**BASH CHALLENGES**

TASK-1 (LINUX COMMANDS)

I first learnt all the commands given, from the below link

<https://youtu.be/ZtqBQ68cfJc?si=Apn9Wq5g7-GuaIt8>

I then solved all the problems one after another. I did not face much of an issue till question number 9.

I had to use the man page of the terminal, google and Chat GPT for the questions 10-14.

**CODING CHALLENGES**

TASK-2 AND 3 (HACKERRANK AND LEAP YEAR QUESTION)

These tasks were comparatively easy. Did not take much of my time. The third task (leap year) took a bit of time to think, but go it at last.

TASK – 4 (SOCKET PROGRAMMING)

This is a very interesting task for me. I’ve wanted to build a chatbot in my Sem holidays and I built one, because of this task. I learnt socket programming, SHA-256 and HMAC from YouTube videos. I took the help of Chat-GPT to view the structure of the code and made one on my own. This is the final code I’ve made.

**Client Code :**

import socket

from cryptography.fernet import Fernet

import hashlib

import hmac

import os

def generate\_key():

return Fernet.generate\_key()

def encrypt\_message(key, message):

cipher = Fernet(key)

encrypted\_message = cipher.encrypt(message.encode())

return encrypted\_message

def decrypt\_message(key, encrypted\_message):

cipher = Fernet(key)

decrypted\_message = cipher.decrypt(encrypted\_message).decode()

return decrypted\_message

def calculate\_sha512(data):

sha512 = hashlib.sha512()

sha512.update(data)

return sha512.digest()

def calculate\_hmac(key, data):

h = hmac.new(key, data, hashlib.sha512)

return h.digest()

def start\_client():

host = '127.0.0.1'

port = 12345

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((host, port))

key = client\_socket.recv(1024)

hmac\_key = client\_socket.recv(32)

try:

while True:

message = input("Enter your message: ")

encrypted\_message = encrypt\_message(key, message)

client\_socket.sendall(encrypted\_message)

client\_socket.sendall(calculate\_sha512(encrypted\_message))

client\_socket.sendall(calculate\_hmac(hmac\_key, encrypted\_message))

data = client\_socket.recv(4096)

received\_sha512 = client\_socket.recv(64)

received\_hmac = client\_socket.recv(64)

decrypted\_response = decrypt\_message(key, data)

if received\_sha512 != calculate\_sha512(decrypted\_response.encode()):

print("Integrity check failed. Message may have been tampered with.")

elif received\_hmac != calculate\_hmac(hmac\_key, data):

print("HMAC verification failed. Message may not be authentic.")

else:

print(f"Server response: {decrypted\_response}")

except KeyboardInterrupt:

print("Exited by user")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

start\_client()

**Server Code :**

import socket

from cryptography.fernet import Fernet

import hashlib

import hmac

import os

def generate\_key():

return Fernet.generate\_key()

def encrypt\_message(key, message):

cipher = Fernet(key)

encrypted\_message = cipher.encrypt(message.encode())

return encrypted\_message

def decrypt\_message(key, encrypted\_message):

cipher = Fernet(key)

decrypted\_message = cipher.decrypt(encrypted\_message).decode()

return decrypted\_message

def calculate\_sha512(data):

sha512 = hashlib.sha512()

sha512.update(data)

return sha512.digest()

def calculate\_hmac(key, data):

h = hmac.new(key, data, hashlib.sha512)

return h.digest()

def start\_server():

host = '127.0.0.1'

port = 12345

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((host, port))

server\_socket.listen(1)

print(f"Server listening on {host}:{port}")

client\_socket, address = server\_socket.accept()

print(f"Connection from {address}")

key = generate\_key()

hmac\_key = os.urandom(32) # 256-bit key for HMAC

client\_socket.sendall(key)

client\_socket.sendall(hmac\_key)

try:

while True:

data = client\_socket.recv(4096)

if not data:

break

received\_sha512 = client\_socket.recv(64)

received\_hmac = client\_socket.recv(64)

decrypted\_message = decrypt\_message(key, data)

if received\_sha512 != calculate\_sha512(data):

print("Integrity check failed. Message may have been tampered with.")

elif received\_hmac != calculate\_hmac(hmac\_key, data):

print("HMAC verification failed. Message may not be authentic.")

else:

print(f"Received: {decrypted\_message}")

response = input("Enter your response: ")

encrypted\_response = encrypt\_message(key, response)

client\_socket.sendall(encrypted\_response)

client\_socket.sendall(calculate\_sha512(response.encode()))

client\_socket.sendall(calculate\_hmac(hmac\_key, encrypted\_response))

except KeyboardInterrupt:

print("Exited by user")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

start\_server()

**REVERSING CHALLENGES**

TASK-5 (ASM CHALLENGES)

I watched the entire video series of pwn college. It was useful for me to do the tasks. The main issue I faced during this task is, I did not know where to start. When I clicked on “Start”, it gave me 2 options, VS Code and GUI desktop. I opened VS Code and tried to type the answers for the questions in the workspace and ran codes in the terminal but it did not workout. I spent a lot of time doing this in different ways.

