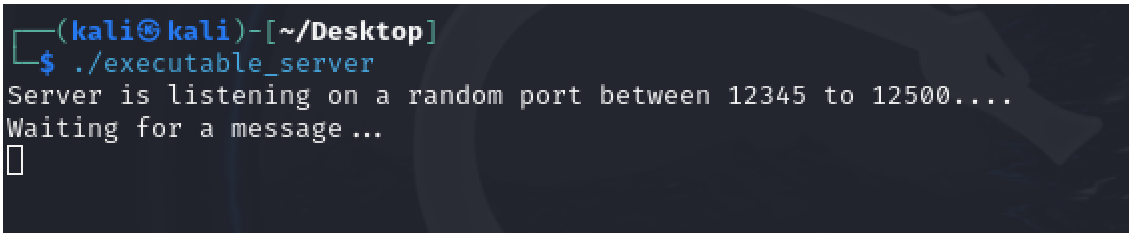
Question 4.

**a) Opening port**

In Terminal a,

Running `./executable_server`



```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./executable_server
Server is listening on a random port between 12345 to 12500....
Waiting for a message ...
```

In Terminal b,

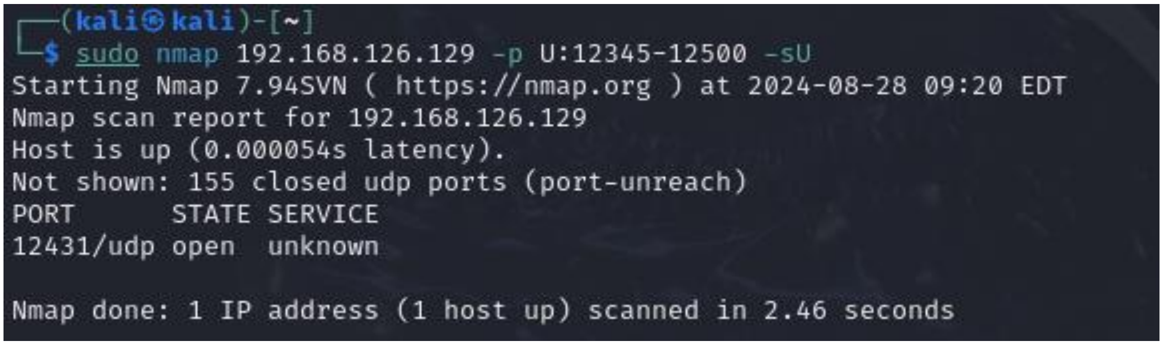
You can run these commands in the root by:

```
sudo su
```

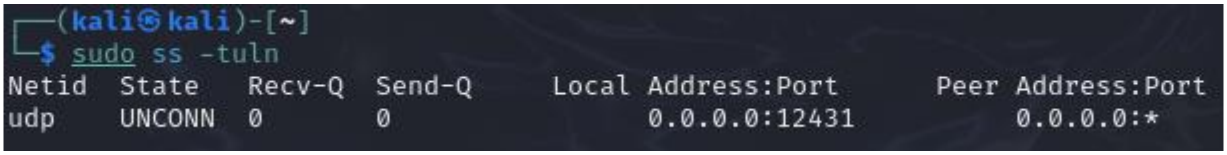
This will lead to the root.

So the following commands do not need `sudo`.

But here, I do not run from the root.



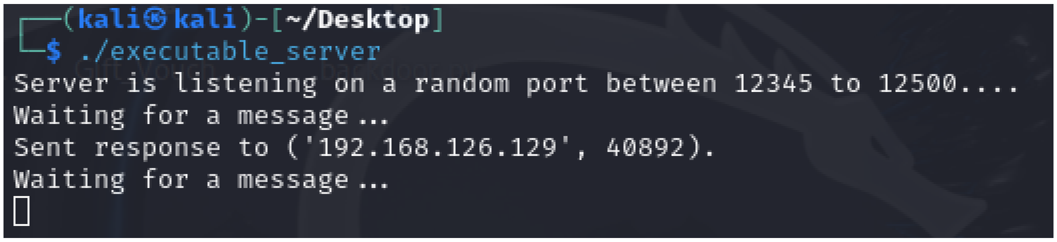
```
┌──(kali㉿kali)-[~]
└─$ sudo nmap 192.168.126.129 -p U:12345-12500 -sU
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-08-28 09:20 EDT
Nmap scan report for 192.168.126.129
Host is up (0.000054s latency).
Not shown: 155 closed udp ports (port-unreach)
PORT      STATE SERVICE
12431/udp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 2.46 seconds
```



```
┌──(kali㉿kali)-[~]
└─$ sudo ss -tuln
Netid  State   Recv-Q  Send-Q      Local Address:Port      Peer Address:Port
udp    UNCONN  0       0                 0.0.0.0:12431          0.0.0.0:*
```

