Mossad Challenge

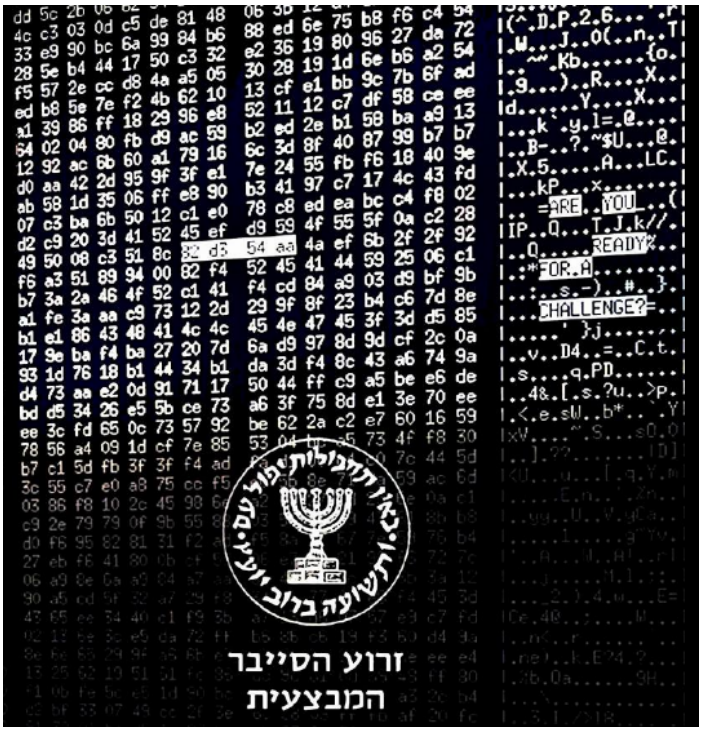
Solvers: **Roman Zaikin (Checkpoint), Asaf Katz(Bugsec).**

In our opinion this challenge was very interesting, we manage to solve it within 3-5 hours. And we are glad to share with you our way of though and the scripts we wrote.

We hope you will enjoy reading and maybe will learn something new ☺

The paper

The challenge begin with a picture at the local newspaper with the following image:



As you can see it’s an IP address in form of Hex, you need to convert it to decimal.

We did it using the following script:

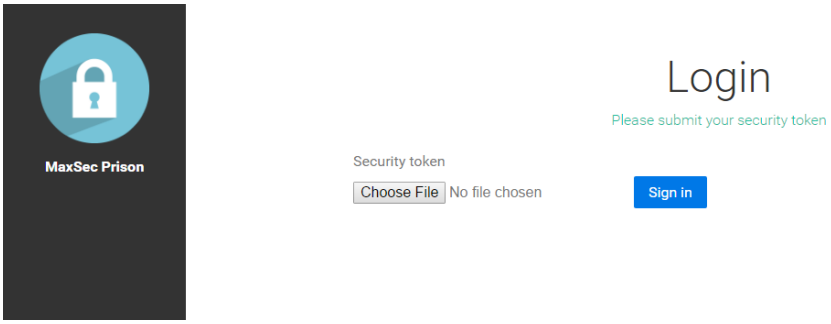
```
data = [str(int("82",16)),str(int("d3",16)),str(int("54",16)),str(int("aa",16))]
print "http://" + ".".join(data)
```

Another good way to do this would be using this script:

```
import socket
ip = socket.inet_ntoa('82d354aa'.decode('hex'))
url = "http://%s" % ip
print url
```

Challenge 1 – the login

As you can see you are an agent and you need to spoof the finger print to connect to this web panel:



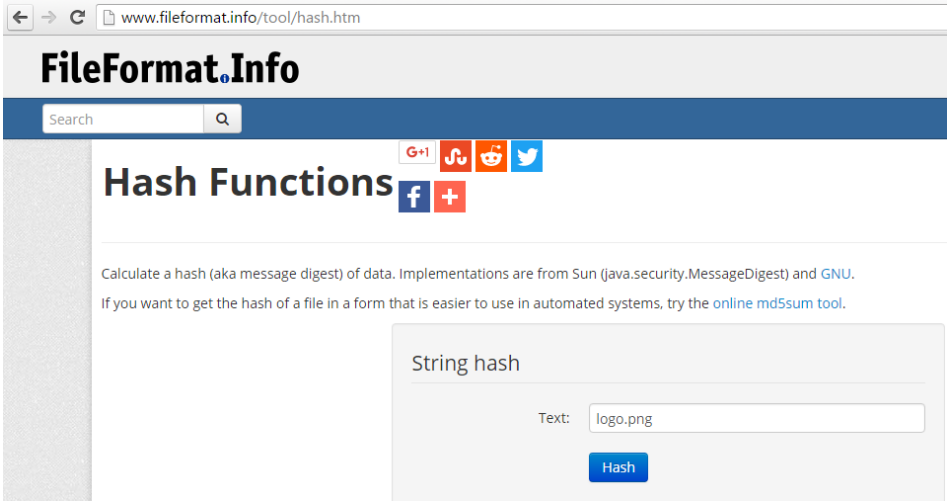
If you will see closely you will that there is a logic behind the picture we see at the left side of the page:

```
http://130.211.84.170/challenge1/get-image?name=logo.png&h=87d41d15f&multiple=0
```

The parameters:

name = logo.png (file name)
h = 87d41d15f (h mean hash usually)
multiple = 0 (don't know yet)

