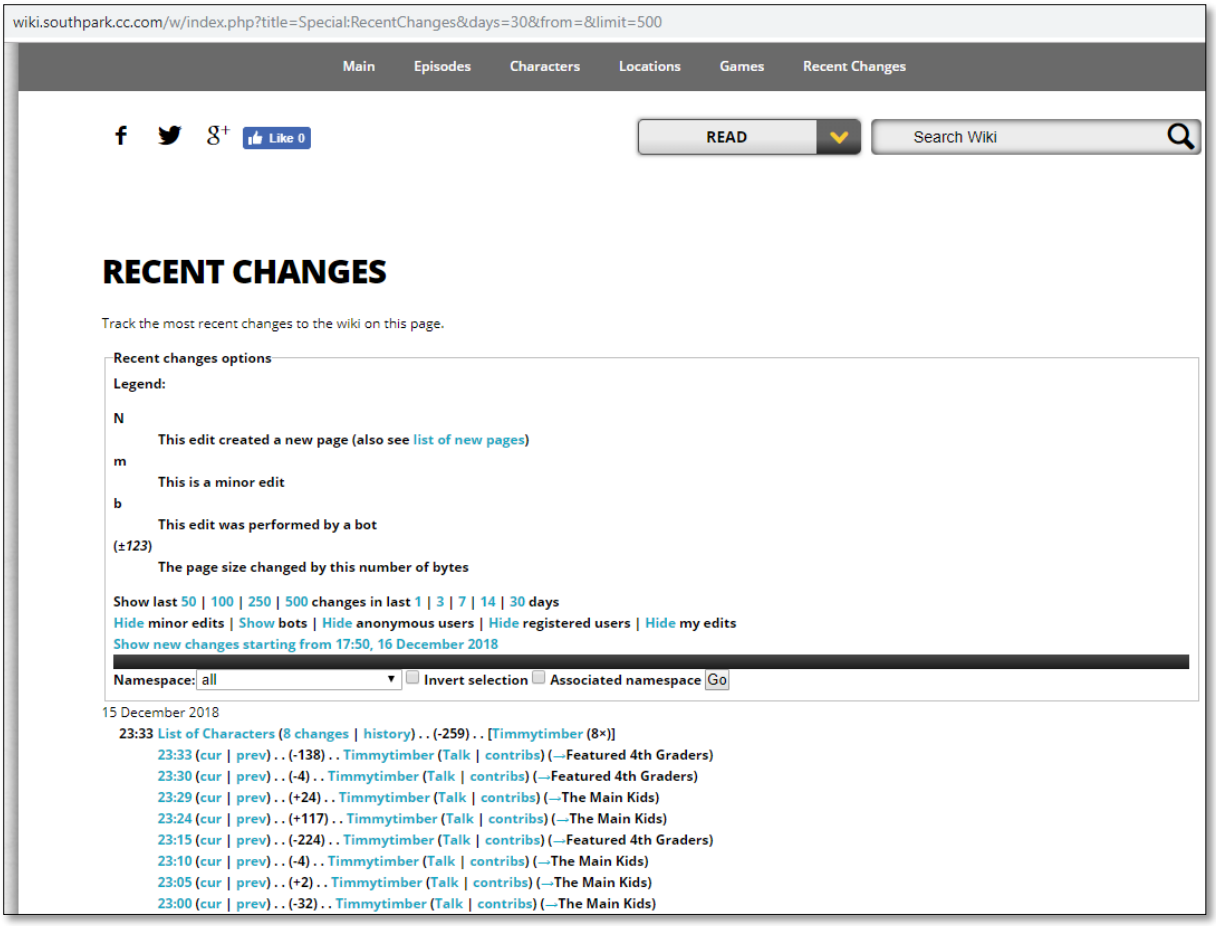
- (secret recipe)*
  https://www.youtube.com/watch?v=7jdsx4zMeB4&list=RD7jdsx4zMeB4&index=1
- https://www.youtube.com/watch?v=uAo0xXn_Syk (Love gravy should be a flag)*
  *We expect the clues provided with these links are important but have yet to see a pattern to them
- http://wiki.southpark.cc.com/wiki/Eric_Cartman
- http://wiki.southpark.cc.com/wiki/Kyle_Broflovski
- http://wiki.southpark.cc.com/wiki/Butters_Stotch
- http://wiki.southpark.cc.com/wiki/Stan_Marsh
- http://wiki.southpark.cc.com/wiki/Kenny_McCormick

We visited each external URL and began to scour the information provided. In the case of the two YouTube links, this meant reading every single comment. While the phrase 'Lovegravy' did appear on page:https://www.youtube.com/watch?v=uAo0xXn_Syk it did not seem to relate to a flag. No other useful information was found in You tube comments. We then tried to view the page through the internet archive 'Way Back Machine'. However, it seems this service has great difficulty in archiving and displaying YouTube comments. While the page was viewable, the comments failed to load.

We decided to take a look at the account of the video poster, and found a video labelled 'love gravy' had been posted to the account and pulled due to copyright infringement. This got us wondering if perhaps the flag had been posted as a comment and deleted and as previously mentioned comment do not work with archives, so; could it be the flag was gone? We agreed to persist a little more before checking this.

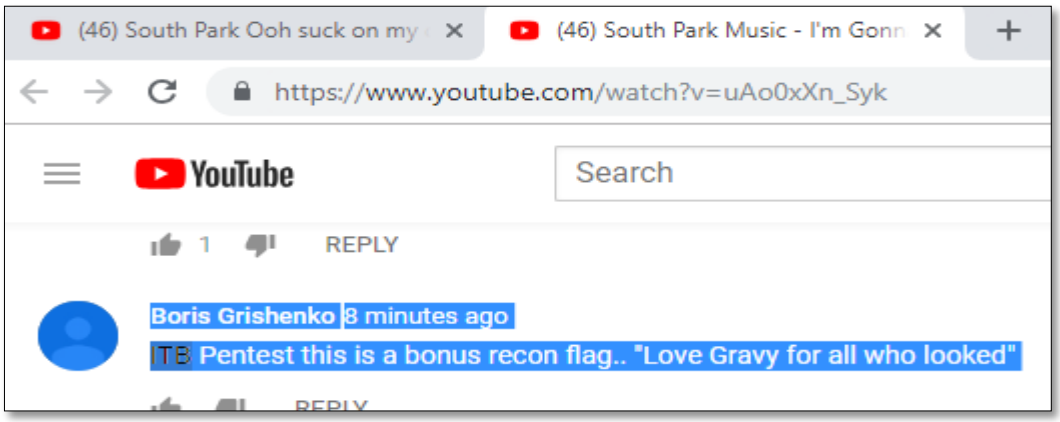
We then decided to scan through all of the text on the wiki pages listed above but again found no relevant data that seem to pertain to a flag. The combination of the clues 'secret recipe' and 'Love gravy should be a flag' lead us to suspect that there may be an actual recipe for a dish called this inspired by south park and this recipe may be the flag but we have yet to find it.

Finally, using the 'recent changes' link, and altering the number of days of 'recently changed data' days to 30, we examined the changes to see if any previous changes had been made to the pages that might indicate that a change had been made to exhibit a flag and then deleted after being cached (basically a self-contained Way Back Machine, similar to Wikipedia). Again, this returned no results.



Here we are, examining the changes to the wiki-chef page for potentially stored then removed flags.

At this point we felt reasonably assured that our hypnotises that the flag was a deleted comment was likely to be correct and contacted the issuer of our VM, our lecturer; who confirmed it was a deleted comment. Naturally after a reposting of flag we found it within seconds. The **Recon Flag** is: “Love gravy for all who looked”.



*Here we see the discovery of the **Recon Flag:** “Love gravy for all who looked”.*

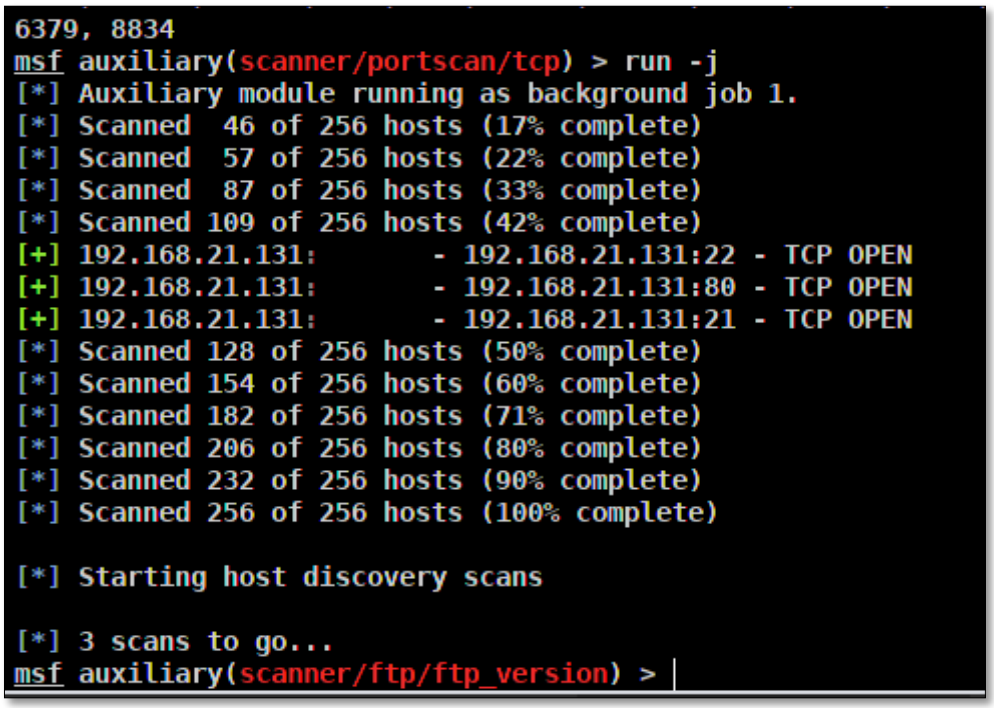
13: Metasploit /Armitage

13.1: Armitage Hail-Mary

It seemed logical at this point having more avenues explore while we continued to brute force shadow files and the SSH logins, that some time should be given over to attempting to force exploits the VM. While our earlier checks told us, no exploits should be available, it is always worth giving it a go physically to confirm this.