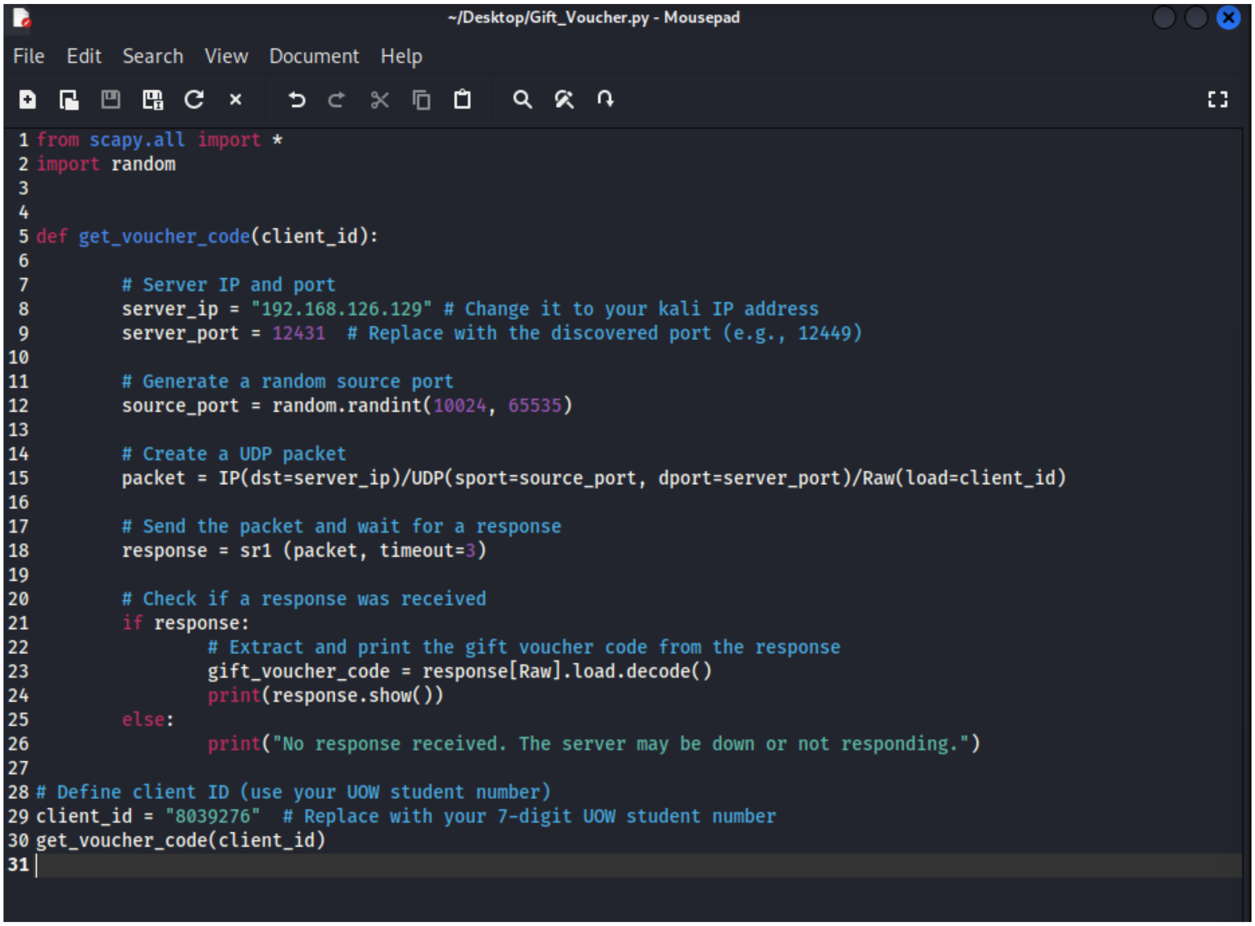
Port **12431** is open.

