**BlackBox1 – Solution**

1. Create an empty .json file named **“config.json”** so the server will be able to be executed.

You can create one by entering: echo > config.json

1. Execute **“server.exe”** and afterwards **“secret.exe”** files on separated **cmd** interfaces.
2. Enter any appropriate input to the **secret.exe** execution, for example:

Enter your username: abcd

Enter your password: 1234

1. Afterwards a packet will be sent to the server, so to track it open **Wireshark** and sniff the packet:

From here we can understand that the client (in red) sends a json format messages to the server with the **“username”** flag that given and some other flags like: **“action”, “token”** and **“hash”**. In addition, to the **config.json** file this data was added:

{"users": [ {"username": "abcd", "token": "MTIzNA=="}]}

Which means that the config file saves users’ details.

1. After writing this script in python:

import base64

def decrypt(encrypted\_text):

    # The function gets a string.

    # The function returns the given string after decrypting it.

    return base64.b64decode(encrypted\_text).decode()

print(decrypt("MTIzNA=="))

I found that the value “**MTIzNA==**“ which was found on **config.json** and the packet itself under the flag “**token**” is equals “**1234**” which is the password I entered on secret.txt(look at section number 2), so from this it can be concluded that the flag **token** saves passwords encrypted.

1. Now, I have tried to enter a username that already exist from my previous try “**abcd**” but with wrong password. Therefore, the **secret.exe** returned me this message: **“Error: Invalid token for existing user”** so from this it can be concluded that **secret.exe** check for already exist users and doesn’t create new one with a different password.
2. Afterwards, I tried to login as “**admin**” with the password “**1234**”. Surprisingly, the secret.exe tells me that “**Error: Bad Password**” which means that the username “admin” probably exist but with another password.
3. I wrote a script in python to edit a packet of a “**get-actions**” action with **admin** username and a random encrypted password as a token and send it to the server. This is the script:

import requests

URL = "http://localhost:5555"

# Get-actions request

message\_body = {"action": "get-actions", "type": "admin", "token": "MTIzNA=="}

respond = requests.post(url=URL, json=message\_body)

respond\_data = respond.json()

if respond.status\_code == 200:

    print(respond\_data)

else:

    print(f"Error{respond.status\_code}")

The server returned to me this message “**get-users”**. It seems to be an action mode so I changed the script’s “**message\_body”** parameter with this action mode:

message\_body = {"action": "get-users", "type": "admin", "token": "MTIzNA=="}

After running the script again and sniff it the returned message was: **“ {"users":[{"date registered":"XX/XX/2019","token":"Y3liZXI=","username":"admin"}]} ”.**

1. It can be concluded from this message that the **admin**’s user password is encrypted to **“Y3liZXI=”** and after decrypting it the password found to be “**cyber**”. In addition, the action **get-users** returns information about specific users. Moreover, the message looks like it is written in the format of **config.exe**.
2. Next, I tried to login with the username and the password of the **admin** user via the **secret.exe** but it returned me a new message “**Error: Good password, wrong hash**”, so now the problem is the hash.
3. After checking some possibilities, I found that the hashes of the other users(**which aren’t admin**) are the same for each day. In addition, I count the number of chars of a single hash and it’s 40 which means that they are probably based on “**SHA-1 hashing”.** Therefore, to verify this, I wanted to hash the date of this day **“17/10/2024”** in **SHA-1** using the following function:

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()

The results of the hash were exactly as the packet shown in:

The function hash:

“d3a193555c8ad52eadf486c309a935d5a040a83e”

The packet hash:

1. Afterwards, I looked again at the message from section 9 and find the date of the creation of the **admin** user: **"XX/XX/2019".** I understood that the month and day are censored, so I wrote this function to bruteforce all the days in 2019:

def check\_right\_hash():

    # The function gets nothing.

    # The function bruteforce all the 2019 dates and find the right hash of the admin user.

    # The function returns the hash which fits to the date.

    URL = "http://localhost:5555"

    message\_body = {"action": "login", "username": "admin", "token": "Y3liZXI=", "hash": ""}

    current\_date = ""

    for current\_month in range(1, 13):

        for current\_day in range(1, 31):

            current\_date = str(current\_day)+"/"+str(current\_month)+"/2019"

            message\_body["hash"] = base64\_decrypter.generate\_sha1\_hash(current\_date)

            respond = requests.post(url=URL, json=message\_body)

            # respond\_data = respond.json()

            if respond.status\_code == 200:

                print("success - " + current\_date + " - " + message\_body["hash"])

                return message\_body["hash"]

            print("fail - " + current\_date + " - " + message\_body["hash"])

The result is: **“success - 11/11/2019 - b1f38bb02297db3a5ca9324b3089fd73459d60b9”.**

1. Then, I apply the hash to the message\_body on the script from section 9:
2. import requests
3. URL = "http://localhost:5555"
4. # Login request
5. message\_body = {"action": "login", "username": "admin", "token": "Y3liZXI=", "hash": "b1f38bb02297db3a5ca9324b3089fd73459d60b9"}
6. respond = requests.post(url=URL, json=message\_body)
7. respond\_data = respond.json()
8. if respond.status\_code == 200:
9. print(respond\_data)
10. else:
11. print(f"Error{respond.status\_code}")

While running this script I sniffed the packet and finished this blackbox!

תמונה שמכילה טקסט, צילום מסך, גופן, מספר

התיאור נוצר באופן אוטומטי