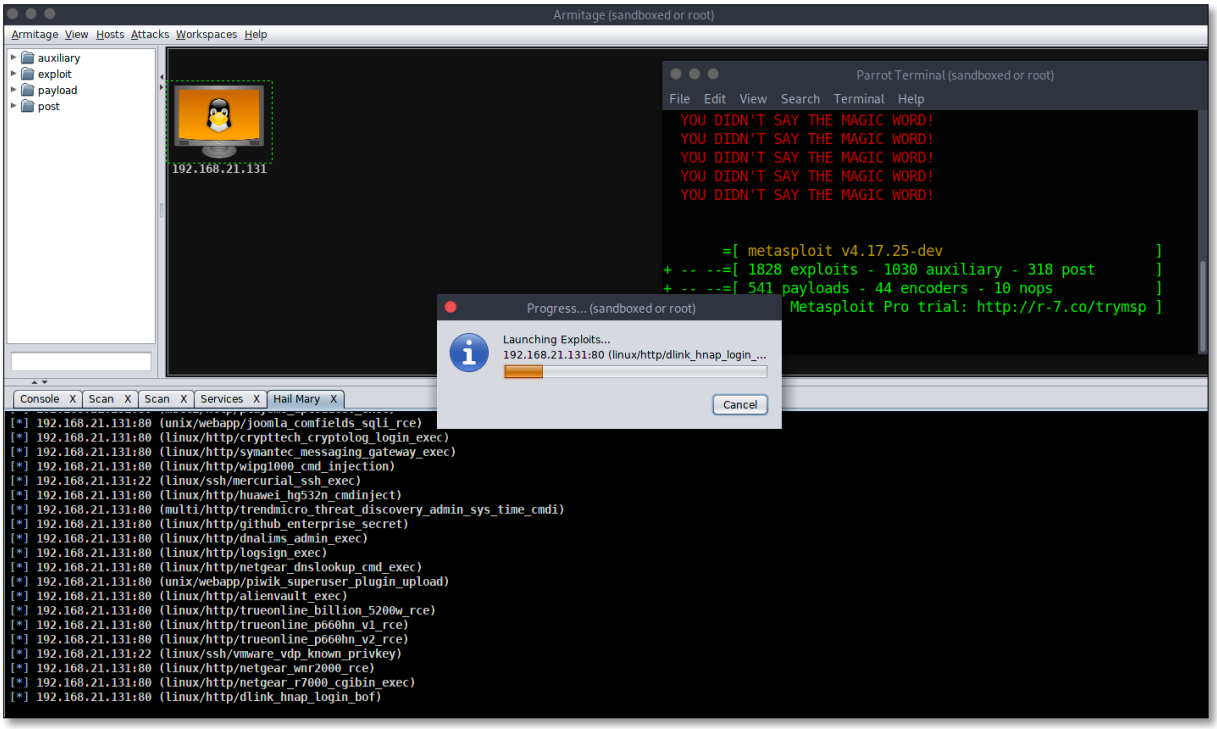
We first loaded up Armitage and executed a brute force 'Hail Mary' attack. We Fired up our postgresql service with the command: service postgresql start. Next, we checked to see if our database was initialized with the command: msfdb init. Next' we started up the Metasploit framework with the command: msfconsole. Finally, we launched the Armitage application through its menu shortcut.

Once opened we initiated our msf scans by selecting 'Hosts', msf scans; and added the network both our host and VM were running on at that time which was the 192.168.21.0/24 network.

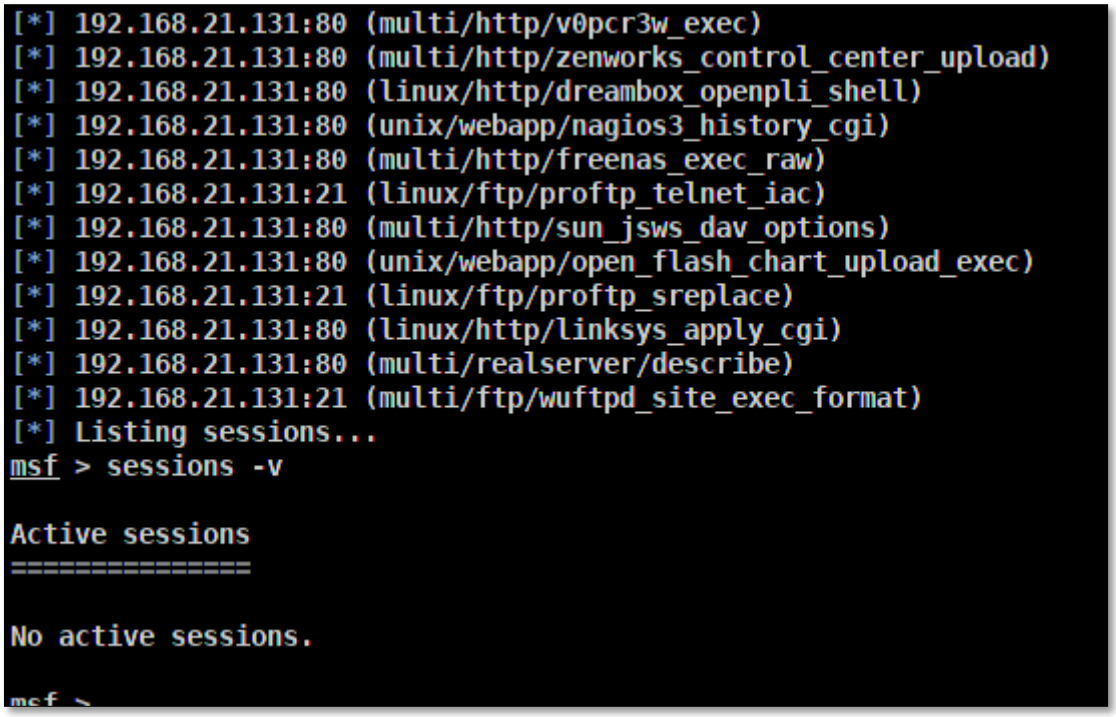


Here we see our VM detected during our msf scanning process.

Once we had our scans completed, we launched our Hail-Mary attack against our VM.



Here we see our Hail Mary attack launching.



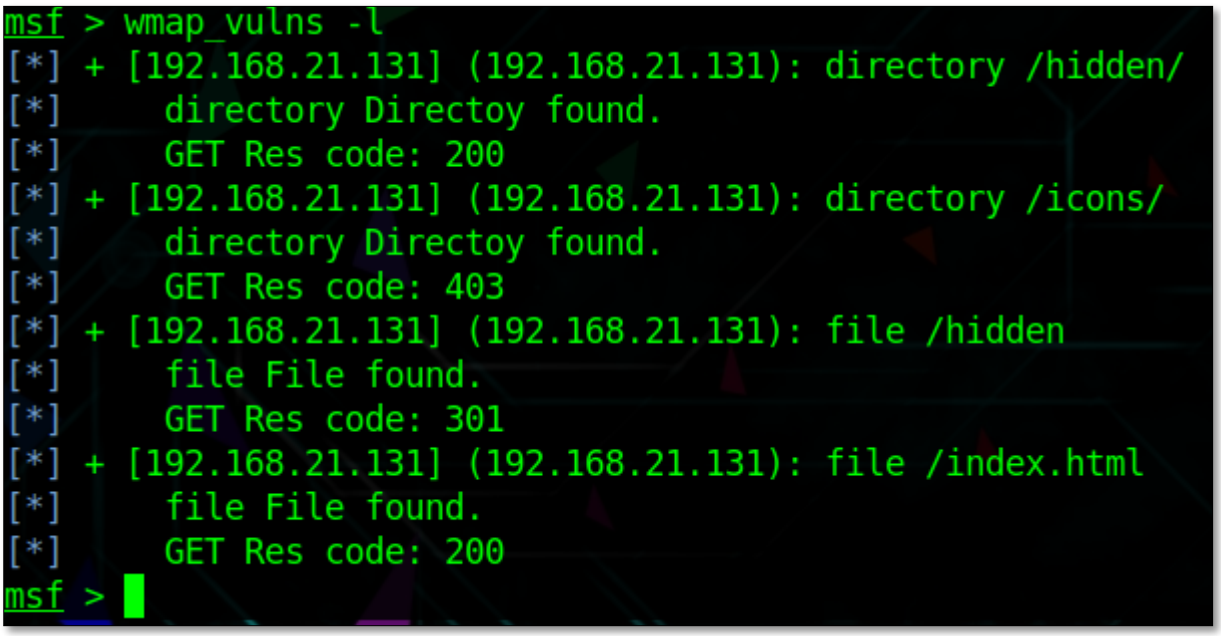
Above, we see the Hail-Mary has failed to create any shell sessions on our VM.

While this failure was anticipated, it was still mildly disappointing. There is always that small chance that a random exploit might succeed. At this point, we have exhausted all of the more obvious options and we determined to try to really think outside the box.

13.2: Metasploit WMAP

In an attempt to determine if the webserver had any as yet undetected vulnerabilities we attempt to perform a WMAP vulnerability assessment. The steps we performed and the commands to execute them to run this scan are listed below:

- Start postgresql service: service postgresql start
- Check if database is initialized: msfdb init
- Launch Metasploit console: msfconsole
- Load WMAP module: load wmap
- Add site to available sites: wmap_sites -a http://192.168.21.131
- Add target page to list of targets: http://192.168.21.131
- Add target page to list of targets: http://192.168.21.131/robots.txt
- Add target page to list of targets: http://192.168.21.131/flag.html
- Add target page to list of targets: http://192.168.21.131/hidden
- Add target page to list of targets: http://192.168.21.131/hidden/chef.html
- Add target page to list of targets: http://192.168.21.131/hidden/Kyle.html
- Add target page to list of targets: http://192.168.21.131/hidden/Stan.html
- Add target page to list of targets: http://192.168.21.131/hidden/kenny.html
- Launch full scan with all local modules against target/s: wmap_run -e
- Examine scan results % any vulnerabilities found: wmap_vulns -e



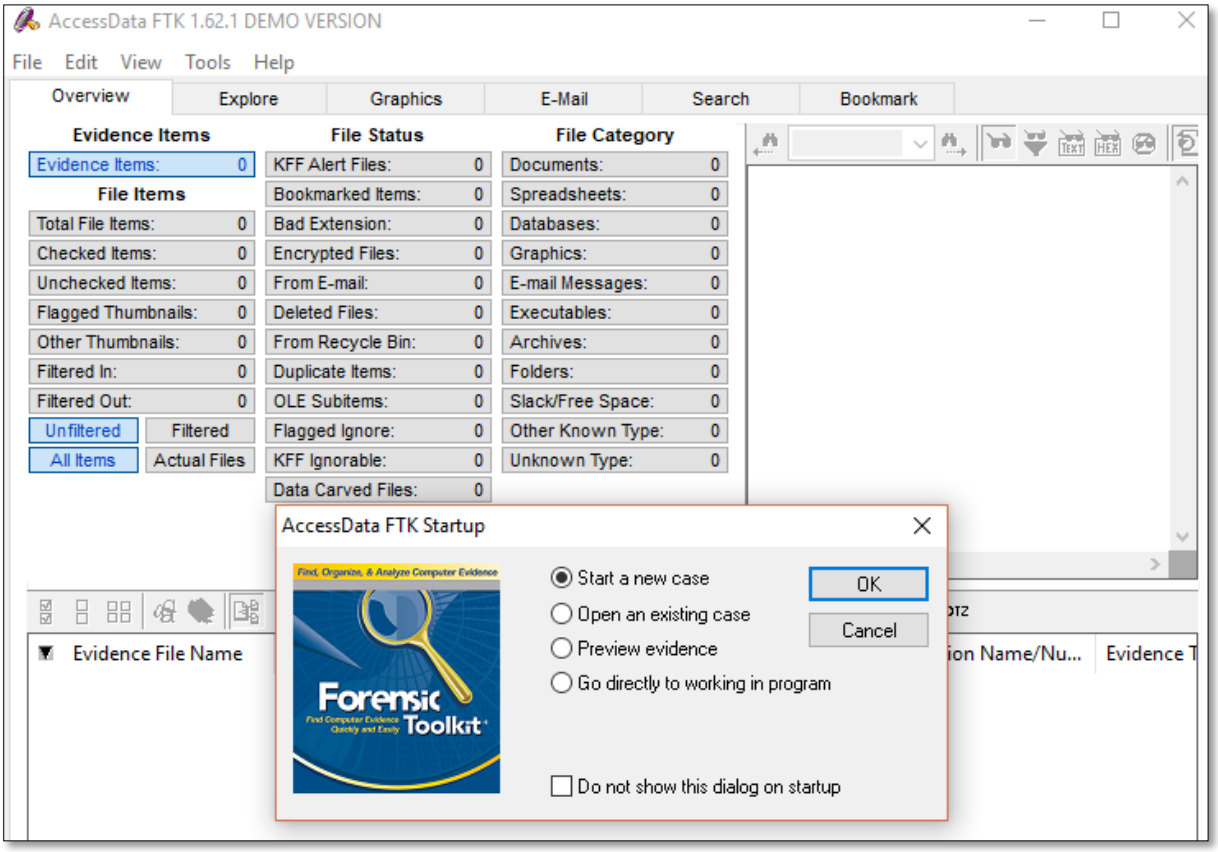
Here we see the results of our WMAP scan via Metasploit.

