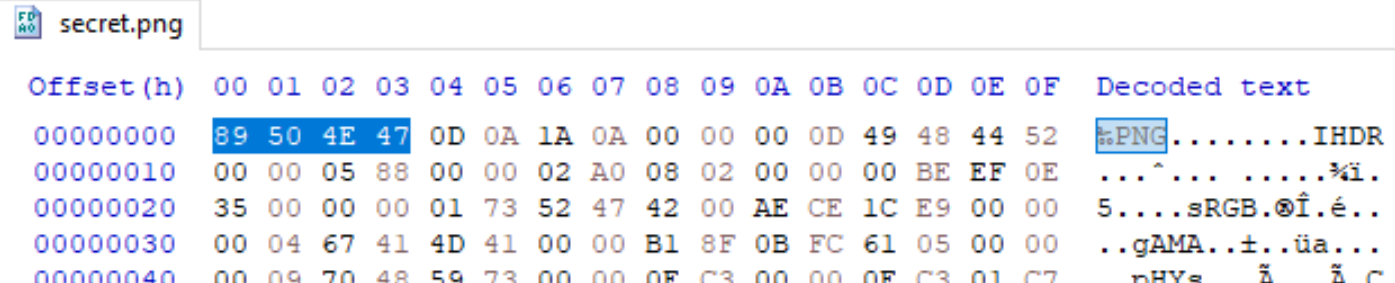
**BlackBox 2**

1. First, when I opened the folder I realized that there are 2 files which should be extracted but I don’t have their password so may I find them in the next stages.
2. Second, I tried to open the image file **secret.png** using a basic images program(like **Photos** on windows platform) but the file didn’t open.
3. Then I opened the image using hex editor (I used **HxD**) and found out that the **magic number** of the **png** format file was wrong, so I changed it to the right one and it worked! You can see that in the following image:



The image is:



1. In the image I saw a combination of characters **ODQxMjM0==** which looked like an encoded text in **UTF-8** encryption. Therefore, I wrote this script in python using **base64** library to decrypt it:

import base64

def decrypt(s:str):

    # The function gets a string.

    # The function returns the string after decrypting it.

    return base64.b64decode(s.encode()).decode()

print(decrypt("ODQxMjM0=="))

The output: **841234**

1. Then, I thought about what to do with this number. I tried using it as the password to check if it’s possible to extract the **secret.rar** folder and it worked!
2. Afterwards, I ran the **secret.exe** file using cmd which printed this text:

**>secret.exe**

**This software was valid up until 2011-11-11 11:11:11**

This output led me to change my machine’s time to a date before that: **2011-11-11 11:11:11**.

1. I run **secret.exe** again and it printed this:

**>secret.exe**

**Date Validation was previously done. Skipping condition check.**

**Current license value: a43fa**

**The current license registry value is invalid.**

**Expected that the sum of the hex values should be above 0x50 and also can also can be divived by 3 without a reminder.**

**Traceback (most recent call last):**

**File "secret.py", line 57, in <module>**

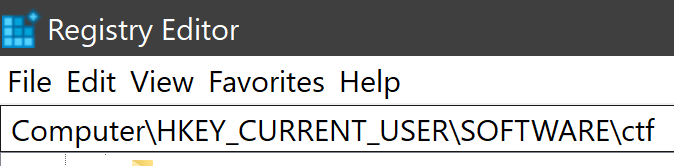
**File "secret.py", line 42, in check\_licence\_registry**

**NameError: name 'exit' is not defined**

**[33332] Failed to execute script 'secret' due to unhandled exception!**

Now I figured that it worked because of the message: “**Date Validation was previously done**”. In addition, I found that the **current license value** that showed, “**a43fa**”, is invalid because it’s hex value should be **above 0x50 and dividable by 3 without a reminder.**

1. Therefore, I tried to change this **current license value** to another value which follows the requirements (like: “**fffffffffffff**”) using the registry in the following path(I found it by using the **find** function with the “**a43fa”** value):