Here we become lazy and want to know what hash is been using so we google for the following website and try our luck:



Using the search we found our hash, this was very simple ☺

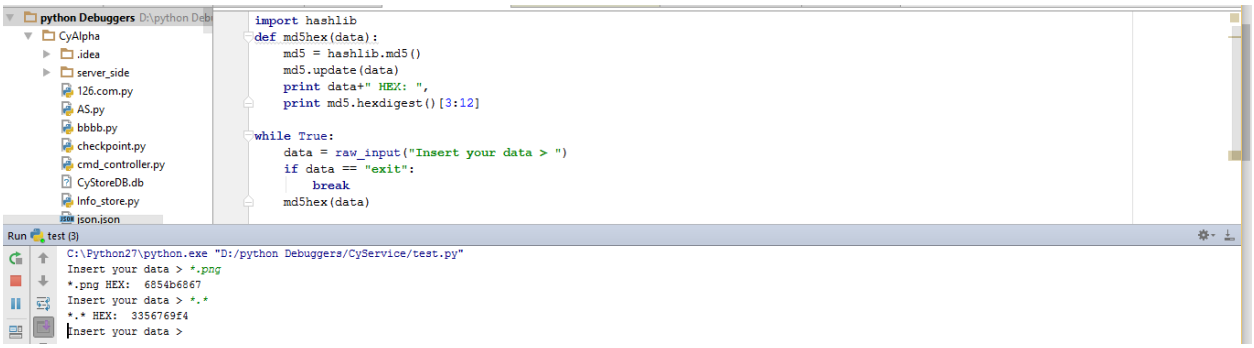
So we found that 87d41d15f … Is MD5 characters [3:12] so we crafted a very simple script to assist with the hashes ☺

```
import hashlib
def md5hex(data):
    md5 = hashlib.md5()
    md5.update(data)
    print data+" HEX: ",
    print md5.hexdigest()[3:12]

while True:
    data = raw_input("Insert your data > ")
    if data == "exit":
        break
    md5hex(data)
```

And a more complete way to achieve this without working too much:

```
import md5, urllib2, sys

word = sys.argv[1]

md5_hash = md5.new(word).hexdigest()[3:12]
url = 'http://130.211.84.170/challenge1/get-
image?name=%s&h=%s&multiple=1' % (word, md5_hash)
print url
while True:
    try:
        response = urllib2.urlopen(url).read()
    except Exception as e:
        if e.getcode() in [404,400,403]:
            print "Nope"
        exit()

print response
```

Hmm… let’s try *.png after we saw the result we tried *.* to get all the files out there.

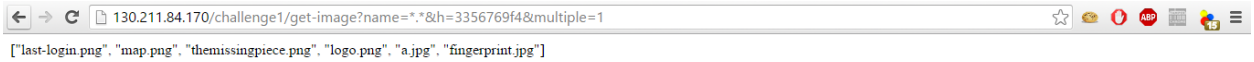


We found something interesting

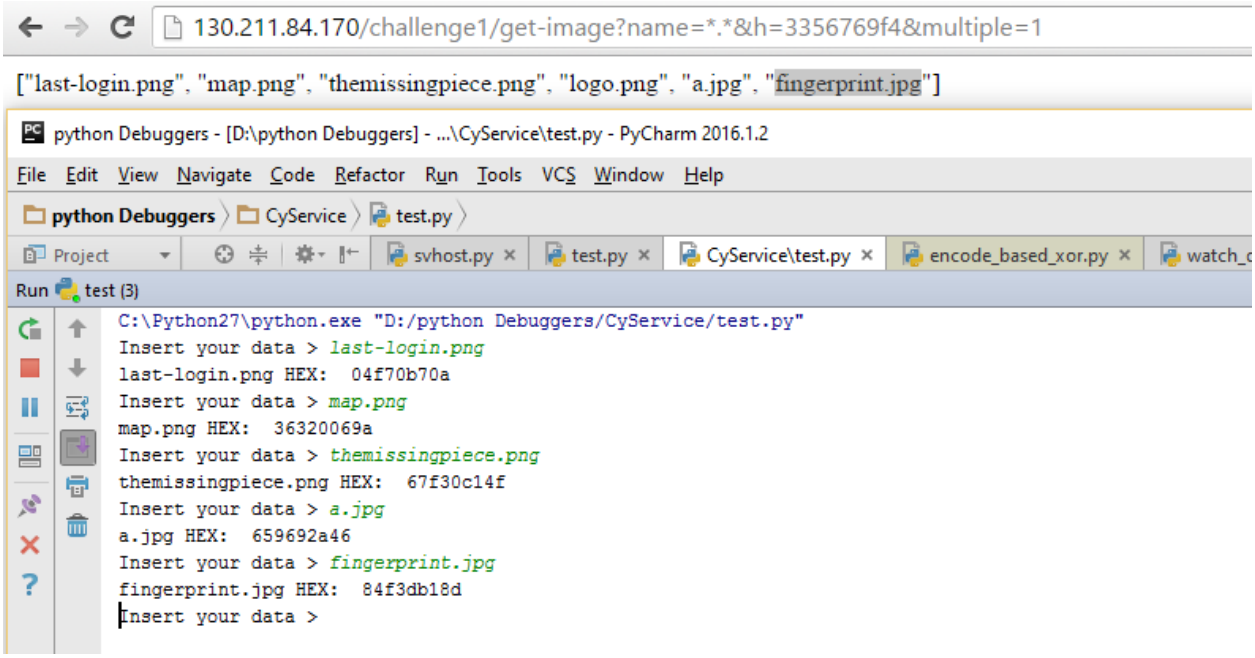
So let’s recon our findings:

We know what **name** parameter does and we know what **h** does.
So what **multiple** does?