As we see above, the scan did manage to find what are ostensibly hidden folders, however these did not prove of any real use as most had been previously discovered and the others, such as /icons did not hold any useful information that we could discern anything from.

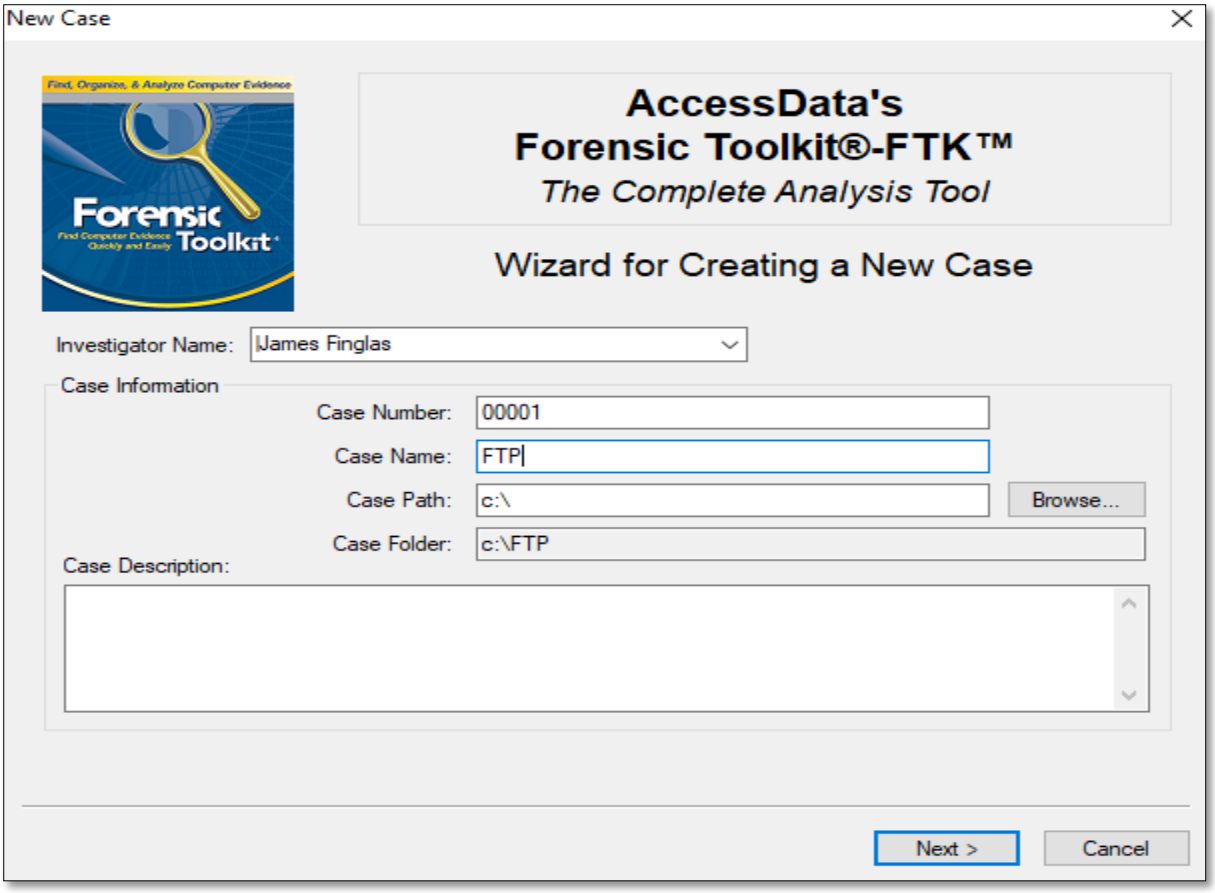
14: Forensic Toolkit

In an effort to determine if any pertinent files had been deleted from the folder structure transferred via ftp, we transferred the folder to a USB and over to windows 10. We selected Forensic Toolkit to perform our analysis.

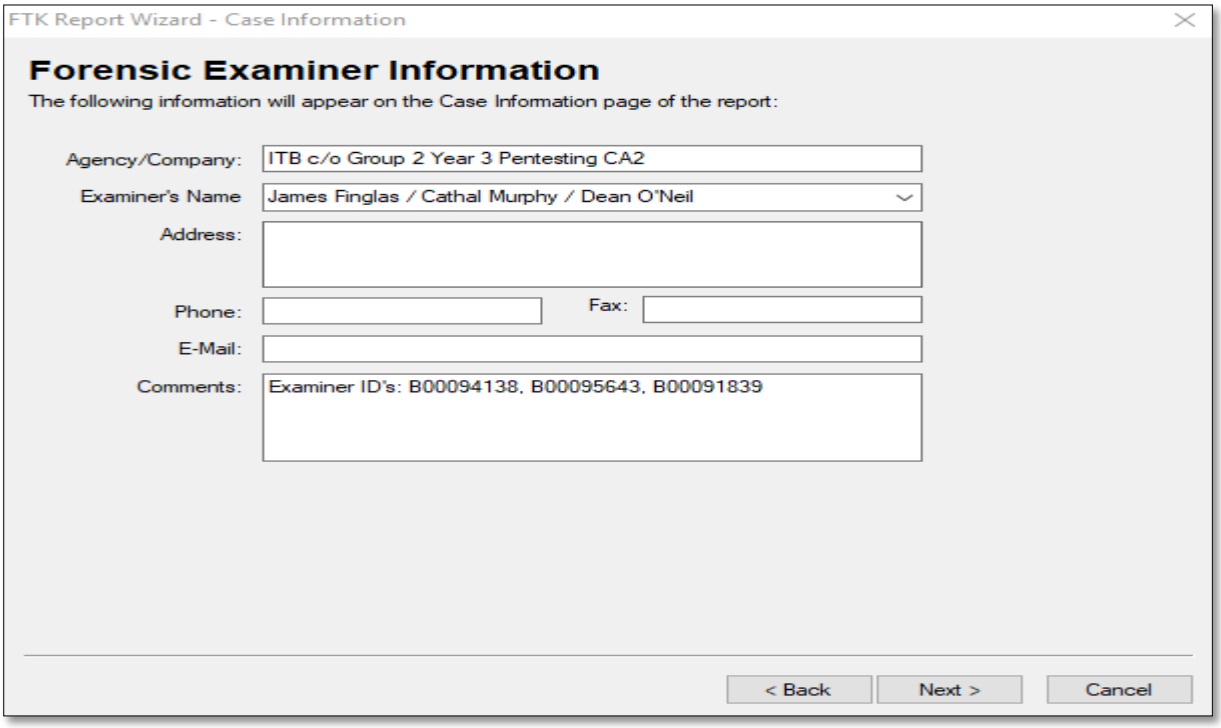
Having opened the programme, we first opened a new case. We gave this case an ID of 00001 and a name of 'FTP'. We then entered our details relating to our company and the investigators as in the third screen shot below. The case log option required no modification. And then we simply selected next on all screens until we add our evidence file. The location of this file will differ from your own. In our case we navigated to the USB stick and select the folder we had transferred from our VM.



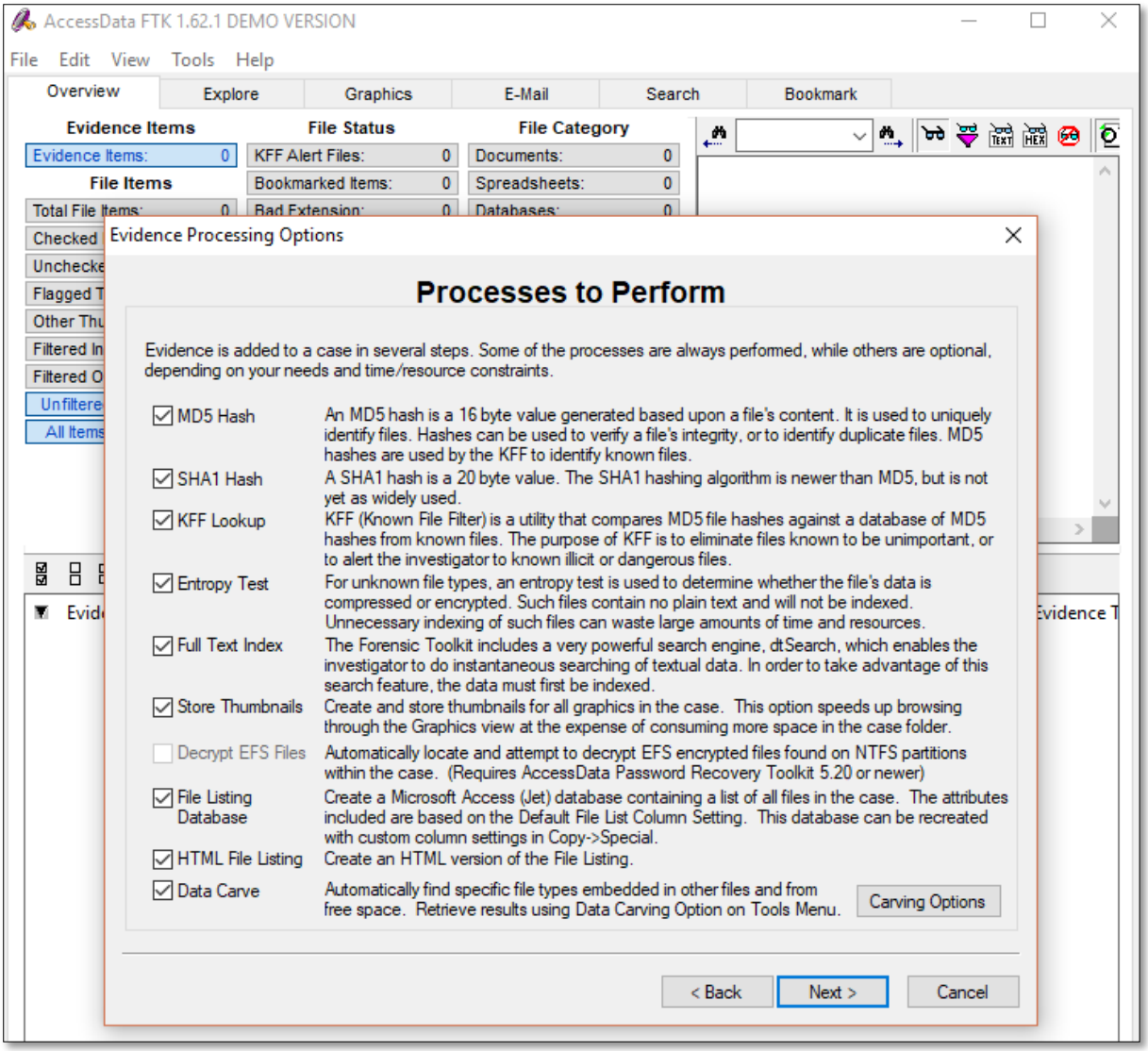
Here is our new case being opened.



Here we assign an ID and name.



Here we enter our Agency/Company, and identification details.



Here we simply add HTML file listing and data carve to our option set.

Our investigation did not reveal any deleted files or slack areas to investigate. No further information was discovered among the files previously discovered and investigated.

