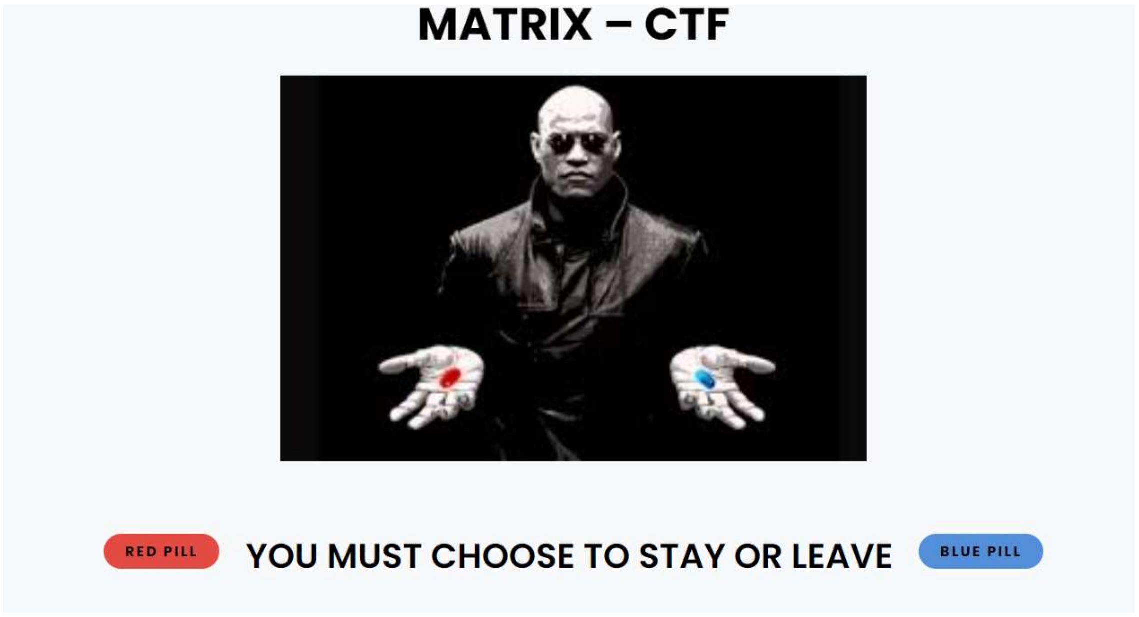
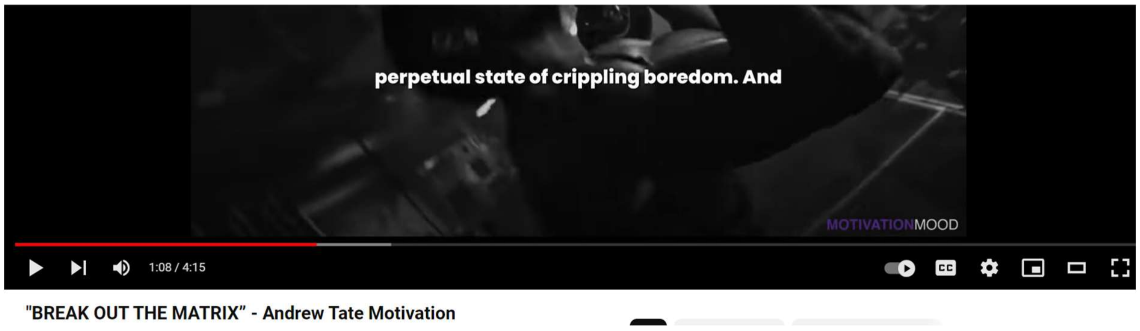
Matrix – Ctf – Write up

We start our journey in the: http://matrixctf.unaux.com:



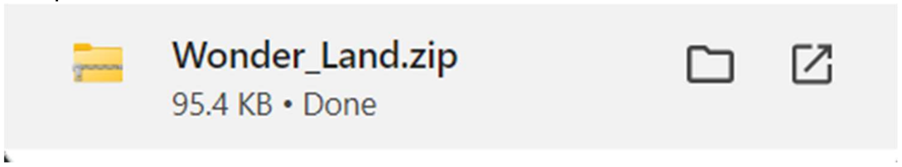
We understand from the video and the story we need to decide if we want to do the CTF – take the red pill, or wake up in our bed – taking the pill.

So, we take the blue pill. Because 2:00 am is a bit too late to start solving the CTF. Good night!


perpetual state of crippling boredom. And
MOTIVATIONMOOD
1:08 / 4:15
"BREAK OUT THE MATRIX" - Andrew Tate Motivation