1. After I ran **secret.exe** again surprisingly the program started running a server by this output:

**>secret.exe**

**Date Validation was previously done. Skipping condition check.**

**Current license value: fffffffffffff**

**The current license is valid.**

**\* Serving Flask app 'secret'**

**\* Debug mode: on**

**Starting login server on port 8080...**

**WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.**

**\* Running on http://127.0.0.1:5000**

**Press CTRL+C to quit**

**\* Restarting with stat**

**Date Validation was previously done. Skipping condition check.**

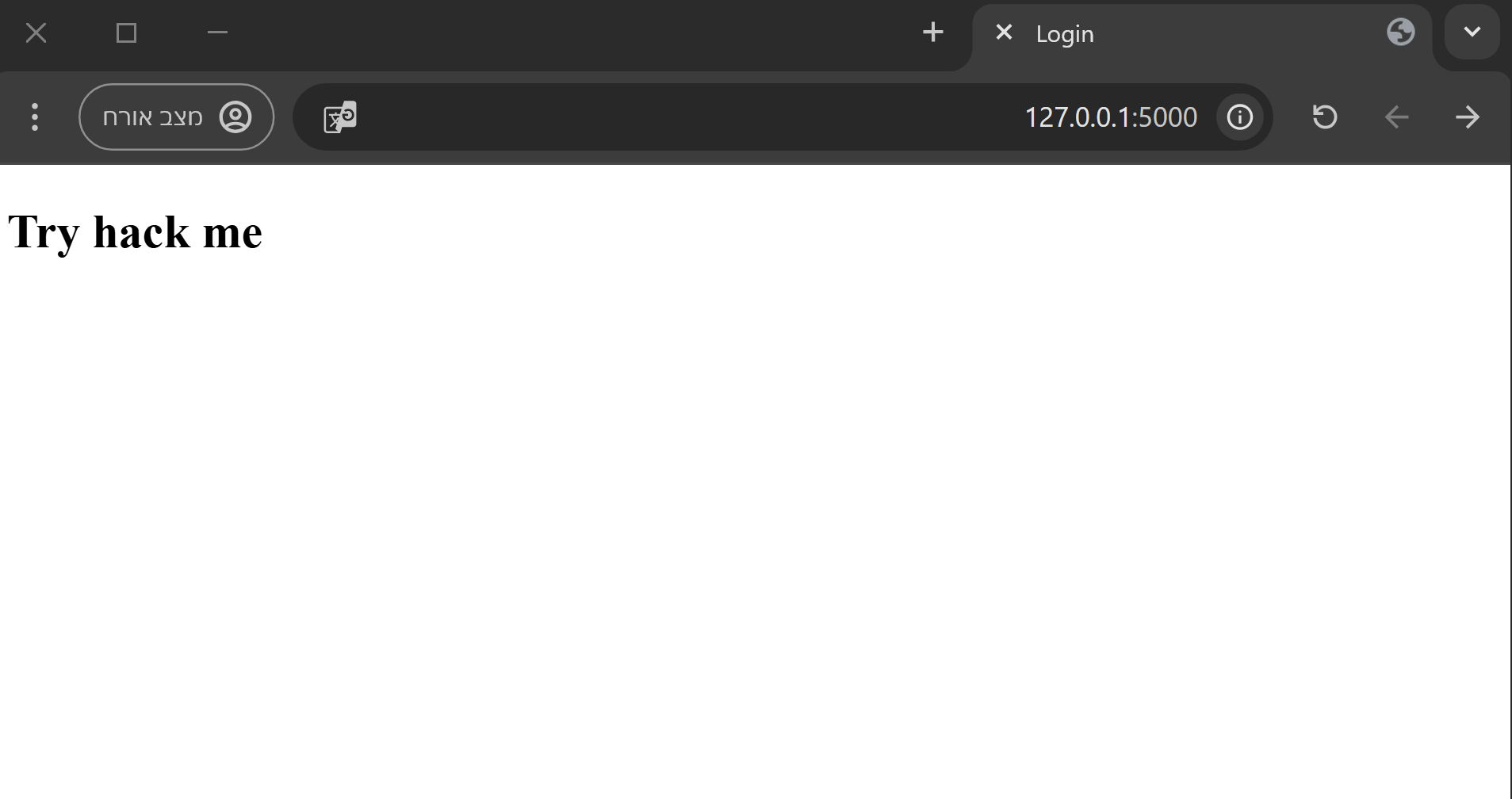
**Current license value: fffffffffffff**