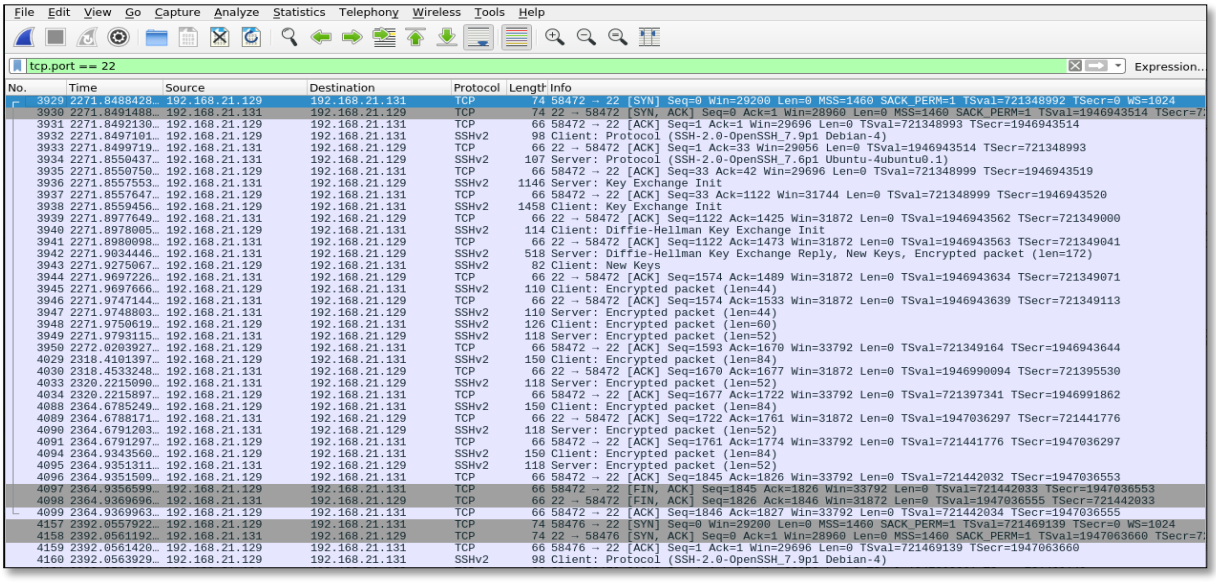
15: FTP Revisited 2 Resulting in SSH Access & The Finding of Flag 7

Having exhausted all options, our team once again returned to the FTP server. This being our primary source of files thus far seemed a likely source for whatever we had missed that had led to our stalling in our attempts to penetrate the server.

However we once again found ourselves hitting that brick wall of permissions. Although much time was given over to the pursuit of exploitation of the VSFTPD server or somehow tricking the server into thinking we had admin access, or local access as if we were logged into the server, all attempts to go down this road failed.

Our last and most desperate approach was to capture as much network traffic as possible in an effort to see if had somehow missed a secret message being sent.

While we did of course capture some traffic most of it seemed useless. We did however capture a significant amount of SSH traffic when we were not attempting to log in. This was quite curious and unexpected.



Here we see a sampling of our captured SSH data.

Here we see a sample of the key exchanges captured.

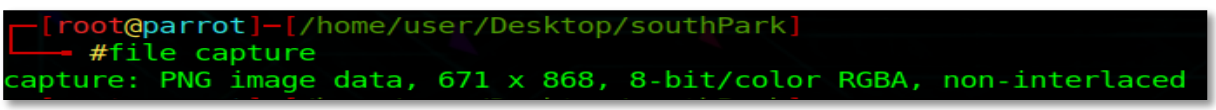
Our investigation into the possibility of cracking this captured data was rapidly acknowledged to be a rabbit hole leading us down a long road to nowhere. SSH being the secure protocol that it is incredibly difficult to crack, if indeed it can be cracked at all. We have only rumours to go on that intelligence agencies such as the NSA 'may' have cracked 'some' SSH intercepts but it seems that even if they did it is a highly sketchy process involving massive levels of computational power. Suffice to say, it is far beyond the scope of our brief and skillset of our group.

This however led us to conclude that there must be a problem. We simply did not appear to be able to retrieve these final files from the FTP server. So, we contacted the provider of the VM image, in this case our lecturer. Upon investigation it was discovered that in fact an error had occurred, and the files ended up having the wrong permissions.

While this did result in the group losing a lot of time, this was not a negative thing. This forced us to experiment, try many avenues of attack we might not have considered and to sharpen our skills with several tools we had previously not worked with extensively. In point of fact, it could be reasonably argued that the fault was with our group failing to come forward sooner with a problem we felt sure was present. Valuable lessons were learned here. The problem files were provided directly so that the test could proceed.

This provided us with 2 new files to examine. A file labelled 'capture', and an image file labelled 'index.png'. The 'index.png' was examined via file, and strings as well as a visual inspection and it was discovered to be the image used as a banner image in the 'index.html' webpage deployed on the webserver. No other pertinent information was found.

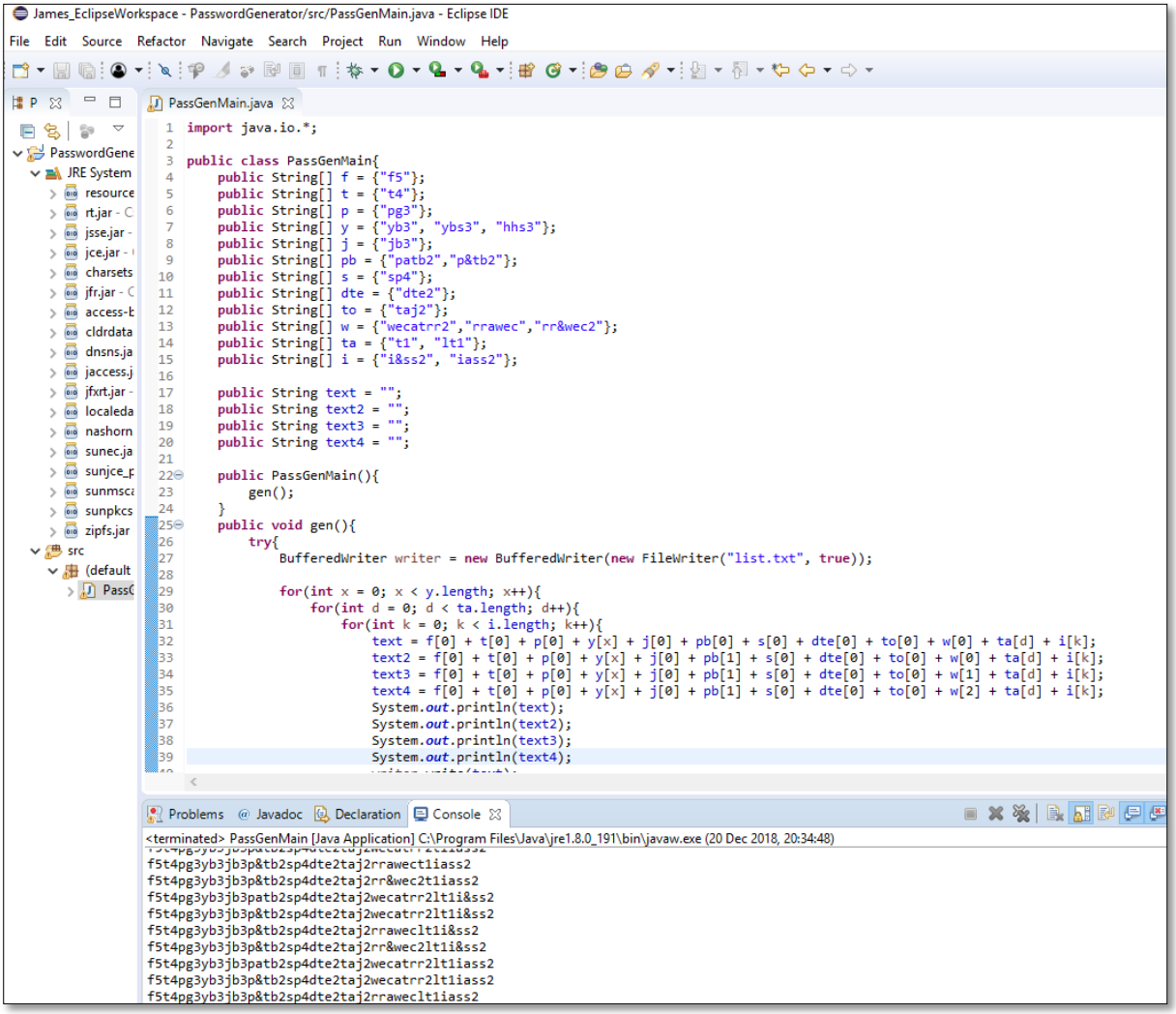
The second file 'capture' was also examined via file and strings. The file command revealed that the image was a png image.



Here we see the capture image extension revealed.

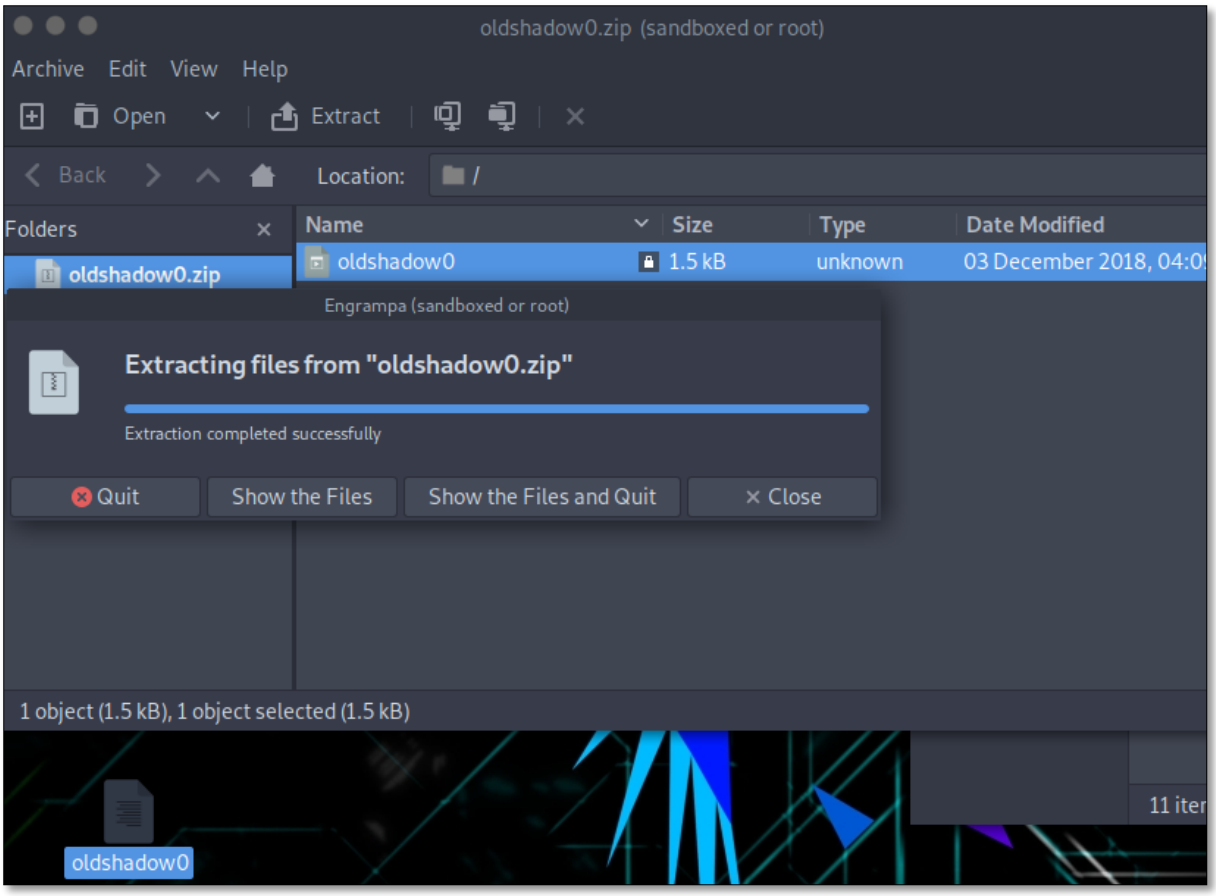
Next, we ran strings on the file, but this revealed no new information. Finally, we viewed the file as an image by right clicking and selecting an appropriate program and it was revealed to be another pixel art image similar to the one which gave us a zip code password previously. We knew just what to do and since we had some experience this time we approached it a little differently.