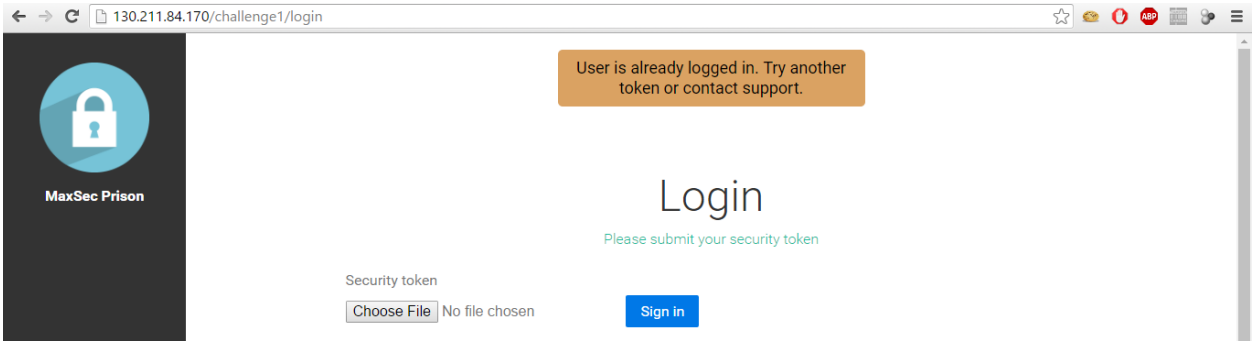
Let’s play with this



Lol, so that is what multiple does…
Let’s download all the images and try to connect… even the jpg (just for fun)



And we will get the following output:



Damn… so we need to get the crc of this image and try to find our picture that will contains the same crc, this should not be that hard…

Let’s get the crc of this png.

```
import crc16
import sys
print hex(crc16.crc16xmodem(open(sys.argv[1],"r").read()))
```

```
root@kaliRz:~# ./xmodem.py last-login.png
0x398e
```

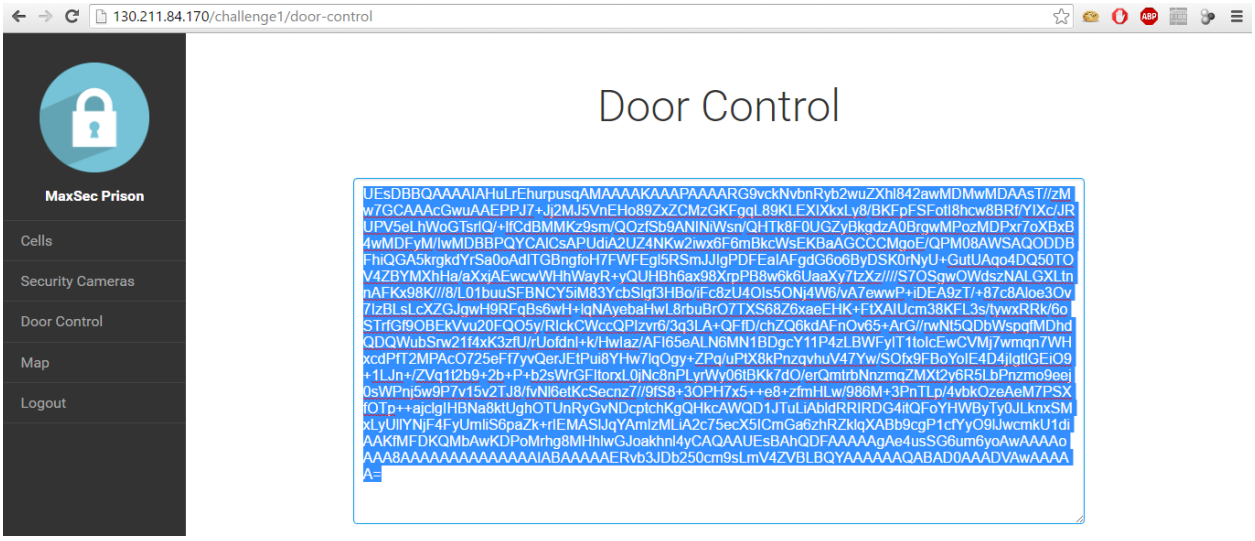
And here is our simple bruteforce script:

```
import crc16
import struct

data = 0
correct_crc = 0x398e
magic = bytearray("\x89\x50\x4e\x47\x0d\x0a\x1a\x0a")
payload = magic + data

while correct_crc != crc16.crc16xmodem(bytes(payload)):
    data += 1
    # will fail if longer then 0xffff, but who cares lol
    output = struct.pack('>H', data)
    payload = bytearray(magic + output)

file("key.png", "wb").write(payload)
```