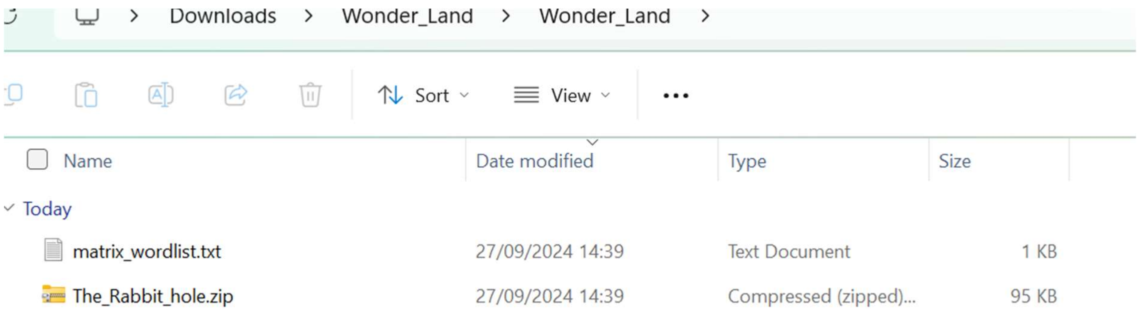
But after the amazingly motivating video of breaking out of the Matrix. I decide that sleep is for people in the matrix and if I want to grow ill need to break out of the matrix.

So I press on the red button:



Wonder_Land.zip
95.4 KB • Done

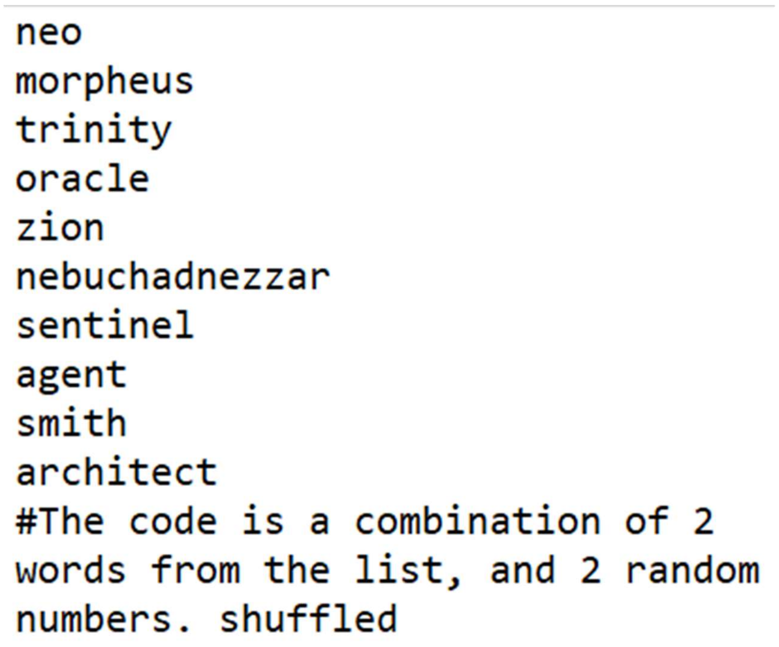
I get a zip folder, and if we unzip it:



Downloads > Wonder_Land > Wonder_Land >

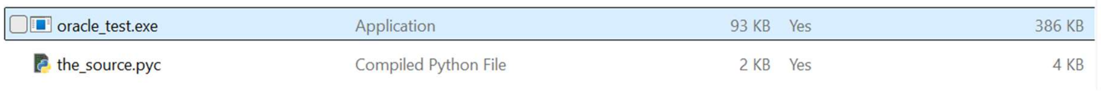
Name	Date modified	Type	Size
matrix_wordlist.txt	27/09/2024 14:39	Text Document	1 KB
The_Rabbit_hole.zip	27/09/2024 14:39	Compressed (zipped)...	95 KB

The txt file contains a couple of words, and a hint to what the password might be.



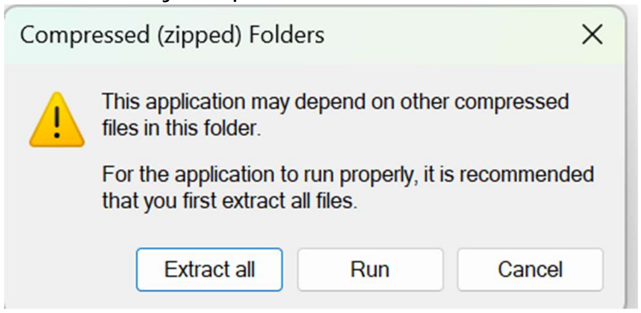
```
neo
morpheus
trinity
oracle
zion
nebuchadnezzar
sentinel
agent
smith
architect
#The code is a combination of 2
words from the list, and 2 random
numbers. shuffled
```

The zip file "the rabbit hole" has these two files:



oracle_test.exe	Application	93 KB	Yes	386 KB
the_source.pyc	Compiled Python File	2 KB	Yes	4 KB

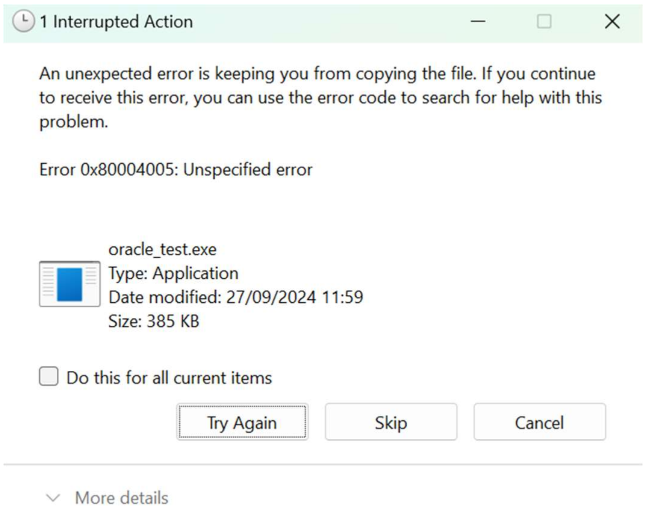
But when I try to open them:


Compressed (zipped) Folders

This application may depend on other compressed files in this folder.