We wrote a script to generate a password list in java which we compiled and executed in the Eclipse Ide. **(1a)** ***The source code is provided in the appendix below.***



Here we see our password generator being executed via the Eclipse IDE

Having generated our passwords, we endeavoured to try to log into the SSH accounts we believed existed. As expected this failed. So, we returned to our one remaining zipped folder and began to enter our passwords. A small point of order here, 'FCrackzip' refused to accept our password lists regardless of their format, leaving us to have to manually enter the passwords one by one. This was a bit of a time consuming process, but ultimately led to success. Our final zipped file 'oldshadow0.zip' unzipped with the password of 'f5t4pg3yb3jb3patb2sp4dte2taj2rr2td1ias2'.

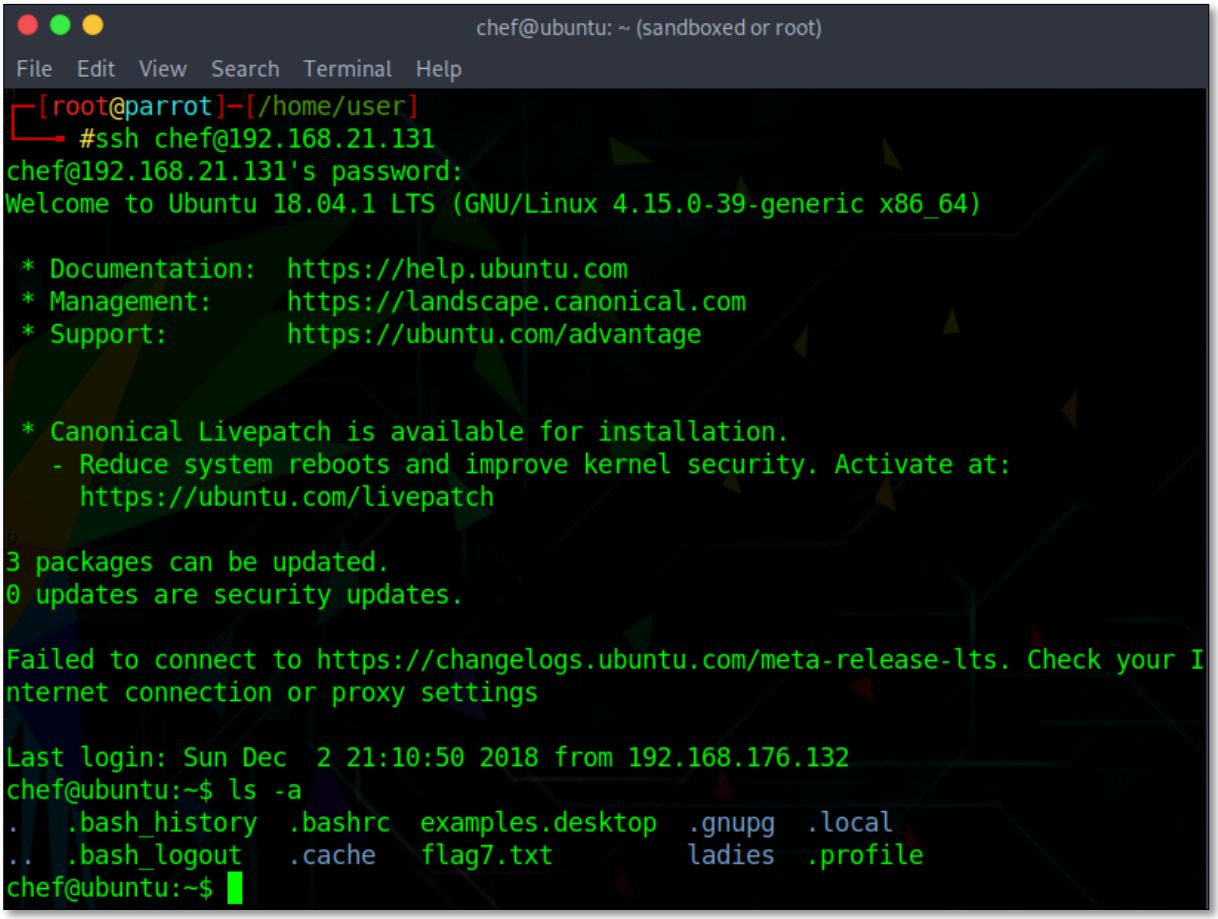


Here we see our final zipped file 'oldshadow0.zip' being unzipped with our password 'f5t4pg3yb3jb3patb2sp4dte2taj2rr2td1ias2'.

We viewed the file and found it was indeed what appeared to be a legitimate shadow file and set out to attempt to crack it with 'HashCat' in windows.

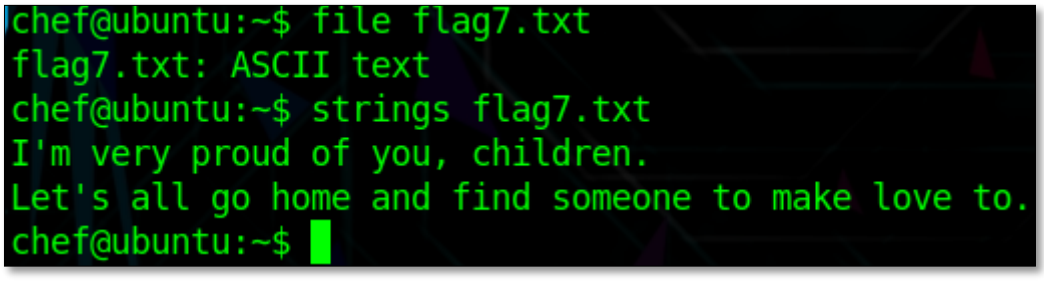
Interestingly our attempt to crack this shadow file failed. The program detected 2 hashes, and immediately discarded one hash to the hashcat.pot file. The final hash failed to crack. This left our group quite puzzled. And we decided to attempt to guess that password. Our cracked shadow file revealed two passwords: 'gravy121' and 'gravy123'. So as an educated guess we decided to start from 15 digits below the lower password and continue to 15 digits above the password. So, we attempted to log in using the command 'ssh chef@192.168.21.131' and the password 'gravy127'.

we had intended to continue this enumeration up to the password 'gravy138', however we successfully achieved a log in with '**gravy127**'.



Here we have our first successful SSH connection to the server. We do not yet have admin access however, as we can see a file labelled flag7.txt can be seen.

Here we see our first successful SSH onto the South Park VM. As you see above, with the command 'ls -a'; a file labelled 'flag7.txt' can be seen. **Flag 7**is revealed to be "I', very proud of you, children. Let's all go home and find someone to make love to.".



*Here we see **Flag7**revealedto be "I'm very proud you, children. Let's all go home and find someone to male love to.".*

16: SSH Access as ‘Chef’

As chef we examined the user personal folders and directories. A folder labelled ‘ladies’ was discovered, examined and found to contain a number of image files. They were as follows:

- 52828-south-park-chef-s-luvshack-windows-screenshot-chef-s-opening.jpg
- Chef_&_Lee_Gifford.png
- Fa0a0d10baaeccf9f3327654ff303b92.jpg
- Giphy.gif
- Index.jpeg
- PARK03.jpg
- Pics_214_5.jpg
- Top-red-womens-southpark-chef-pyjama-set-red-product-1-22364774-1-624095849-normal.jpeg

Also, there was one file which could not be accessed due to restricted permissions which was:

- Jxvxdj.png

Following the protocol previously established we examined these files sequentially using strings and file commands however, no relevant information was discovered. When began a lengthy investigation of the other files and folders.

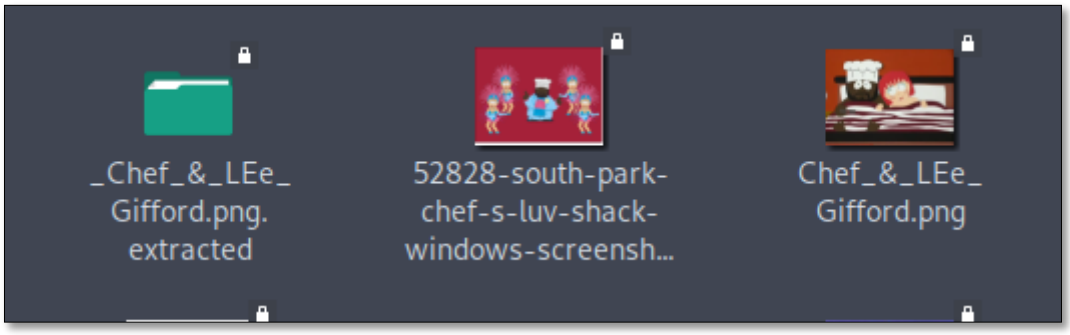
We discovered 4 user profiles ‘chef’, ‘cartman’, ‘stan’ and ‘kenny’. We believed we already had passwords to the ‘chef’ and ‘kenny’ accounts and thus we focused our efforts on the ‘stan’ and ‘cartman’ accounts. Using our knowledge of the South Park show we suspected that ‘cartman’ would prove to be admin as he generally demands a lead role and that his authority be respected.

We examined the /etc folder and attempted to recover the shadow file. However, our permissions were restricted. We were able to recover the ‘passwd’ file but this was not much use on its own. We then viewed the group file and our suspicions regarding the ‘cartman’ account where proved correct.

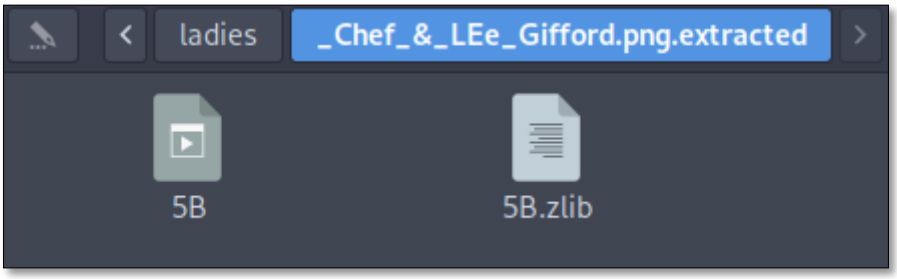
Here we see our suspicions regarding the ‘cartman’ account confirmed. The account is the admin account.

We returned to our exhaustive search of the system for files revealing information and could not find anything relevant. All attempts to alter permissions files such as '/etc/ssh/sshd.config' or '/etc/vsftpd/vsftpd_conf' failed due to restricted permissions. All attempts to deploy payloads via Metasploit and Armitage also failed.

Having failed to gain any information we decided to perform a 'Binwalk' examination on the files and something very strange happened. The 'binwalk -e' of the file 'Chef_&_Lee_Gifford.png' seemed to produce a folder called 'Chef_&_Lee_Gifford.png.extracted'. Inside this folder were two files called '5B' and '5B.zlib'. These files had icons which indicated they were music or audio files but all attempts to listen to them failed. We cannot explain what happened here.



Here we see the strange folder whose creation we cannot explain.



Here we see the strange files located in this folder. (second file icon changed due to us changing the file used to open it in an attempt to use the file).

Although we could determine that these files did in fact contain data, we could not successfully determine which program should be used to access the data and therefore failed to retrieve anything useful from these files.