Another way to do this would be a simple wild brute force:

```
from crc16 import crc16xmodem
from random import randint

with open('last-login.png','rb+') as fp:
   data = fp.read()

crc = crc16xmodem(data)
new_crc=0
while (crc!=new_crc):
    # Bruteforce until we get a matching new_crc
    if ord(data[-1])==0xff:
        data = data[:-1]+chr(randint(0,255))
        data=data+chr(0)
    if ord(data[-1])!=0x00:
        data = data[:-1]+chr(ord(data[-1])+1)
    new_crc = crc16xmodem(data)
    if ord(data[-1])==0x00:
    data = data[:-1]+chr(ord(data[-1])+1)

with open('new.png','wb') as fp:
   fp.write(data)
```

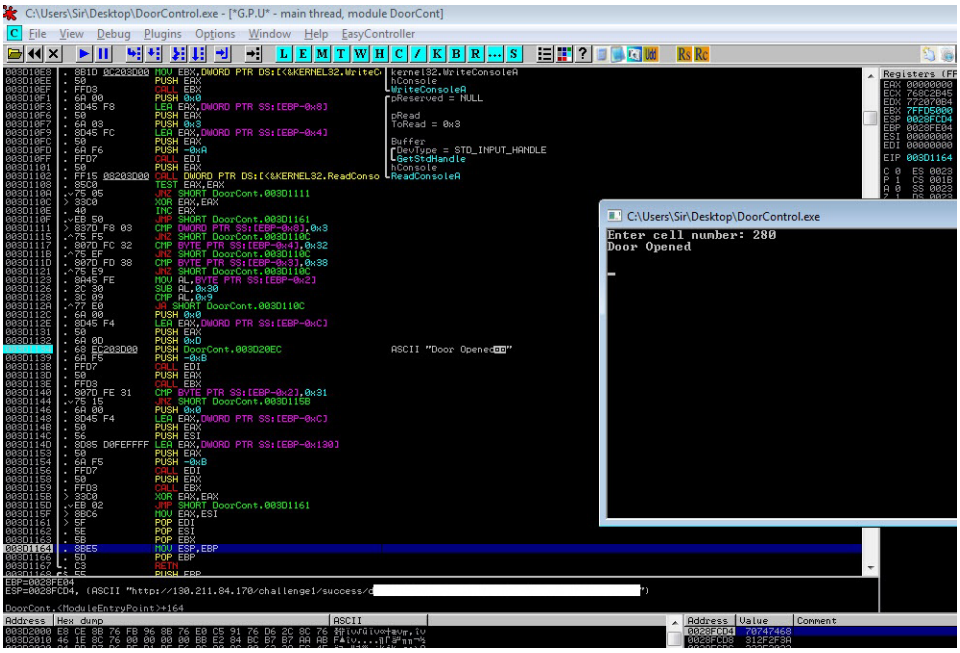
Cool we logged in let’s see what is out there:



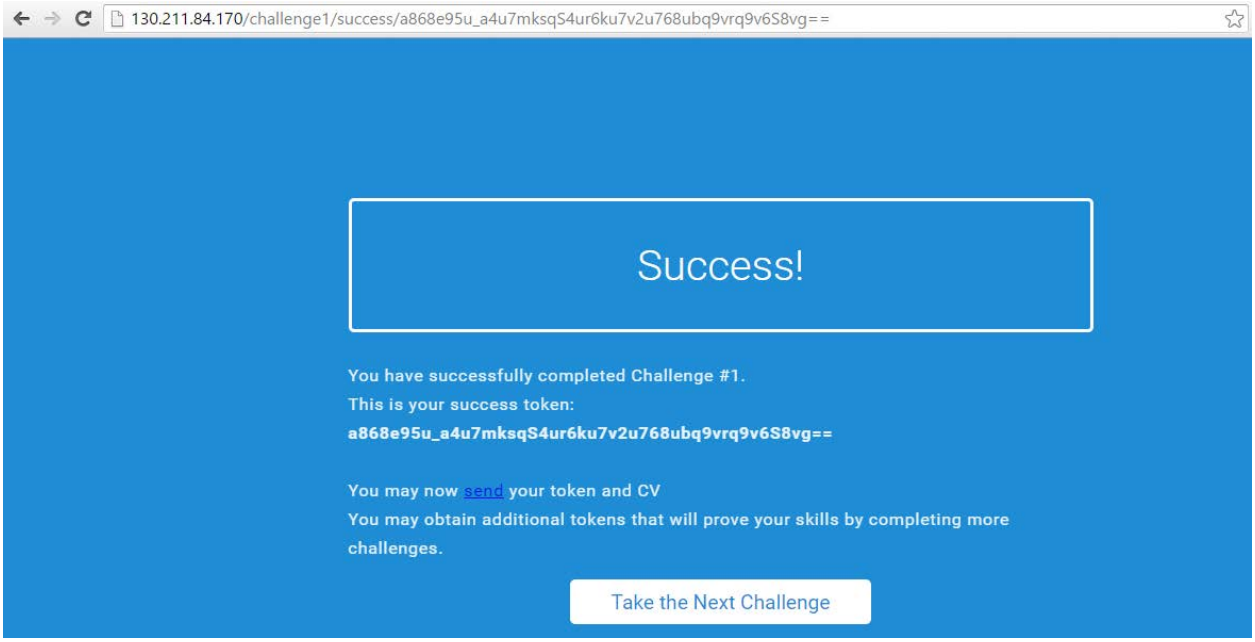

```
with open("a.zip", "w") as f:

f.write("UEsDBBQAAAAIAHuLrEhurpusqAMAAAAKAAAPAAAARG9vckNvbnRyb2wuZXhl842aw
MDMwMDAAsT//zMw7GCAAAcGwuAAEPPJ7+Jj2MJ5VnEHo89ZxZCMzGKFgqL89KLEXIXkxLy8/BK
FpFSFotI8hcw8BRf/YIXc/JRUPV5eLhWoGTsrlQ/+lfCdBMMKz9sm/QOzfSb9ANINiWsn/QHTk
8F0UGZyBkgdzA0BrgwMPozMDPxr7oXBxB4wMDFyM/IwMDBBPQYCAlCsAPUdiA2UZ4NKw2iwx6F
6mBkcWsEKBaAGCCCMgoE/QPM08AWSAQODDBFhiQGA5krgkdYrSa0oAdITGBngfoH7FWFEgl5RS
mJJIgPDFEaIAFgdG6o6ByDSK0rNyU+GutUAqo4DQ50TOV4ZBYMXhHa/aXxjAEwcwWHhWayR+yQ
UHBh6ax98XrpPB8w6k6UaaXy7tzXz////S7OSgwOWdszNALGXLtnnAFKx98K///8/L01buuSFB
NCY5iM83YcbSlgf3HBo/iFc8zU4OIs5ONj4W6/vA7ewwP+iDEA9zT/+87c8Aloe3Ov7IzBLsLc
XZGJgwH9RFqBs6wH+lqNAyebaHwL8rbuBrO7TXS68Z6xaeEHK+FtXAIUcm38KFL3s/tywxRRk/
6oSTrfGf9OBEkVvu20FQO5y/RIckCWccQPIzvr6/3q3LA+QFfD/chZQ6kdAFnOv65+ArG//rwN
t5QDbWspqfMDhdQDQWubSrw21f4xK3zfU/rUofdnl+k/HwIaz/AFI65eALN6MN1BDgcY11P4zL
BWFyIT1toIcEwCVMj7wmqn7WHxcdPfT2MPAcO725eFf7yvQerJEtPui8YHw7lqOgy+ZPq/uPtX
8kPnzqvhuV47Yw/SOfx9FBoYoIE4D4jIgtlGEiO9+1LJn+/ZVq1t2b9+2b+P+b2sWrGFItorxL
0jNc8nPLyrWy06tBKk7dO/erQmtrbNnzmqZMXt2y6R5LbPnzmo9eej0sWPnj5w9P7v15v2TJ8/
fvNl6etKcSecnz7//9fS8+3OPH7x5++e8+zfmHLw/986M+3PnTLp/4vbkOzeAeM7PSXfOTp++a
jclgIHBNa8ktUghOTUnRyGvNDcptchKgQHkcAWQD1JTuLiAbldRRIRDG4itQFoYHWByTy0JLkn
xSMxLyUllYNjF4FyUmliS6paZk+rIEMASlJqYAmIzMLiA2c75ecX5ICmGa6zhRZklqXABb9cgP
1cfYyO9lJwcmkU1diAAKfMFDKQMbAwKDPoMrhg8MHhlwGJoakhnl4yCAQAAUEsBAhQDFAAAAAg
Ae4usSG6um6yoAwAAAAoAAA8AAAAAAAAAAAAAAIABAAAAAERvb3JDb250cm9sLmV4ZVBLBQYAA
AAAQABAD0AAADVAwAAAAA=".decode("base64"))
```

We found some executable, very simple one.

1. create config file in **c:\doors\config.txt**
2. put the content **c:\OpenDoors.key**

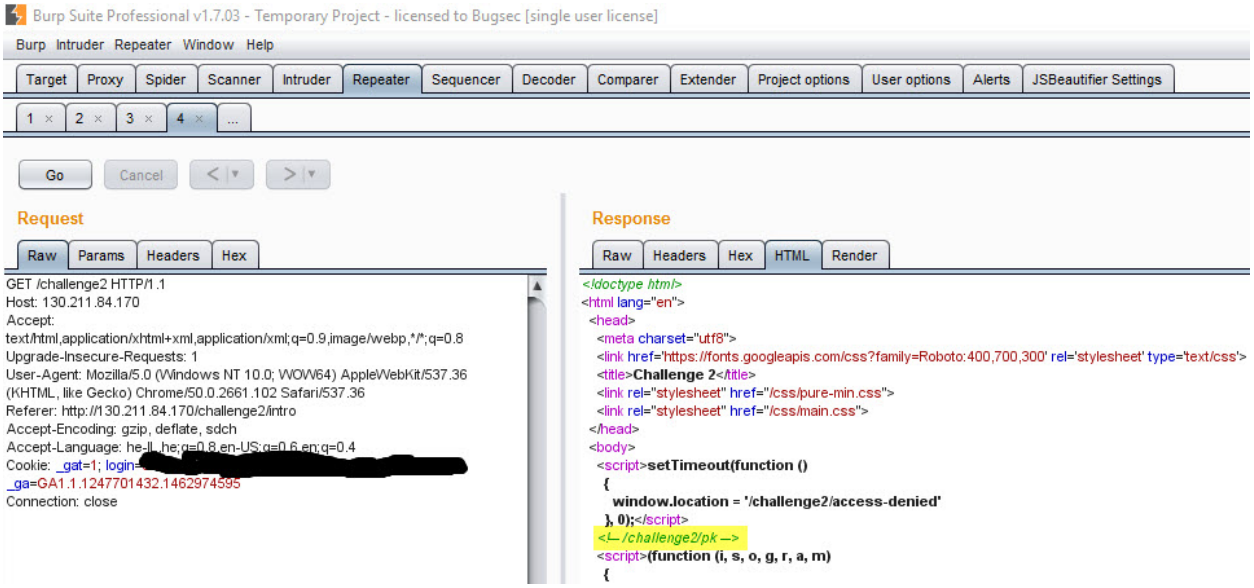


And the output:



Challenge 2 – access-denied

After clicking on start challenge in challenge 2, we were instantly redirected to access-denied. Something must be happening that we're not seeing… Burp comes to the rescue:



Accessing /challenge2/pk downloads us a zip file. But the zip file only contains a file "Almost there". Isn't this a bit weird that the zip is about 1MB and "Almost there" is only 27 bytes?

And here is the file:

```
root@kaliRz:~# binwalk x.zip

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, compressed size: 69,  uncompressed size: 75, name: "port_knocking.cfg"
116           0x74            Zip archive data, at least v2.0 to extract, compressed size: 1048896,  uncompressed size: 1048576, name: "random"
1049163       0x10024B        End of Zip archive
1049185       0x100261        Zip archive data, at least v2.0 to extract, compressed size: 29,  uncompressed size: 27, name: "Almost there"
1049314       0x1002E2        End of Zip archive
```

lets try to knock ☺
Apt-get install knockd

```
[Port Knocking]
timeout: 1000ms
knock_ports: 6460,6398,6568
dest_port: 1337
```

Knock knock knock ☺

```
[Port Knocking]
timeout: 1000ms
knock_ports: 6460,6398,6568
dest_port: 1337root@kaliRz:~/roman/_x.zip.extracted# knock 130.211.84.170 6460 6398 6568 && nmap -sV -p1337 130.211.84.170

Starting Nmap 6.49BETA5 ( https://nmap.org ) at 2016-05-12 14:41 EDT
Nmap scan report for 170.84.211.130.bc.googleusercontent.com (130.211.84.170)
Host is up (0.026s latency).
PORT     STATE SERVICE VERSION
1337/tcp open  telnet  SMC SMC2870W Wireless Ethernet Bridge
Service Info: Device: bridge

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 5.70 seconds
root@kaliRz:~/roman/_x.zip.extracted# knock 130.211.84.170 6460 6398 6568 && telnet 130.211.84.170
Trying 130.211.84.170...
^C
root@kaliRz:~/roman/_x.zip.extracted# knock 130.211.84.170 6460 6398 6568 && telnet 130.211.84.170 1337
Trying 130.211.84.170...
Connected to 130.211.84.170.
Escape character is '^]'.

Welcome, agent
>
```

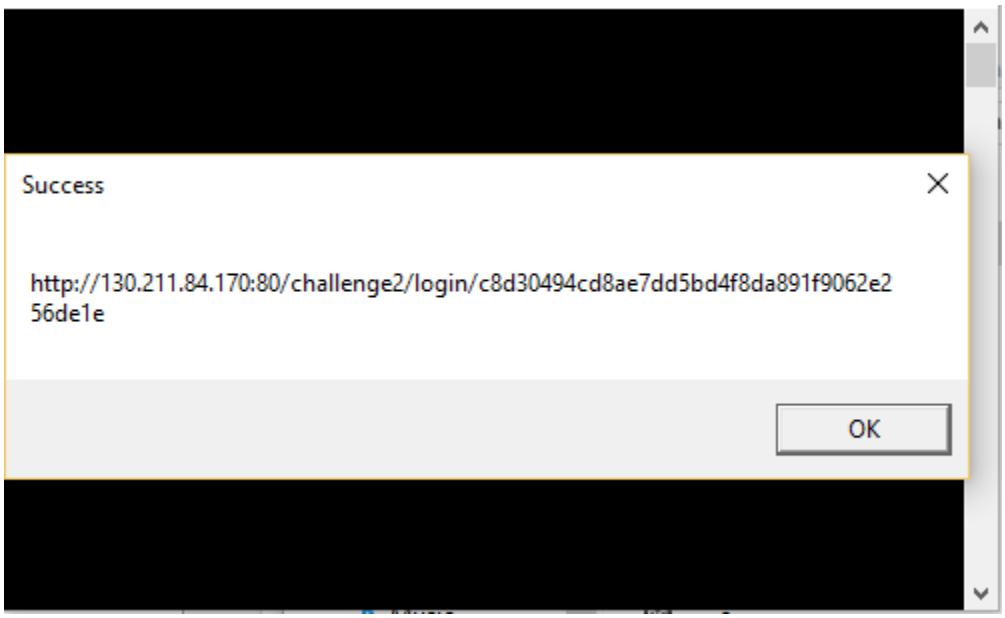
We can’t read the files with cat but we can hdump… really easy.

```
> ls
dontreadme.doc  readme.doc  passwords.txt
> hdump dontreadme.doc
746f6c6420796f75206e6f7420746f2072656164206d652e2e2e203a28
> hdump readme.doc
7961692120796f752072656164206d6521
> hdump passwords.txt
6a73643826363273063830613233 2a736a
```