**The current license is valid.**

**\* Debugger is active!**

**Starting login server on port 8080...**

**\* Debugger PIN: 130-702-582**

1. From this message I found some things: the **license** value is valid and the program running an http server which run on the **localhost** with the port **5000**. Therefore, I open a browser and entered the url of the server which is: “**http://127.0.0.1:5000**”. An image from the website: 
2. In the inspector I found this **JavaScript** script:

<script>

const x = window.location.href.split("\x69\x68\x61\x63\x6b\x79\x6f\x75");

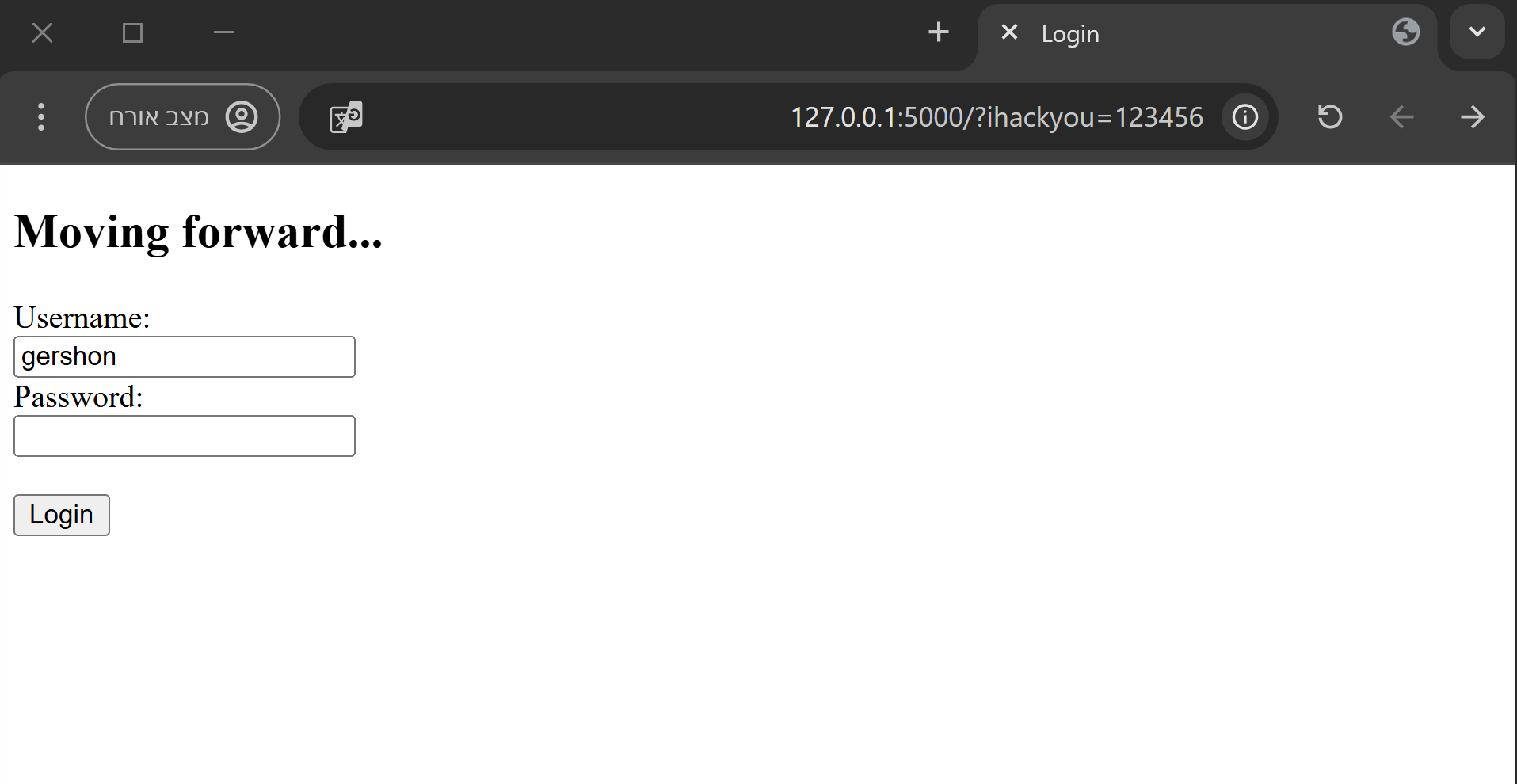
if (x.length === 1) {

console.log("You are missing something");

}

</script>

Then I went to the **Console** of the inspector and entered this **JavaScript** command to understand the string value : **> console.log("\x69\x68\x61\x63\x6b\x79\x6f\x75");** The output of this command was: “**ihackyou**”. Then I understood from the **JavaScript** script that probably something different will happen if the url contain the substring “**ihackyou**”.

1. Now, I add a simple unimportant parameter is the url with the name “**ihackyou**” and add it a random value and the website loaded a new page which seems like a login page: 