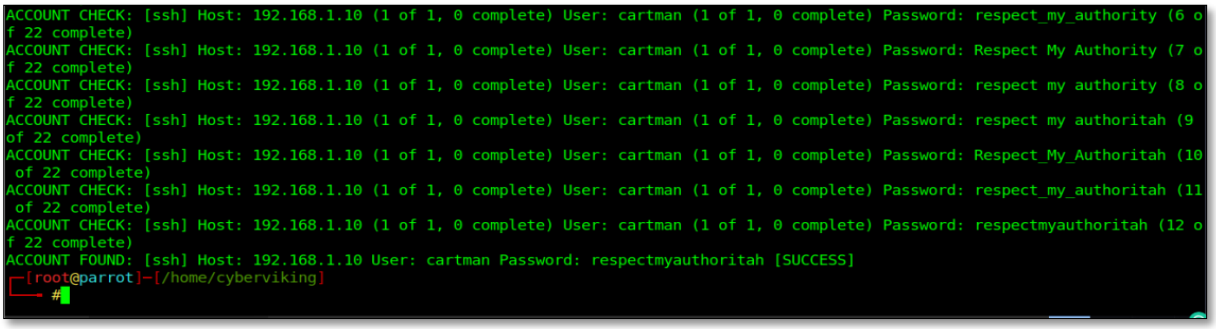
This failure to mine further data and further penetrate the server left us quite confused. We felt no stone had been left unturned. So, we resolved to ask ourselves some simple questions. What have we learned? Do we see any patterns? Is there anything we could attempt to 'guesstimate' from the process thus far?

We learned that cartman is the admin. We learned that the passwords are all relevant to each character and were also related to either a famous catchphrase, or something directly and obviously applicable to the character. We could, in theory attempt to guess the 'cartman' accounts password.

So out of sheer desperation we compiled a simple password list, and I will stress that we did NOT expect this to succeed. We used top recommended quotes as part of our list.

Here we see a sample of our custom wordlist 'sshList.txt' compiled for attacking the 'cartman' account.

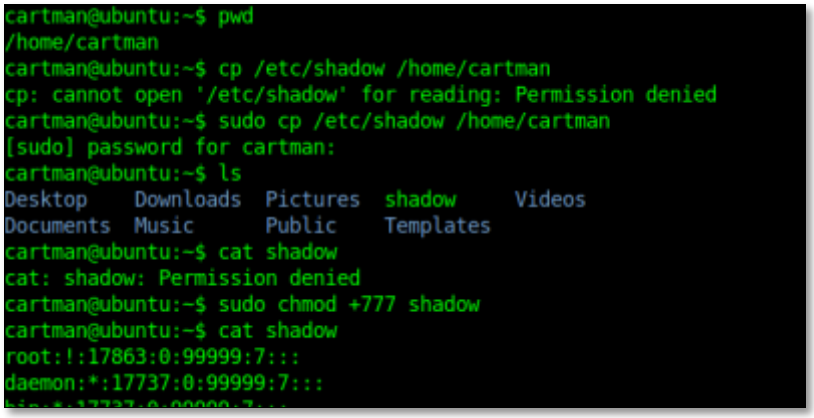
We then ran this list against the 'cartman' SSH account using Medusa and got a hit with the 12th attempt. The 'cartman' account is 'respectmyauthoritah'. The command used was: medusa -u cartman -P sshList.txt -h 192.168.21.131 -M ssh.



Here we see our SSH attempt to brute force the 'cartman' account succeed.

17: SSH as ‘cartman’ and the Finding of Flags8 &9

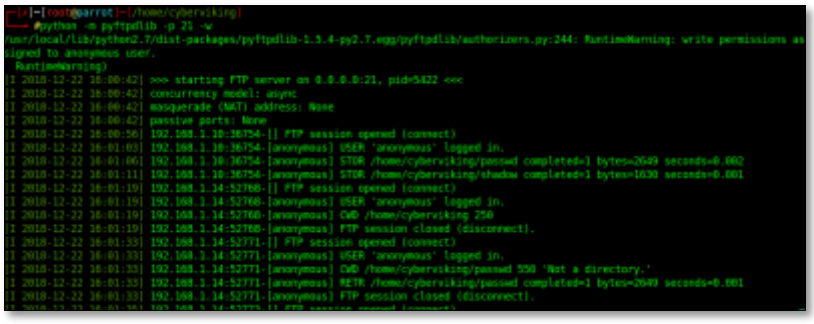
Immediately upon logging into the ‘cartman’ account we made a copy of the shadow and passwd filesand placed them in the ‘cartman’ folder in the home directory using the command ‘sudo cp /etc/shadow /home/cartman’ and tried to view the files. However, due to file restrictions we could not view the files. We therefore used the command ‘sudo chmod +777 shadow’ to grant ourselves full permissions. We could now view the file.



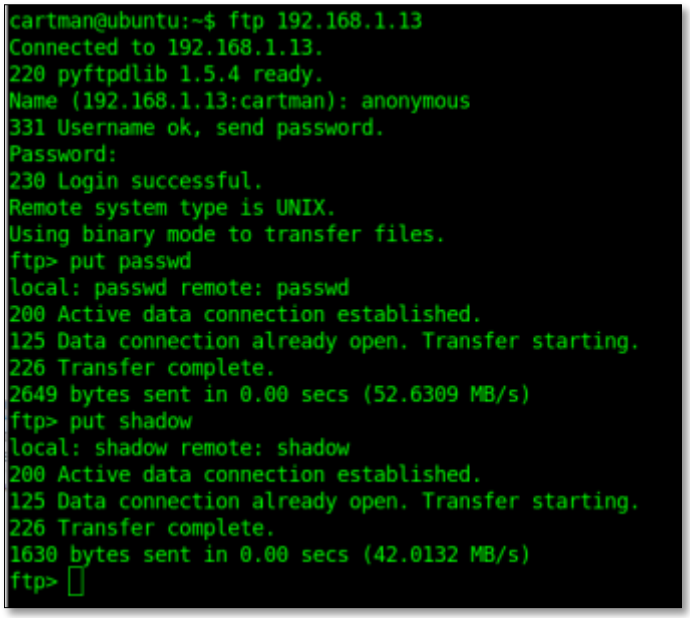
Here we see our copying of the shadow file and altering of the permissions.

In order to transfer the files to our local system, we set up an ftp server using python on our local machine and then executed a reverse ftp connection from the south park VM and transferred the files over.

The command used to setup the ftp server was: python -m pyftpdlib -p 21 -w. **(1)**.

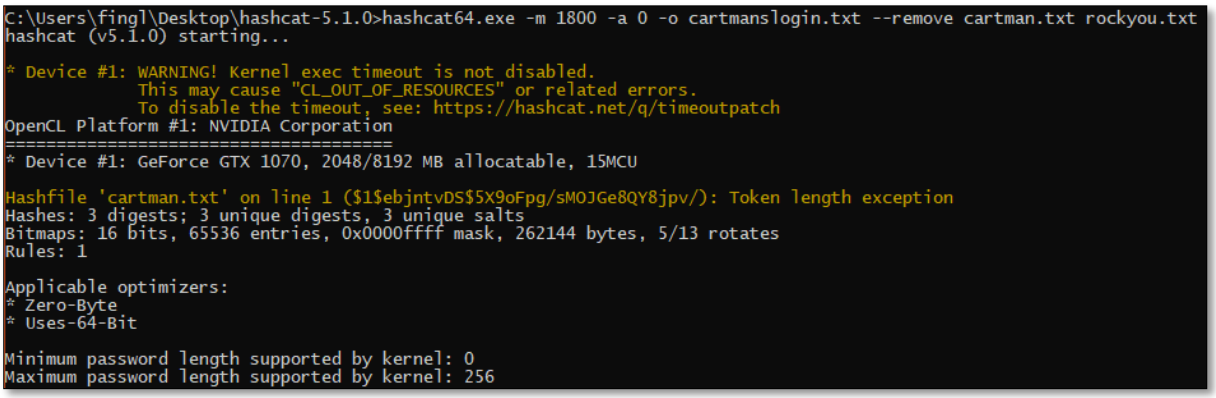


Here we see our FTP server being set up on our local machine.

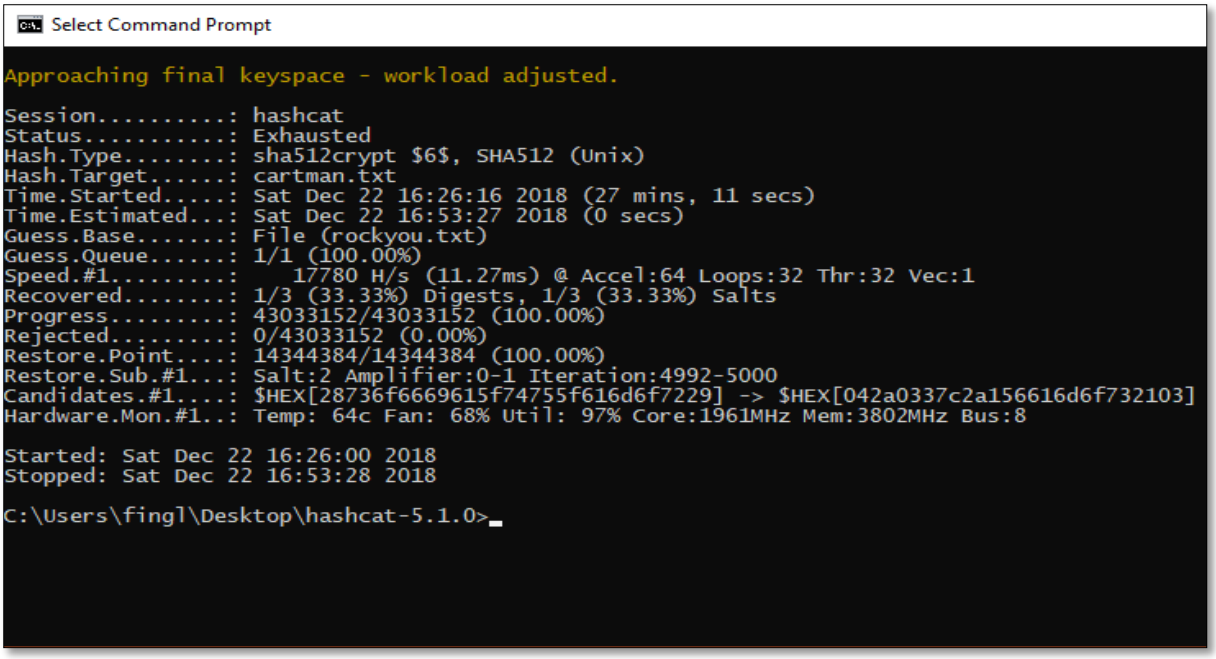


Here we see the files sent to our local FTP server from the south park VM via the 'cartman' account.

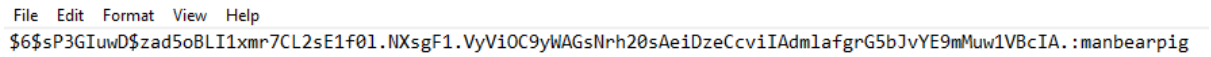
Once we had these files we immediately set about cracking the hashes as described previously. This resulted in a new password being revealed. The account 'stan' password is: **manbearpig.**



Here we see the brute force attack on our unshadowed shadow file retrieved from the 'cartman' account beginning.



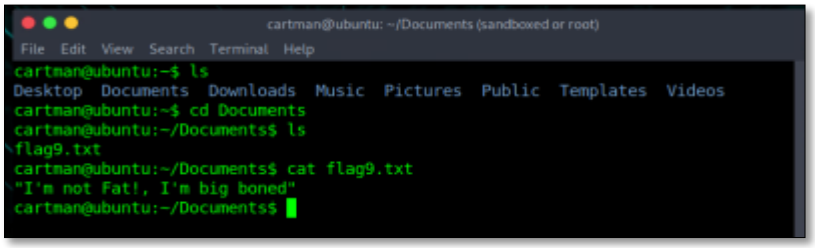
Here we see the output of our attack. One password recovered.



Here we see our new password file reveals a new password: ***manbearpig.***

Curiously, this brute force attack only revealed one further password. We thought perhaps new passwords might be revealed that would allow other accounts elevated SSH permissions. A comparison of the hashed data from the shadow file revealed that our recovered password was associated with the 'stan' account. This meant we now had all of the account passwords **(2a)**. The server is fully breached however, we did not yet have all the flags, nor did we consider our penetration complete.