The files passwords.txt contains:
jsd8&62s \x06 80a23*sj

The file **login.exe**:

```
import binascii
data =
"4d5a90000300000004000000ffff0000b800000000000000400000000000000000000000000000000000000000000000
000000000000000000000000c00000000e1fba0e00b409cd21b8014ccd21546869732070726f6772616d2063616e6e6f7
42062652072756e20696e20444f53206d6f64652e0d0d0a2400000000000000b97941c1fd182f92fd182f92fd182f9220
e7e492fe182f92fd182e92fc182f928061cf92fc182f928061f192fc182f9252696368fd182f920000000000000000504
500004c010300ffacde560000000000000000e00002010b010c0000020000000400000000000000100000001000000020
0000000040000010000000020000060000000000000006000000000000000040000000040000000000000300408500001
0000010000000001000001000000000000010000000000000000000000a82000002800000000000000000000000000000
0000000000000000000000000000300000100000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000200000080000000000000000000000000000000000000000000000
000000002e746578740000005000000000100000000200000004000000000000000000000000000000200000602e7264617
4610000f2000000002000000002000000060000000000000000000000000000000400000402e72656c6f630000100000000
30000000020000000800000000000000000000000000000040000042000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000558bec81ec9800000033c056576a2559be082040008dbd68fffffff3a566a5a45f5e80b40568ffffffc47
408403d9700000072ee6a0068a02040008d8568ffffff506a00ff150020400033c08be55dc30000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000d820000000000000acb0b0b4feebebf
5f7f4eaf6f5f5eafcf0eaf5f3f4fefcf4eba7aca5a8a8a1aaa3a1f6eba8aba3adaaeba7fca0f7f4f0fdf0a7a0fca5a1f3
a0a0f1a6a0f0a2fca0a5fcfdf5a2fdf4f2f6a1f6f1f2a0a1f5a1c4bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb000053
75636365737300d02000000000000000000000e620000000200000000000000000000000000000000000000000000000d82
000000000000046024d657373616765426f7841005553455233322e646c6c000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000100000100000001130373046300000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000"

with open('login.exe', 'wb') as out:
        out.write(binascii.unhexlify(data))
```



http://130.211.84.170:80/challenge2/login/c8d30494cd8ae7dd5bd4f8da891f9062e256de1e

LOL!!!
This is where we are led to:

And we have this (**instructions.txt** from our telnet session):

```
**** ** ******** ***, *** *** ****!! **** **** ***** * **** **
*****. **** ***!
```

And the password we didn’t used…
The instructions didn’t really help us, and the password didn’t work as well.

What we can notice is that these are 2 images, and there is probably some kind of steganography involved.

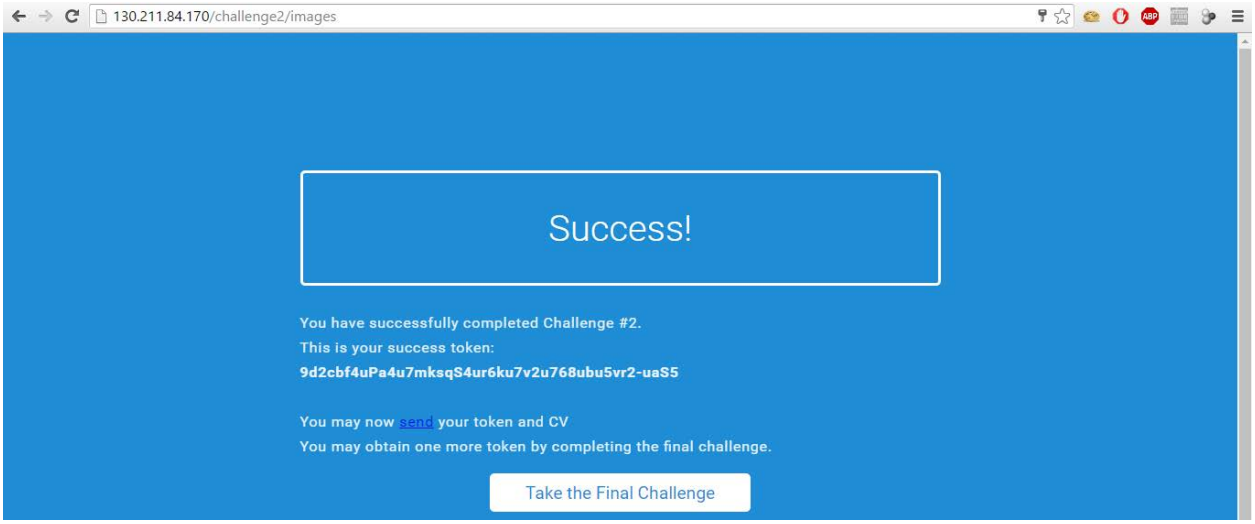
So at first we printed our all the different pixel values. After that it was easy to see that some picture's pixel values (where the pixels are different), are printable ASCII. Is this the password?

We had to tamper with this a little bit, since we don't know how to iterate through the pictures.

Maybe they put the password where you have to iterate through the height and then the width? Maybe width then height? Maybe from the top to the bottom? From right to left?

So eventually this held the right password:

```
from PIL import Image

a = Image.open('1.bmp')
b = Image.open('2.bmp')

pw=""
for i in xrange(500-1,-1,-1):
    for k in xrange(500-1,-1,-1):
        ap = a.getpixel((k,i))
        bp = b.getpixel((k,i))
        if ap!=bp:
            #print 'pixels (%d,%d)'%(i,k)
            pw+= chr(bp)
print pw
```



Challenge 3 – Port Forwarding

Let’s see what’s now
Port forwarding and download the stream to file.