2. I have noticed that the username field already contains a defualt value which is: “**gershon**”. Then I tried to implement sql injection(like:”**1' OR 1=1**”) on the password field and it worked!
3. Now a new page was loaded on the website:

תמונה שמכילה טקסט, צילום מסך, חשמל, מולטימדיה

התיאור נוצר באופן אוטומטי

1. On this page, in the inspector, I found this JS script:

const urlParams = new URLSearchParams(window.location.search);

const a = urlParams.get("\x69\x68\x61\x63\x6b\x79\x6f\x75");

function \_\_das213csdas43(\_\_321dsa) {

    return 3 + \_\_321dsa -  parseInt("\x01".charCodeAt(0), 16)

}

const y = \_\_das213csdas43(3)

if (a && a  == "\x63\x79\x62\x72\x65\x72".split("\x72")[(Math.sin(Math.PI / 2) \*\* 2) + (Math.cos(0) \*\* 2) - 1 + y - 5]) {

    document.write('good');

    console.log("good job")

    window.location.href = "\x79\x6f\x75\x61\x72\x65\x63\x6c\x6f\x73\x65" + a

} else {

    document.write('Think harder now');

}

After analyzing it, I understood that “**x63\x79\x62\x72\x65\x72**” means “**x63ybrer**” and “**\x79\x6f\x75\x61\x72\x65\x63\x6c\x6f\x73\x65**” means “**youareclose**”.

In addition, I found that “**Math.cos(0) \*\* 2) - 1 + y - 5]**” **= 0**

“**Math.sin(Math.PI / 2) \*\* 2**” = **1**.