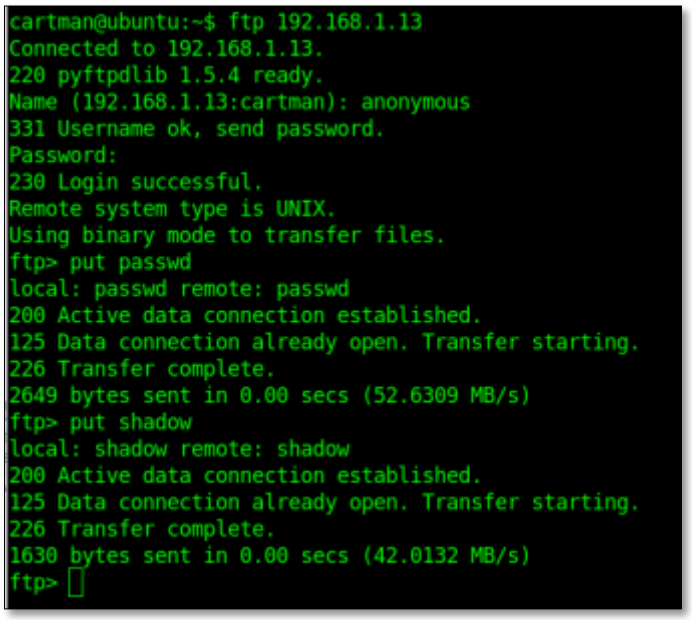
Returning to our file investigation of the 'cartman' account, we returned to the root folder and began a manual search through all files and folders which was rewarded very swiftly. While investigating the folders of the 'cartman' profile we looked inside the documents folder and found a flag. This flag revealed that **Flag 9** is: "I'm not Fat!, I'm big boned!".



Here we see **Flag 9** revealed to be: "I'm not Fat!, I'm big boned!".

We continued to investigate the ‘cartman’ profile for some time but found no relevant information or flags. We stepped back at this point, knowing we had only one flag left to reveal, and realised we had not yet touched the ‘stan’ account. It seemed likely that the ‘stan’ account would be the most likely hiding place for this final flag.

While still on the chef account we decided to investigate the ‘stan’ profile. However, found that we had no permissions to view this profile. In an effort to see if this could be circumvented we executed the command ‘sudo ls -la stan’. This revealed to immediate contents of the ‘stan’ profile.



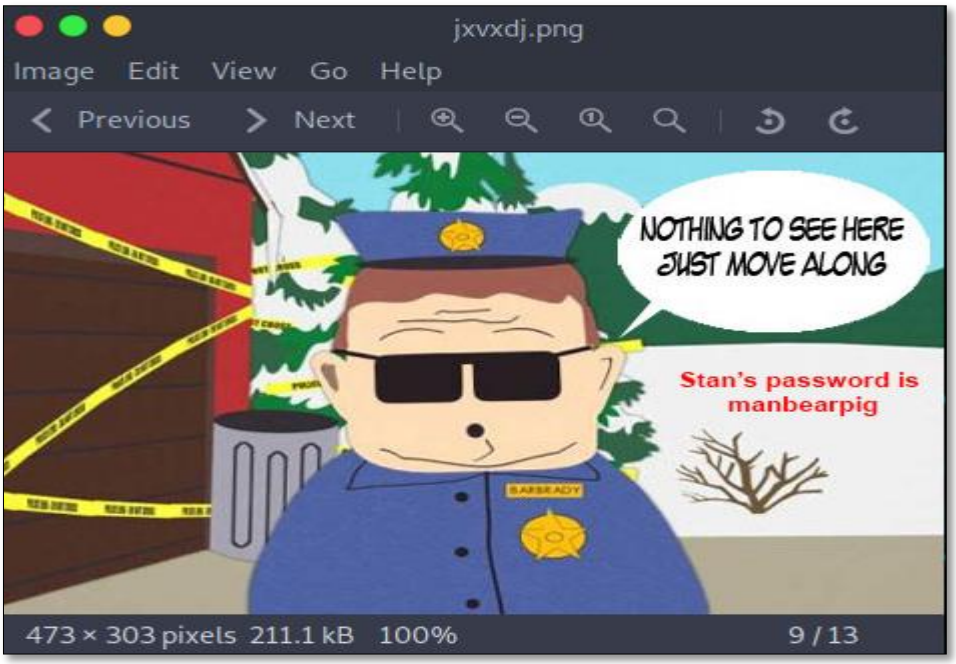
*Here we see the contents of the stan profile directory, revealing **Flag 8.***

As we see above, we have revealed the location of the final flag. However, we run into one final hurdle in that we cannot access or view this file again due to restricted permissions. In order to circumvent this last hurdle, we used the command ‘sudo cp stan/flag8.txt ./flag8.txt’ to copy this file onto the ‘cartman’ account. This allowed us to finally view **Flag 8**. **Flag 8** is:“Dude this is pretty fucked up right”.

```
drwxr-x--- 2 stan stan 4096 Dec  2 21:03 .
drwxr-xr-x 6 root root 4096 Dec 22 07:57 ..
-rw------- 1 stan stan  260 Dec  2 21:03 .bash_history
-rw-r--r-- 1 stan stan  220 Dec  2 20:38 .bash_logout
-rw-r--r-- 1 stan stan 3771 Dec  2 20:38 .bashrc
-rw-r--r-- 1 stan stan 8980 Dec  2 20:38 examples.desktop
-rw-r--r-- 1 root root   44 Dec  2 20:56 flag8.txt
-rw-r--r-- 1 stan stan  807 Dec  2 20:38 .profile
cartman@ubuntu:/home$ cd stan
-bash: cd: stan: Permission denied
cartman@ubuntu:/home$ sudo cd stan
sudo: cd: command not found
cartman@ubuntu:/home$ cat /stan/flag8.txt
cat: /stan/flag8.txt: No such file or directory
cartman@ubuntu:/home$ sudo cp stan/flag8.txt ./flag8.txt
[sudo] password for cartman:
cartman@ubuntu:/home$ ;s
-bash: syntax error near unexpected token `;'
cartman@ubuntu:/home$ ls
cartman  chef  cyberviking  flag8.txt  kenny  stan
cartman@ubuntu:/home$ cat flag
cat: flag: No such file or directory
cartman@ubuntu:/home$ cat flag8.txt
"Dude, this is pretty fucked up right here"
cartman@ubuntu:/home$
```

*Here we see **Flag 8**revealed to be "Dude, this is pretty fucked up right here".*

Finally, having learned our lesson from our earlier delay in seeking information on what we suspected was an issue, we once again contact the supplier of the VM to determine if the restrictions on the file Jxvxdj.png preventing us from accessing it were a mistake. He confirmed it was a mistake. This file was then issued directly and upon viewing it, it was found to be an image containing the password for the 'stan' account in plain text on the image.

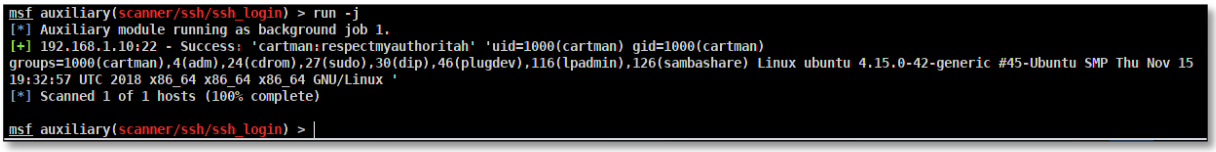


Here we see the previously inaccessible jxvxdj.png image with the 'stan' account appearing in plain text within the image.

18: Persistence, Going the Extra Mile

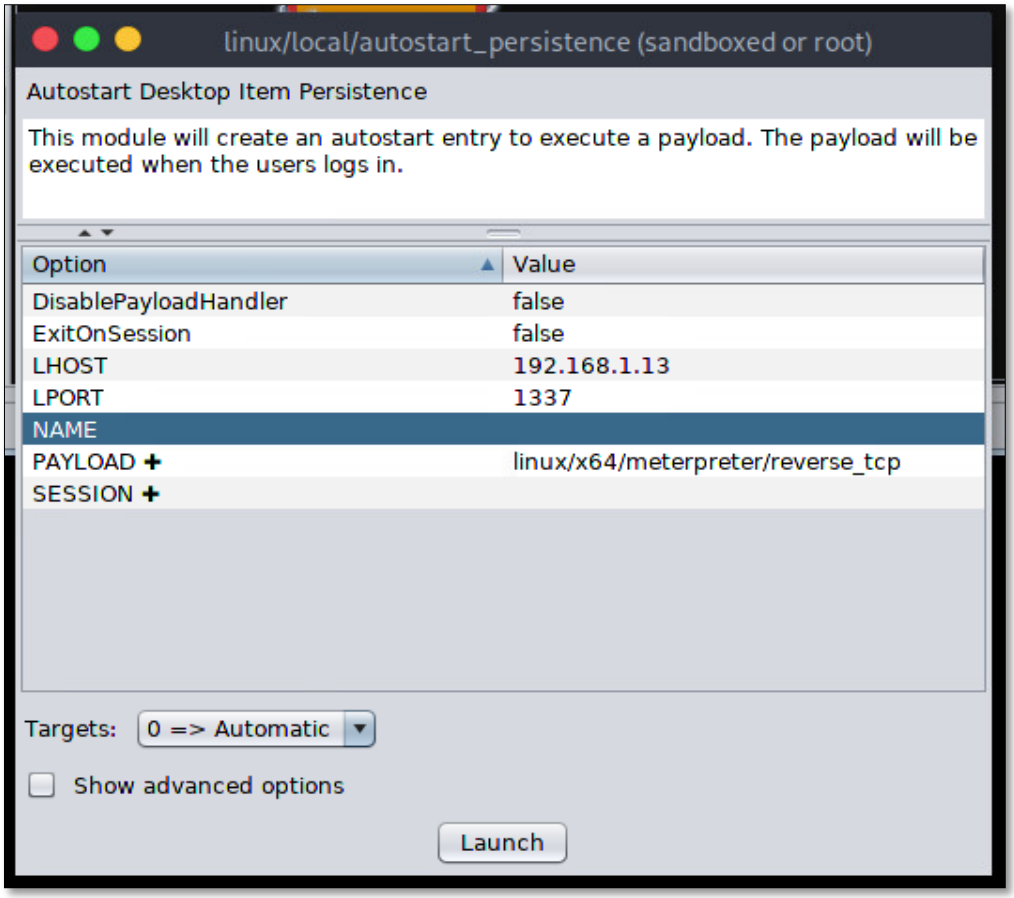
During the final section of the penetration test after completing all the flags we felt it was important to look at what would be next in a real-world penetration test, post exploitation.

Since we had access to a root account ‘cartman’ we wanted to maintain persistence, logging into the SSH trough the Metasploit framework using the auxiliary scanner/ssh/ssh_login, this gives us a SSH session through Metasploit, sending the session to background.



Logging into ssh via Metasploit.

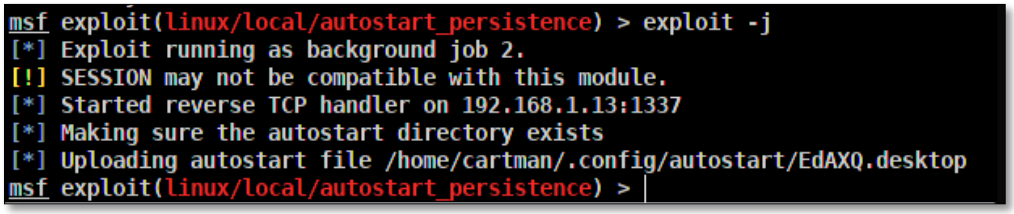
Using the post exploit script ‘linux/local/autostart_peristence’, this would allow us to add a script that would use the payload ‘linux/x64/meterpreter/reverse_tcp’ to spawn a reverse connection to my system through a specified port.