This is an image with a nice game of RGB as bit array:

0, 255, 0) (0,0)

(255, 0, 0) (1,0)

(255, 0, 0) (2,0)

(255, 0, 0) (3,0)

(0, 255, 0) (4,0)

(255, 0, 0) (5,0)

(0, 255, 0) (6,0)

(0, 255, 0) (7,0)

(0, 255, 0) (8,0)

(255, 0, 0) (9,0)

(255, 0, 0) (10,0)

(0, 255, 0) (11,0)

(255, 0, 0) (12,0)

(255, 0, 0) (13,0)

(0, 255, 0) (14,0)

(255, 0, 0) (15,0)

(0, 255, 0) (16,0)

(255, 0, 0) (17,0)

(255, 0, 0) (18,0)

(255, 0, 0) (19,0)

(0, 255, 0) (20,0)

(0, 255, 0) (21,0)

(0, 255, 0) (22,0)

```python
from PIL import PngImagePlugin
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

a = PngImagePlugin.Image.open('fromserv.png')
width, height = a.size

bytestring=""

for i in xrange(height):
    for k in xrange(width):
        ab = a.getpixel((k,i))
        if ab[0] == 255 and ab[1] == 0:
                bytestring+="1"
        if ab[0] == 0 and ab[1] == 255:
                bytestring+="0"

print bytestring
with open("data","wb") as f:
        f.write(bytestring)
```

script 2 because I am lazy and did it later…

```python
#!/usr/bin/python
def bitstring_to_bytes(s):
    v = int(s, 2)
    b = bytearray()
    while v:
        b.append(v & 0xff)
        v >>= 8
    return bytes(b[::-1])

s = file("data","r").read()
file("new","wb").write(bitstring_to_bytes(s))
```

binwalk on the answer

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             POSIX tar archive (GNU), owner user name: "root", owner group name: "root"
```

Let's untar this:

tar –xvf new

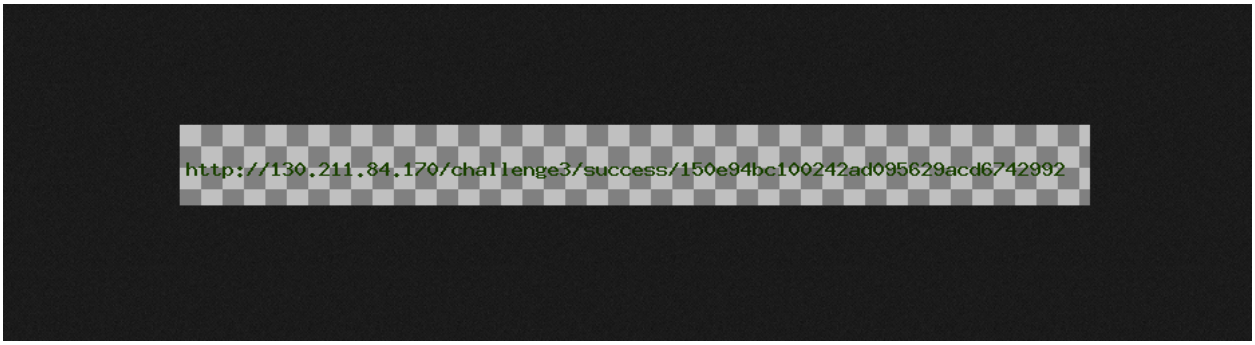
```
tmp/concat.1
tmp/concat.2
tmp/concat.3
tmp/concat.4
tmp/concat.5
```

If we concat them to new file we will see that it’s **squashfs**
apt-get install squashfs-tools
binwalk –e new

and we have this:

```
root@kaliRz:~/asaf/tmp/_new.extracted/squashfs-root# ls
file.png
```

done…



The script to get the file from the server. Both Roman & I had a really bad experience with opening port ranges on our routers (**THANK YOU BEZEQ**), I (Asaf), actually had to use **"open_port_on_router"** function which included accessing the CGI directly with the credentials in order to make it for the transmission time. Roman solved this by SSH tunneling to a network he has access to where he controls a CISCO router that isn't as retarded as Bezeq's routers.

```
import sys
import threading
import time
from socket import *

def open_port_on_router(port):
    print "CENSORED"

# to get all the data
def recv_timeout(the_socket,timeout=5):
    the_socket.setblocking(0)

    total_data=[]
    data=""

    begin=time.time()
    while True:
        if total_data and time.time() - begin > timeout:
            break
        elif time.time()-begin > timeout*2:
            break
        try:
            data = the_socket.recv(8192)
            if data:
                total_data.append(data)
                begin = time.time()
            else:
                time.sleep(0.1)
        except:
            pass
    return ''.join(total_data)

def TCP():
    server = socket(AF_INET, SOCK_STREAM)
    PORT = int(sys.argv[1])
```

```
    server_address = ('', PORT)

    server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    server.bind(server_address)

    server.listen(5)
    print "Socket Started On ",sys.argv[1]
    client, addr = server.accept()

    data = recv_timeout(client)
    with open('aa.png', 'ab+') as fp:
        fp.write(data)

    print "Done..."

    client.shutdown(SHUT_RD | SHUT_WR)
    client.close()
    server.close()

if __name__ == '__main__':
    threading.Thread(target=open_port_with_router, args=(sys.argv[1],)).start()
    TCP()
```