For the application to run properly, it is recommended that you first extract all files.

And when extracted, we encounter this problem:


1 Interrupted Action

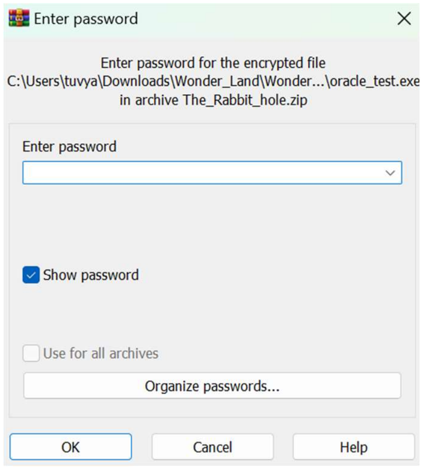
An unexpected error is keeping you from copying the file. If you continue to receive this error, you can use the error code to search for help with this problem.

Error 0x80004005: Unspecified error

oracle_test.exe
Type: Application
Date modified: 27/09/2024 11:59
Size: 385 KB

Do this for all current items

So, we try to use "Win rar"
and we discover that the file has a passcode on it:


Enter password

Enter password for the encrypted file
C:\Users\tuvya\Downloads\Wonder_Land\Wonder...\oracle_test.exe
in archive The_Rabbit_hole.zip

Enter password

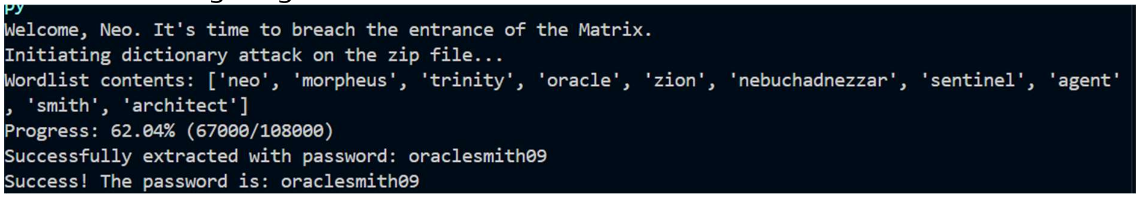
Show password ✓

Use for all archives

So, we now we understand that the txt file was talking about the password for the zip file, so we go and learn how to write a python code in relationship to zip files, and with the hint we got from the txt file:

Crack_rabbit_hole.py

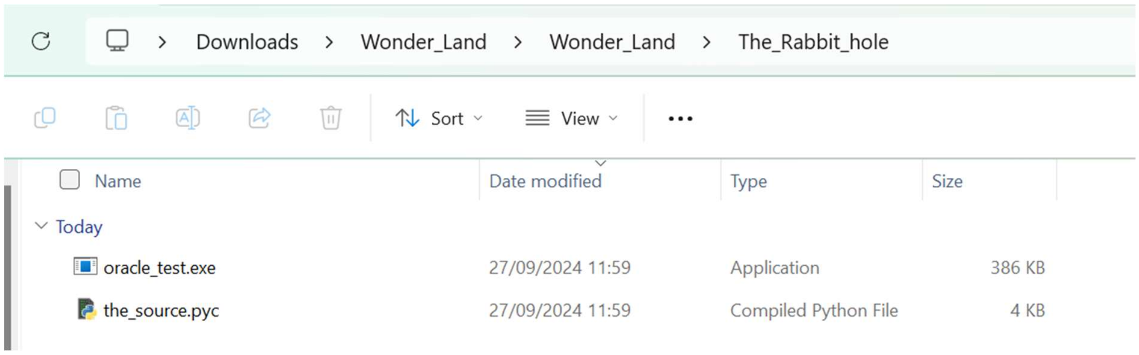
And after running we get:



```
py
Welcome, Neo. It's time to breach the entrance of the Matrix.
Initiating dictionary attack on the zip file...
Wordlist contents: ['neo', 'morpheus', 'trinity', 'oracle', 'zion', 'nebuchadnezzar', 'sentinel', 'agent'
, 'smith', 'architect']
Progress: 62.04% (67000/108000)
Successfully extracted with password: oraclesmith09
Success! The password is: oraclesmith09
```

So, the password is:
oraclesmith09

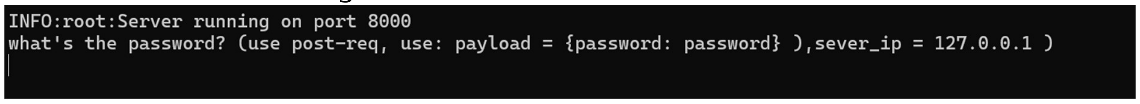
So, let's go and extract the files in the zip:



Downloads > Wonder_Land > Wonder_Land > The_Rabbit_hole

Name	Date modified	Type	Size
oracle_test.exe	27/09/2024 11:59	Application	386 KB
the_source.pyc	27/09/2024 11:59	Compiled Python File	4 KB

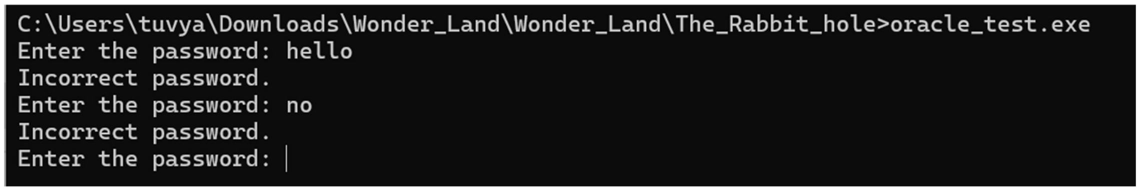
So when I run the_source I get this:



```
INFO:root:Server running on port 8000
what's the password? (use post-req, use: payload = {password: password} ),sever_ip = 127.0.0.1 )
```

ok, looks like a http server using post. But I have no idea what the password is.

Let's see the exe file:



```
C:\Users\tuvya\Downloads\Wonder_Land\Wonder_Land\The_Rabbit_hole>oracle_test.exe
Enter the password: hello
Incorrect password.
Enter the password: no
Incorrect password.
Enter the password:
```

It looks like it's asking for a password, so let's open the exe file in ida and see what we have:

Breaking the exe file:

When we open the exe file, we could see its really complicated with a lot of anti-debug functions.

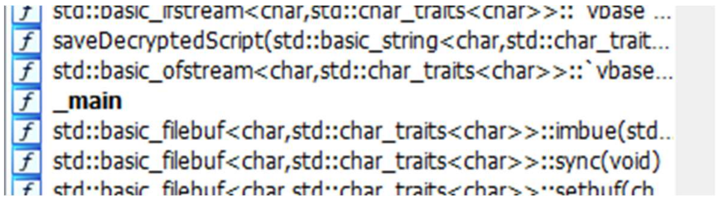
We look in the data sections, nothing the resembles a password.

So there are two ways to find the password:

1- Because the password is a string class, so you can't see it in the data section without running the code and debugging. But we can clearly see there's a lot of antibug.
2- To patch the if:

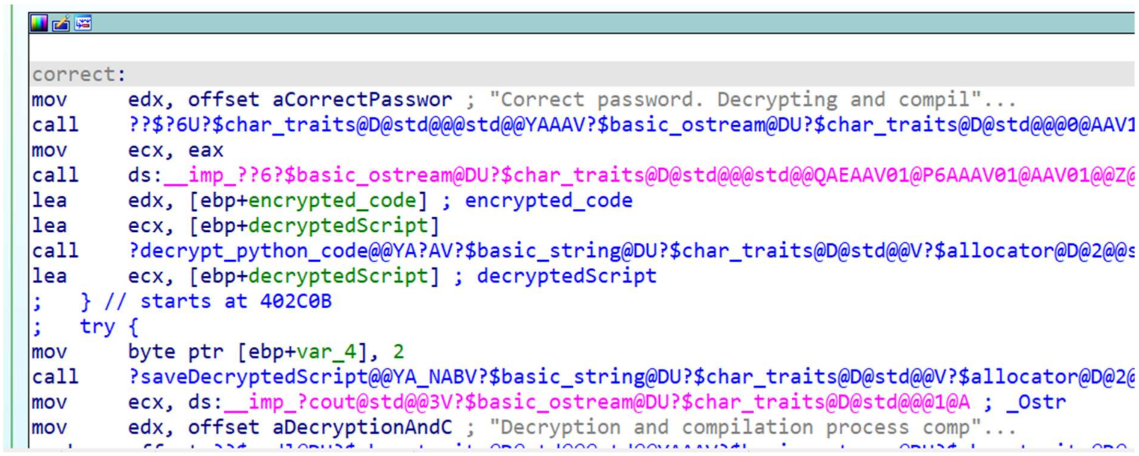
So let's look for the "If" that takes care of the password to see if its correct.

We look for the main:



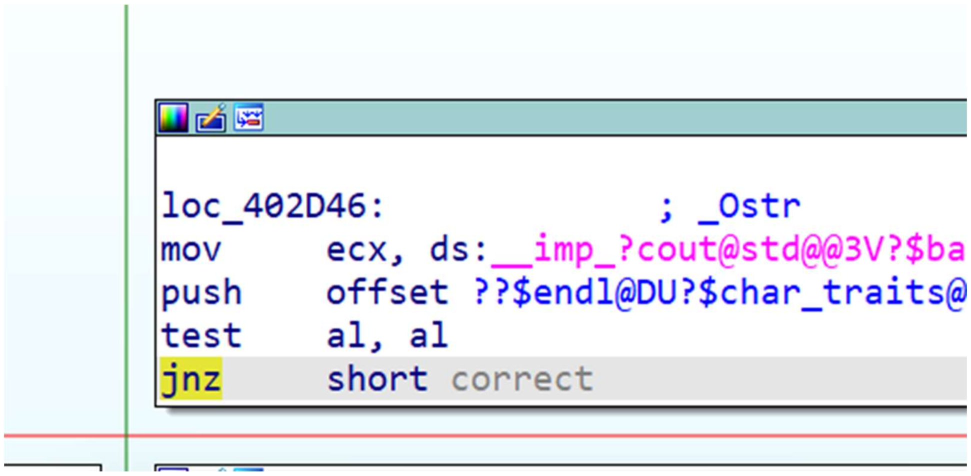
```
std::basic_ifstream<char,std::char_traits<char>>:: vbase ...
saveDecryptedScript(std::basic_string<char,std::char_trait...
std::basic_ofstream<char,std::char_traits<char>>::`vbase...
_main
std::basic_filebuf<char,std::char_traits<char>>::imbue(std..
std::basic_filebuf<char,std::char_traits<char>>::sync(void)
std::basic_filebuf<char,std::char_traits<char>>::setbuf(ch
```

We find the correct password box, and we rename it "correct":



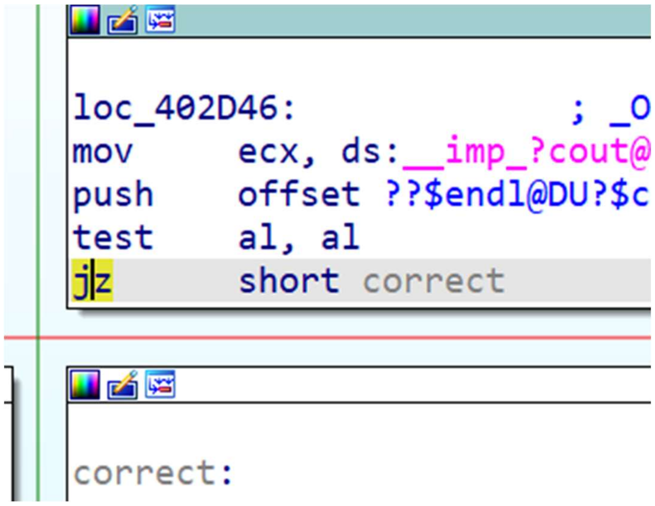
```
correct:
mov     edx, offset aCorrectPasswor ; "Correct password. Decrypting and compil"...
call    ??$?6U?$char_traits@D@std@@@std@@YAAAV?$basic_ostream@DU?$char_traits@D@std@@@0@AAV1
mov     ecx, eax
call    ds:__imp_??6?$basic_ostream@DU?$char_traits@D@std@@@std@@QAEAAV01@P6AAAV01@AAV01@@Z@
lea     edx, [ebp+encrypted_code] ; encrypted_code
lea     ecx, [ebp+decryptedScript]
call    ?decrypt_python_code@@YA?AV?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@s
lea     ecx, [ebp+decryptedScript] ; decryptedScript
;   } // starts at 402C0B
;   try {
mov     byte ptr [ebp+var_4], 2
call    ?saveDecryptedScript@@YA_NABV?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@
mov     ecx, ds:__imp_?cout@std@@3V?$basic_ostream@DU?$char_traits@D@std@@@1@A ; _Ostr
mov     edx, offset aDecryptionAndC ; "Decryption and compilation process comp"...
```

And we path the program so even when the password is wrong itll go to the the correct label:



```
loc_402D46:                 ; _Ostr
mov     ecx, ds:__imp_?cout@std@@3V?$ba
push    offset ??$endl@DU?$char_traits@
test    al, al
jnz     short correct
```

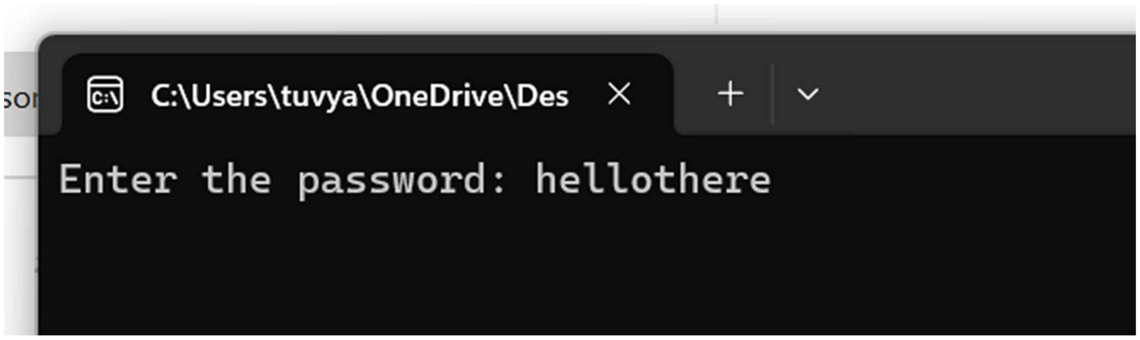
To:



```
loc_402D46:                 ; _O
mov     ecx, ds:__imp_?cout@
push    offset ??$endl@DU?$c
test    al, al
jz      short correct

correct:
```

And now when we run it:

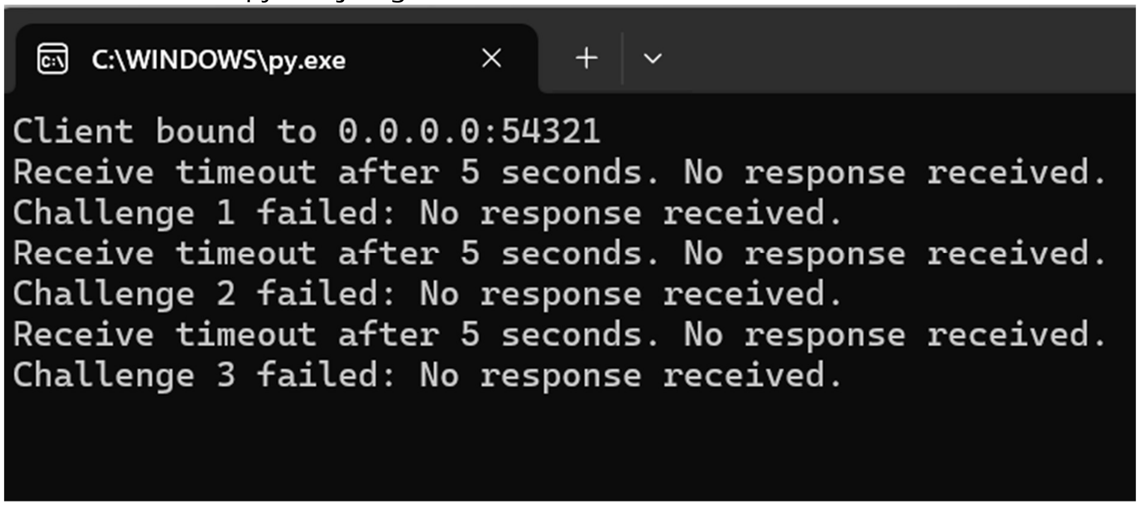


```
C:\Users\tuvya\OneDrive\Des
Enter the password: hellothere
```

the program runs and we can see in our folder we have a new file:

oracle_test.pyc

So lets run the new pyc we just got:

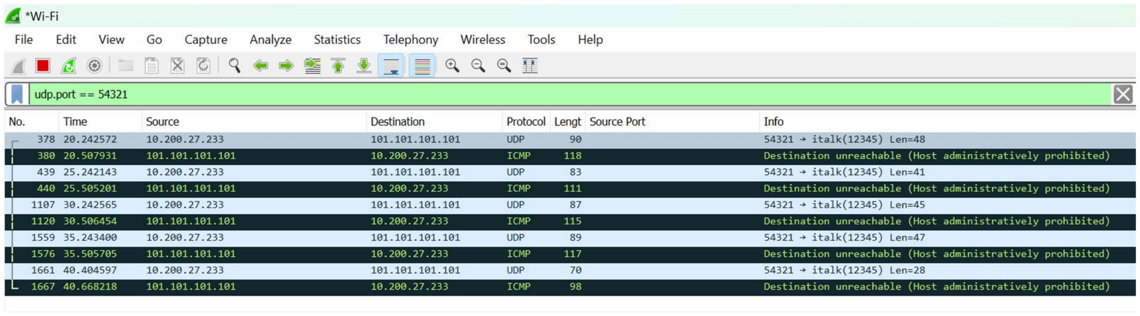


```
C:\WINDOWS\py.exe
Client bound to 0.0.0.0:54321
Receive timeout after 5 seconds. No response received.
Challenge 1 failed: No response received.
Receive timeout after 5 seconds. No response received.
Challenge 2 failed: No response received.
Receive timeout after 5 seconds. No response received.
Challenge 3 failed: No response received.
```

So we can clearly see that the code is probably a client that send something to somewhere and is waiting 5 seconds for a response.

Lets open Wire-shark:

We understand that its sending udp packets because there's no evidence of a handshake, and we guess we should use the wifi.


*Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

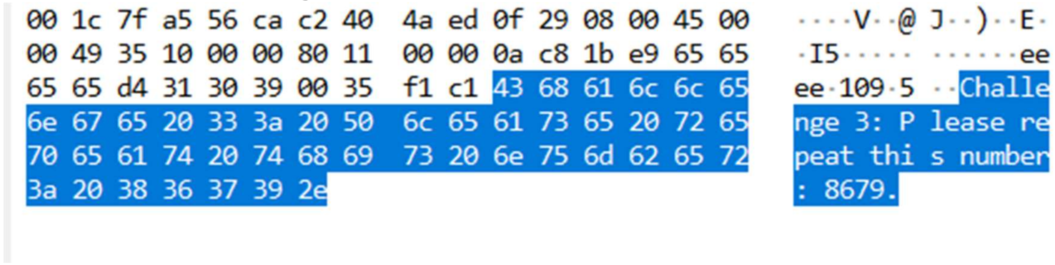
udp.port == 54321

No.	Time	Source	Destination	Protocol	Lengt	Source Port	Info
378	20.242572	10.200.27.233	101.101.101.101	UDP	90		54321 → italk(12345) Len=48
380	20.507931	101.101.101.101	10.200.27.233	ICMP	118		Destination unreachable (Host administratively prohibited)
439	25.242143	10.200.27.233	101.101.101.101	UDP	83		54321 → italk(12345) Len=41
440	25.505201	101.101.101.101	10.200.27.233	ICMP	111		Destination unreachable (Host administratively prohibited)
1107	30.242565	10.200.27.233	101.101.101.101	UDP	87		54321 → italk(12345) Len=45
1120	30.506454	101.101.101.101	10.200.27.233	ICMP	115		Destination unreachable (Host administratively prohibited)
1559	35.243400	10.200.27.233	101.101.101.101	UDP	89		54321 → italk(12345) Len=47
1576	35.505705	101.101.101.101	10.200.27.233	ICMP	117		Destination unreachable (Host administratively prohibited)
1661	40.404597	10.200.27.233	101.101.101.101	UDP	70		54321 → italk(12345) Len=28
1667	40.668218	101.101.101.101	10.200.27.233	ICMP	98		Destination unreachable (Host administratively prohibited)

We see that the client is sending 5 challenges to the ip : 101.101.101.101, and port: 12345 , and waits for an answer.

So, we'll write a server to answer the challenges.

(we can see the challenges in the data section of the packets:

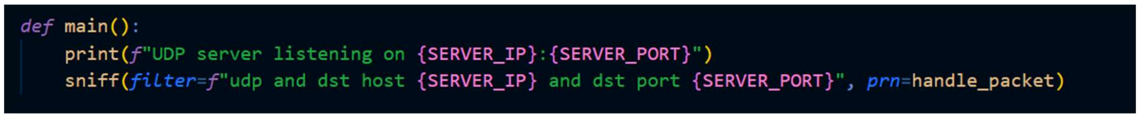


```
00 1c 7f a5 56 ca c2 40  4a ed 0f 29 08 00 45 00    ····V··@ J··)··E·
00 49 35 10 00 00 80 11  00 00 0a c8 1b e9 65 65    ·I5····· ······ee
65 65 d4 31 30 39 00 35  f1 c1 43 68 61 6c 6c 65    ee·109·5 ··Challe
6e 67 65 20 33 3a 20 50  6c 65 61 73 65 20 72 65    nge 3: P lease re
70 65 61 74 20 74 68 69  73 20 6e 75 6d 62 65 72    peat thi s number
3a 20 38 36 37 39 2e                                : 8679.
```

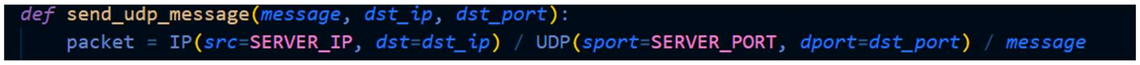
)

While writing the code we understand that we need to sniff and snoof the packets, because the client is expecting the answers to come from the ip it sent to.

So well use scapy:

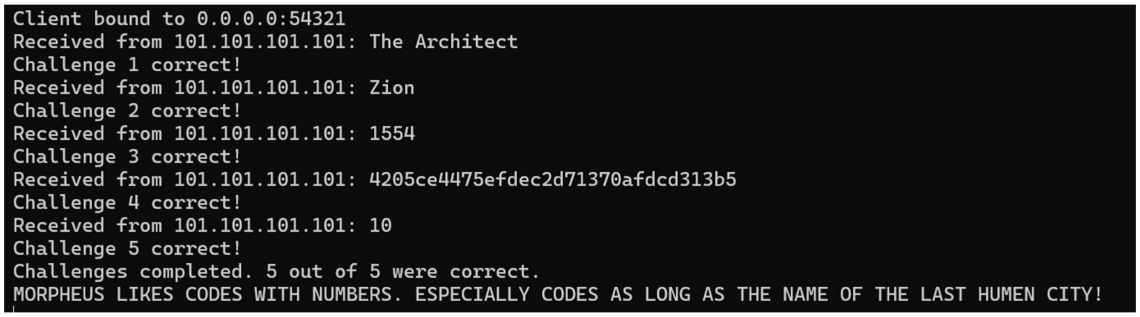


```python
def main():
    print(f"UDP server listening on {SERVER_IP}:{SERVER_PORT}")
    sniff(filter=f"udp and dst host {SERVER_IP} and dst port {SERVER_PORT}", prn=handle_packet)
```



```python
def send_udp_message(message, dst_ip, dst_port):
    packet = IP(src=SERVER_IP, dst=dst_ip) / UDP(sport=SERVER_PORT, dport=dst_port) / message
```