Here we prepare our first exploit.

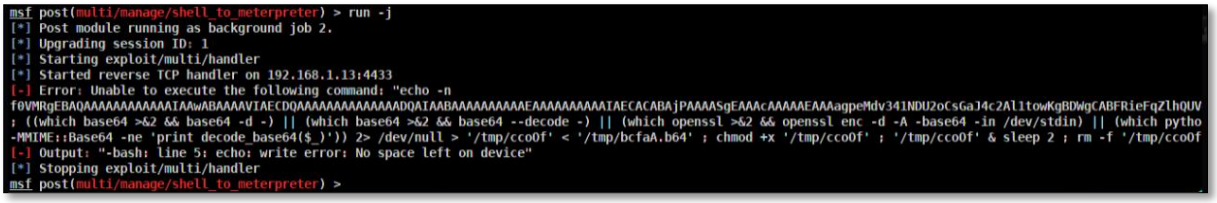
The image was show in Armitage to show the script with the exploitation, but the actually exploitation was done via command line Metasploit.

Bellow we can see the auto-start file with the payload being uploaded to the 'cartman' account



Preparing to deploy auto-start.

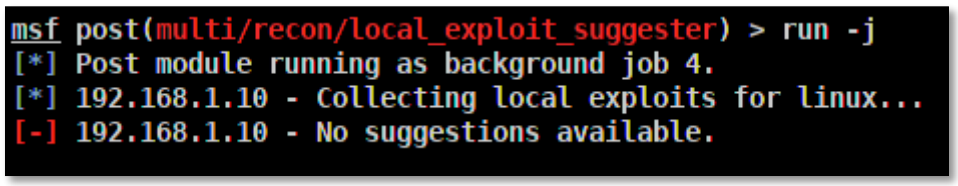
After this we felt it was important to spawn a meterpreter shell as there is a few more commands that are available through meterpreter with a single command instead of searching for the script. Upon attempting this it came up with an error on the VM due to lack of space.



Attempting to convert shell to meterpreter.

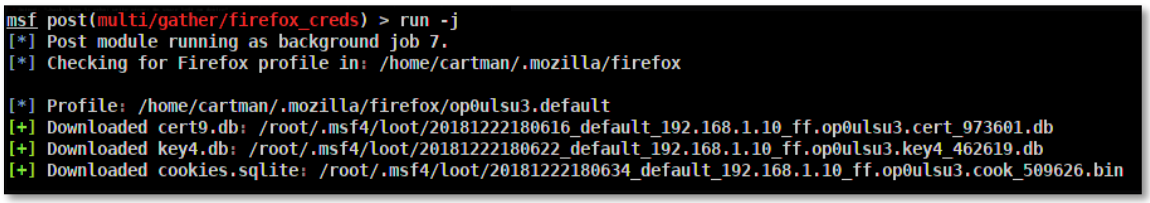
With this failure we decided to just continue with the scripts instead of jumping through meterpreter.

The first command to search was for possible local exploits that may allow for kernel-based exploitation if we lose our persistence access to the admin.



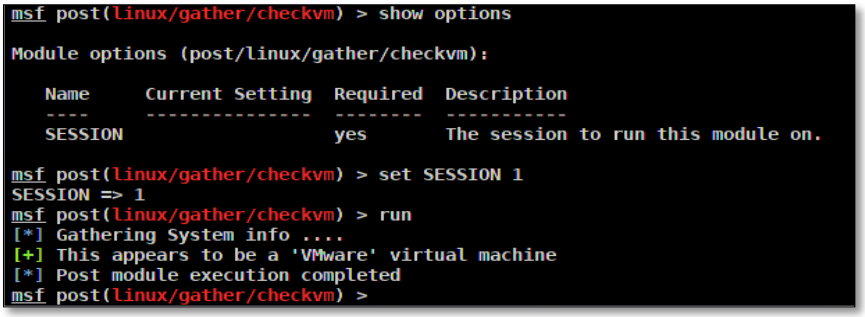
No suggestions returned.

The post exploitation script was unable to suggest any exploits to be used. The next script to run as to check for Firefox credentials, it may be possible that there might be data saved in the Firefox browser such as a password. For this we used the post script multi/gather/firefox_creds.



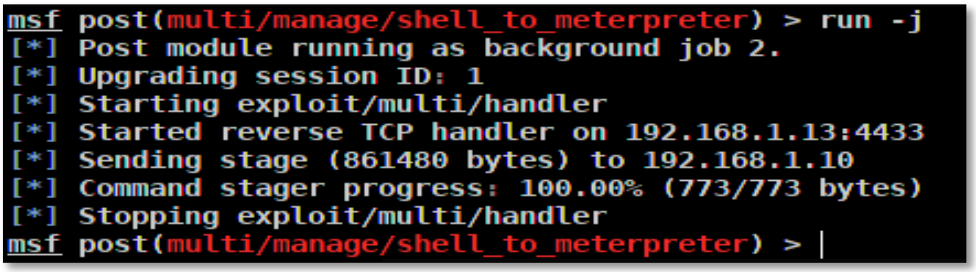
No credentials discovered.

This unfortunately gave out no credentials stored. We were able to confirm that we were indeed working on a virtual machine, in this situation it isn't useful but in a real-world test could indicate we are in a possible sandbox, honeypot or virtual system, this could also give us the option to look at exploiting from the virtual machines to the host, using one of many forums of VM escapes that are currently available.



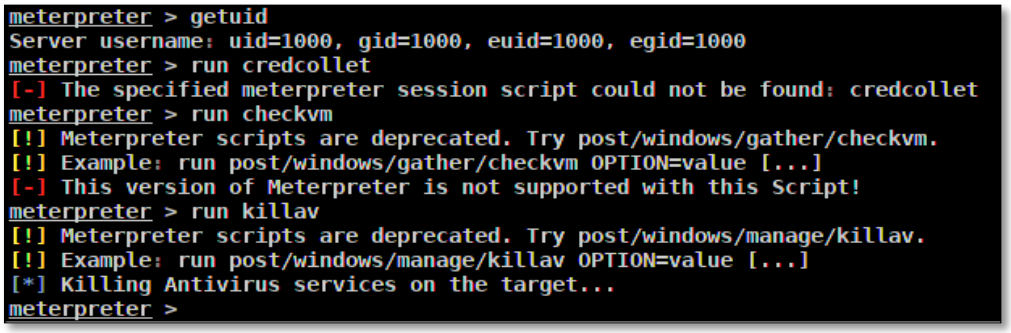
Displaying options for checkvm.

We decided to restart the system remotely to see if everything was still working or if there was a caching issue take up the storage. When we reset the system and attempted the shell_to_meterpreter script again it worked this time.



Deploying our shell conversion exploit.

Having a meterpreter session allowed us to deploy scripts faster and easier.



Success. We have converted our ssh access to a meterpreter shell.

As you can see above we threw some basic scripts at the system to see outcomes using meterpreter to handle the scripts.

19: Conclusions

The penetration test successfully concluded, with all flags recovered including the bonus flag,along-sidesome post exploitation work; our group took time to discuss the difficulty of the penetration test. With our groups average level of knowledge, skills, experience and determination, we concluded that the box was of moderate to high difficulty.

Had we not encountered the bugs we did encounter, we estimate approximately one week could have been saved in time. However, as mentioned previously, we cannot lay the blame for that extra weeks' time on anybody but ourselves, as we failed to come forward soon enough. We learned from that experience and did not repeat the mistake.

The bugs, effectively acted as extra layers of security in point of fact, and that is good from the perspective of securing the machine.

We would recommend in future that no users, but the admin; be allowed SSH access. Passwords should be restricted to no less than 13 digits using upper case, lower case, numbers and where possible special characters, and 'leet**(viii)** speak' alterations to words should be used; to preclude the use of most, if not all wordlists and/or dictionary attacks. Not leaving the password in plain text on a picture is also recommended. That being said, we feel that only a really determined attacker would gain access to this server without all but the best in 0day exploits, a deep wealth of knowledge and skill and access to some seriously high-end computational mining rigs.

OurTeam had access to both moderately powerful local machines, and high end remote mining machines at our disposal and at least a moderate level of skill and a high level of dedication and determination. We did of course lack access to any viable 0day exploitation tools but as it turned out we only used exploitation scripts in post exploitation. Not to mention, that one of our group has a reasonably high level of both skill and knowledge giving us, perhaps; an advantage over other groups. Another group member dedicated almost all his time for the better part of two weeks to this test.

All things considered the server can be considered reasonably well secured. No server will ever be 100% hack proof, but this server from the perspective of our team's average moderate skill level was pretty robust, and although it did fall, it took every tool in our arsenal to achieve this. The addition of firewalls and port blocking would greatly increase security, although; It should be remembered that this was a test, and the server must have some access points.

We greatly appreciated this challenge and the valuable lessons we learned from it, not to mention the sharpening of our skillset.

We would like to extend our groups sincere thanks to our lecturer, Mark Cummins, not only for such a fantastic challenge, and his help in working our way through the bugs discovered during the test, but also for his excellent tutelage this year which made it possible for us to achieve all we achieved during this test.

20: References

(1) Python script used to run ftp server: https://github.com/giampalo/p.

21: Appendix

(1a) Here is listed the source code of our custom Java based IDE compiled and executed via the Eclipse IDE. (Testing of this code should be as simple as copying and pasting into a new project in Eclipse and running as Java application):

```
import java.io.*;


public class PassGenMain{

	public String[] f = {"f5"};

	public String[] t = {"t4"};

	public String[] p = {"pg3"};

	public String[] y = {"yb3", "ybs3", "hhs3"};

	public String[] j = {"jb3"};

	public String[] pb = {"patb2","p&tb2"};

	public String[] s = {"sp4"};

	public String[] dte = {"dte2"};

	public String[] to = {"taj2"};

	public String[] w = {"wecatrr2","rrawec","rr&wec2","rr2","wc2"};

	public String[] ta = {"t1", "lt1","td1"};

	public String[] i = {"i&ss2", "iass2","ias2"};


	public String text = "";

	public PassGenMain(){

		gen();

	}

	public void gen(){

		try{

			BufferedWriter writer = new BufferedWriter(new FileWriter("list.txt", true));


			for(int x = 0; x < y.length; x++){

				for(int h = 0; h < pb.length; h++){

					for(int g = 0; g < w.length; g++){

						for(int d = 0; d < ta.length; d++){
```

```
                                                        for(int k = 0; k < i.length; k++){

                                                                text = f[0] + t[0] + p[0] + y[x] + j[0] +
pb[h] + s[0] + dte[0] + to[0] + w[g] + ta[d] + i[k];

                                                                System.out.println(text);
                                                                writer.newLine();
                                                        }
                                                }
                                        }
                                }
                        }
                        writer.close();
                }catch(Exception e){
                        e.printStackTrace();
                }
        }

        public static void main(String[] args){
                PassGenMain g = new PassGenMain();
        }
}
```

(2a) Account Passwords:

chef = gravy127

stan = manbearpig

Kenny = theykilledme

Cartman(admin) = respectmyauthoritah

Glossary:

(i) TCP = Transmission control Protocol

(ii) UDP = User Datagram Protocol

(iii) VM = Virtual Machine.

(iv) IP = Internet Protocol Address (Personalized individual address within a network and global network address external of a network).

(v) FTP = File Transfer protocol

(vi) VSFTPD = Very Secure File Transfer Protocol Daemon (Program used to set up a secure FTP server).

(vii) SSH = Secure Shell

(viii) Leet Speak = Elite Speak (The substitution of numbers in place of letters to give the closest visual representation of the word in plain text. E.g.: 3lit3, ElitE. Generally used in computer blogs and in gaming circles, but also used to further obfuscate passwords in order to greatly increase the difficult in crack said password).