I then opened the GUI desktop and became confused completely. As I has no other way, I started digging into the folders of the GUI desktop and found a “run” text file. I read through the file, and noticed that it was a python file. I continued reading it, and found classes which had questions of every ASM level in it. So I thought of a way to run the text file as a python file.

I searched for python softwares in the desktop and found Jupyter Notebook. I copied the text file and pasted in “Jupyter Lab”, and then ran it. But the code showed few errors. I then started to correct those errors, but was able to correct only 1. I was then completely tired and thought of leaving it.

After a while I again sat to open it, I noticed that the files I created in VS Code are there in the GUI Desktop. So I opened the VS Code and tried to access the run file from it , and it worked out.

Here are the codes of all the Levels of ASM (1-10) :

LEVEL-1

.global \_start

.intel\_syntax noprefix

\_start:

mov rdi, 0x1337

pwn.college{IeT6mCkBClWUMIQjgL-wyGvaBTP.0FN5EDL0gjNzQzW}

LEVEL-2

.global \_start

.intel\_syntax noprefix

\_start:

mov rax, 0x1337

mov r12, 0xCAFED00D1337BEEF

mov rsp, 0x31337

pwn.college{c5BVpCFgP6b7vYwgmbsp4pVBlX7.dBTM4MDL0gjNzQzW}

LEVEL-3

.global \_start

.intel\_syntax noprefix

\_start:

add rdi, 0x331337

pwn.college{AgMDoAaMNAjjnPZ6mg4UFjSgEKw.0VN5EDL0gjNzQzW}

LEVEL-4

.global \_start

.intel\_syntax noprefix

\_start:

imul rdi, rsi

mov rax, rdi

add rax, rdx

pwn.college{UIQzM8BboM3yQ689boJaMFtdwLA.0lN5EDL0gjNzQzW}

LEVEL-5

.global \_start

.intel\_syntax noprefix

\_start:

xor rdx, rdx

mov rax, rdi

div rsi

pwn.college{owKbaWgVQydAjUusWFdHSivM9Qm.01N5EDL0gjNzQzW}

LEVEL-6

.global \_start

.intel\_syntax noprefix

\_start:

xor rdx, rdx

mov rax, rdi

div rsi

mov rax, rdx

pwn.college{8zZGADaC8npAVm4UsR5SpNuSZP9.0FO5EDL0gjNzQzW}

LEVEL-7

.global \_start

.intel\_syntax noprefix

\_start:

mov ah, 0x42

pwn.college{Uw\_nh22mdJ8FuYGbQTRiElGNhzD.dFTM4MDL0gjNzQzW}

LEVEL-8

.global \_start

.intel\_syntax noprefix

\_start:

mov al, dil

mov bx, si

pwn.college{YjV9gPIvXGHYiCZd0iUaRkIg5la.0VO5EDL0gjNzQzW}

LEVEL-9

.global \_start

.intel\_syntax noprefix

\_start:

mov rax, rdi

shl rax, 24

shr rax, 56

pwn.college{Qqjlkf1CoJ3P9jLYa9UkzbGjN4g.0FMwIDL0gjNzQzW}

LEVEL-10

.global \_start

.intel\_syntax noprefix

\_start:

and rdi, rsi

and rax, rdi

pwn.college{0iOkM-7WieSLFDXsO0V9yJqoLkl.0VMwIDL0gjNzQzW}

TASK-6 (FINDING FLAGS)

I watched the entire 2-hour video provided. It was very helpful to do the task.

1. LearnJava.zip

I extracted the downloaded file, and opened the file using VS Code. VS code recommended, FernFlower to decompile the file, so I did the same.

When I read the code, the code had 2 parts. The user inputting the password, and a function telling if the password is correct or not. When I saw the “checkPassword” function it was converting the password into Byte Code, which is into ASCII number. Hence finding the characters from the ASCII number is needed to find the password. The ASCII number gave the password as - “KnoW\_YouR\_A5C115}”. When I ran the code and typed it, it did not work.

So, I went through the code again and saw the main function. The string that was passed into the “checkPassword” function was the substring of the input password. The string “flag{“ was cut off from the user input before passing it into the “checkPassword” function. Hence the final flag I got was …..

flag{KnoW\_YouR\_A5C115}

1. Encryptedflag.zip

I extracted the downloaded file, and opened the file using VS Code. VS code recommended, FernFlower to decompile the file, so I did the same.

The code has 3 parts. The “yes” function which does the XOR encryption. The “no” function which does t hash encryption and the main code which substitutes the variables into these functions.

According to the code, the user gives 2 inputs. If the inputs matches “G[Z@Z@G[VU\_RT” and “flag” then the flag is generated, where “flag” undergoes XOR encryption and “flag” and “G[Z@Z@G[VU\_RT” undergoes hash encryption.

The concatenated output of different variables gives the flag , which is –

flag{thisistheflag1lG1lY}