The full code: NetworkAnswer.py



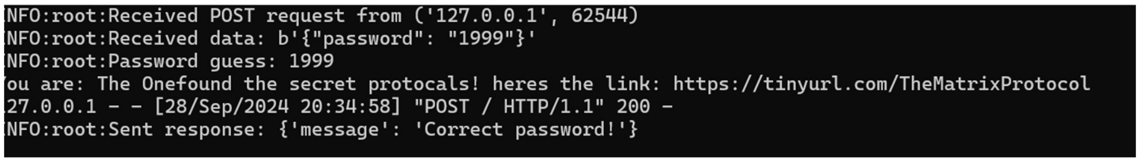
```
Client bound to 0.0.0.0:54321
Received from 101.101.101.101: The Architect
Challenge 1 correct!
Received from 101.101.101.101: Zion
Challenge 2 correct!
Received from 101.101.101.101: 1554
Challenge 3 correct!
Received from 101.101.101.101: 4205ce4475efdec2d71370afdcd313b5
Challenge 4 correct!
Received from 101.101.101.101: 10
Challenge 5 correct!
Challenges completed. 5 out of 5 were correct.
MORPHEUS LIKES CODES WITH NUMBERS. ESPECIALLY CODES AS LONG AS THE NAME OF THE LAST HUMEN CITY!
```

We got a hint for a password, probably for the the_source.pyc, the hint just says that the code is a 4-digit in. so let's write a http client to brute force server:

The full code: http_cracker.py

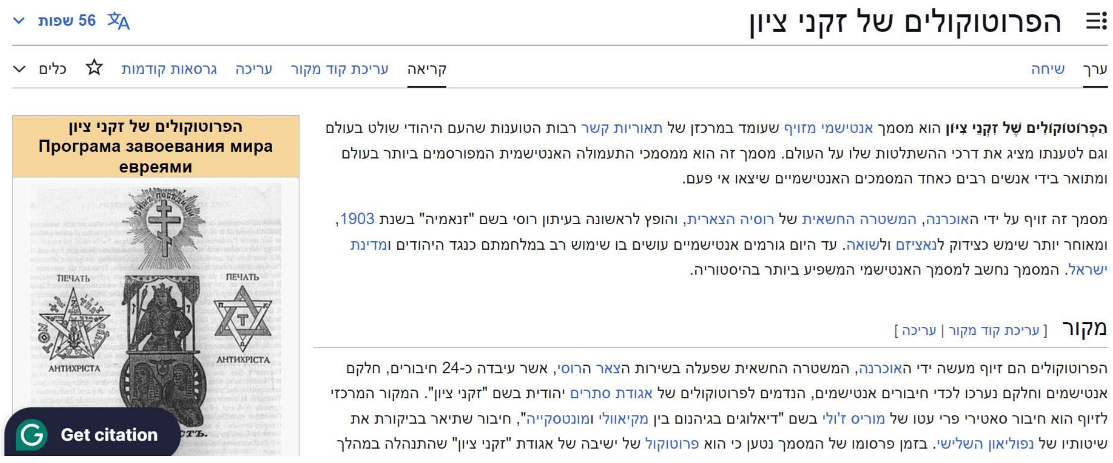
And we find that the code is: 1999 (the year the Matrix came out_)

and we can see we get a http link to the protocols:



```
NFO:root:Received POST request from ('127.0.0.1', 62544)
NFO:root:Received data: b'{"password": "1999"}'
NFO:root:Password guess: 1999
ou are: The Onefound the secret protocals! heres the link: https://tinyurl.com/TheMatrixProtocol
27.0.0.1 - - [28/Sep/2024 20:34:58] "POST / HTTP/1.1" 200 -
NFO:root:Sent response: {'message': 'Correct password!'}
```

And if we open the link to see the protocols:



הפרוטוקולים של זקני ציון

הפרוטוקולים של זקני ציון
Програма завоевания мира евреями

הַפְּרוֹטוֹקוֹלִים שֶׁל זִקְנֵי צִיּוֹן הוא מסמך אנטישמי מזויף שעומד במרכזן של תאוריות קשר רבות הטוענות שהעם היהודי שולט בעולם וגם לטענתו מציג את דרכי ההשתלטות שלו על העולם. מסמך זה הוא ממסמכי התעמולה האנטישמית המפורסמים ביותר בעולם ומתואר בידי אנשים רבים כאחד המסמכים האנטישמיים שיצאו אי פעם.

מסמך זה זויף על ידי האוכרנה, המשטרה החשאית של רוסיה הצארית, והופץ לראשונה בעיתון רוסי בשם "זנאמיה" בשנת 1903, ומאוחר יותר שימש כצידוק לנאציזם ולשואה. עד היום גורמים אנטישמיים עושים בו שימוש רב במלחמתם כנגד היהודים ומדינת ישראל. המסמך נחשב למסמך האנטישמי המשפיע ביותר בהיסטוריה.

מקור [עריכת קוד מקור | עריכה]

הפרוטוקולים הם זיוף מעשה ידי האוכרנה, המשטרה החשאית שפעלה בשירות הצאר הרוסי, אשר עיבדה כ-24 חיבורים, חלקם אנטישמים וחלקם נערכו לכדי חיבורים אנטישמים, הנדמים לפרוטוקולים של אגודת סתרים יהודית בשם "זקני ציון". המקור המרכזי לזיוף הוא חיבור סאטירי פרי עטו של מוריס ז'ולי בשם "דיאלוגים בגיהנום בין מקיאוולי ומונטסקייה", חיבור שתיאר בביקורת את שיטותיו של נפוליאון השלישי. בזמן פרסומו של המסמך נטען כי הוא פרוטוקול של ישיבה של אגודת "זקני ציון" שהתנהלה במהלך

<u>We found the protocols of the Matrix!!!</u>