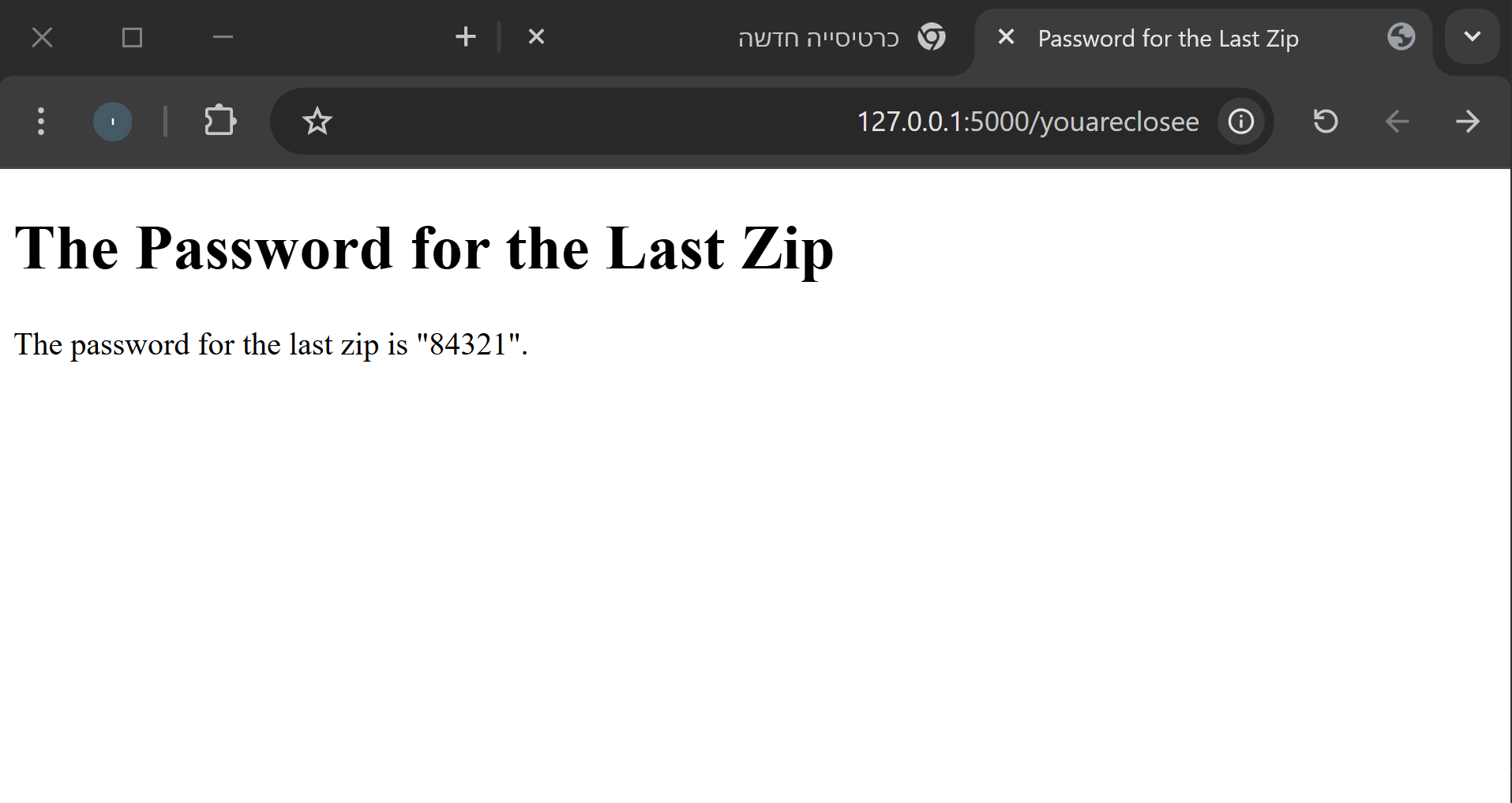
|||

\_|||\_

\|||/

\/

"**\x63\x79\x62\x72\x65\x72".split("\x72")[(Math.sin(Math.PI / 2) \*\* 2) + (Math.cos(0) \*\* 2) - 1 + y - 5])**” = “**e**”.

1. Therefore, I understood that I should add the same field of the parameter “**ihackyou**” to the url with the value of “**e**” in the following syntax “**?ihackyou=e**”.
2. Then, a new page has been loaded: 

This page, as you can see, contains the second zip folder’s code to unzip it.

1. After unzipping it and running stage2.exe file via the cmd, this program was shown up:

**>stage2.exe**

**You are pretty, what's your phone number?**

**Enter your phone number:**

So, I entered a random phone number(like:0511111111) and this message suddenly appeared:

**Sent UDP packet to loopback**

**You are pretty, what's your phone number?**

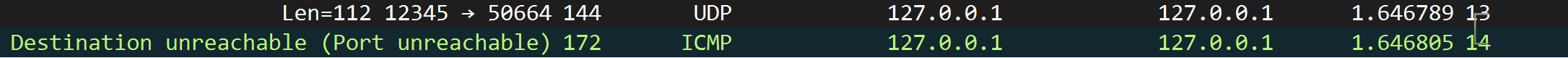
**Enter your phone number:**

From here, I understood that probably an UDP packet was sent to the **loopback** address(which is **127.0.0.1**). This message led me to open **Wireshark** and find this message.

1. When I tried to use Wireshark, the program suddenly closed it and printed this message:

**You aren't allowed to use Wireshark.exe**

Hence, I chose to run **Wireshark.exe** program as administrator.

After sniffing those 2 packets: 

I followed the UDP stream and found this message:תמונה שמכילה צילום מסך, טקסט, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

After reading this message I checked on the internet the number of “**Ana Zak**” and found that its first 6 digits are: **052538**. In addition, I thought that maybe the given **phonenumber** it’s probably a hash like **md5** or **sha-1** so I used both of them in order to find the correct one. Therefore, I wrote this py script to find the full number:

def generate\_sha1\_hash(input\_string:str):

    # The function gets a string.

    # The function returns the hash of the given string according to SHA-1.

    sha1\_hash = hashlib.sha1()

    sha1\_hash.update(input\_string.encode())

    return sha1\_hash.hexdigest()

def generate\_md5\_hash(input\_string:str):

    # The function gets a string.

    # The function returns the hash of the given string according to SHA-1.

    sha1\_hash = hashlib.md5()

    sha1\_hash.update(input\_string.encode())

    return sha1\_hash.hexdigest()

def generate\_phonenumber\_bruteforce():

    # Bruteforcing

    for a in range(10):

        for b in range(10):

            for c in range(10):

                for d in range(10):

                    print(f"{a}{b}{c}{d}")

                    yield "052538" + f"{a}{b}{c}{d}"

print("Bruteforce Started!")

for phone\_number in generate\_phonenumber\_bruteforce():

    if generate\_sha1\_hash(phone\_number) == "6688795677c1c8f2d3cd14b710f60153":

        print(f"The phone number of the hash is: {phone\_number}. The correct hash is: sha-1")

        break

    elif generate\_md5\_hash(phone\_number) == "6688795677c1c8f2d3cd14b710f60153":

        print(f"The phone number of the hash is: {phone\_number}. The correct hash is: md5")

        break

The output is:

**“The phone number of the hash is: 0525384975. The correct hash is: md5”**

1. Now, I entered this phone number on the **stage2.exe** program which printed this:

**>stage2.exe**

**You are pretty, what's your phone number?**

**Enter your phone number:0525384975**

**Sent UDP packet to loopback**

**Nice... not so bad as i thought you are**

**Padding password with random bytes to fit 16 characters**

**Performing base64**

**Server is listening on port 8543**

**Enter password:**

When I tried to enter a random password(like: 123) it printed:

**Enter password: 123**

**Password incorrect. Try again.**

**Enter password:**

Therefore, I tried to create a client socket in **py** which is going to connect to the server of stage2. Because of this reason, I used the command netstat -a on the cmd to understand the protocol type of the port 8543 which the server probably use.

The script is:

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect(('127.0.0.1', 8543))

s.send("1234".encode())

I opened **Wireshark**(of course, in administrator mode) and after running this script I sniffed this packet:

תמונה שמכילה טקסט, צילום מסך, תוכנה, תכונות מולטימדיה

התיאור נוצר באופן אוטומטי

Then, I tried to decrypt this encryption using the **decrypt** function that I wrote from the beginning(on **section 4**) and used the encode() function on the result. The output is: “**mgdf32gf43@\x00\x00\x00\x00\x00”** so I understood that to this password has a padding characters and therefore I entered each combination of the password, but each time remove 1 character. In the end, the substring: “**mgdf32**” was the correct password and the program printed me this: “**WOW\_AMAZING\_JOB\_YOU\_FOUND\_THE\_SECRET654543**”.