In Terminal a,



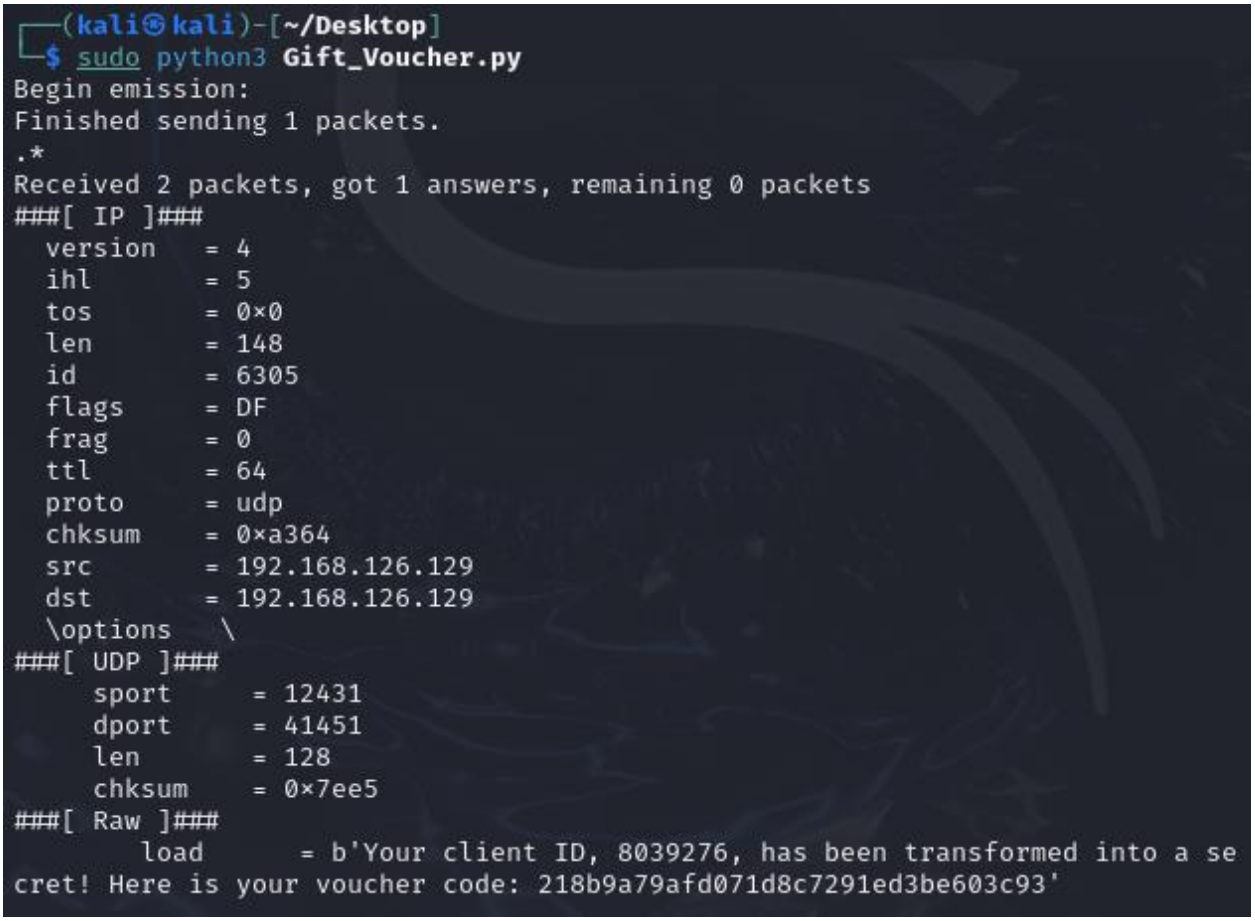
```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./executable_server
Server is listening on a random port between 12345 to 12500....
Waiting for a message ...
Sent response to ('192.168.126.129', 40892).
Waiting for a message ...
```

**b) Generating the gift voucher code.**



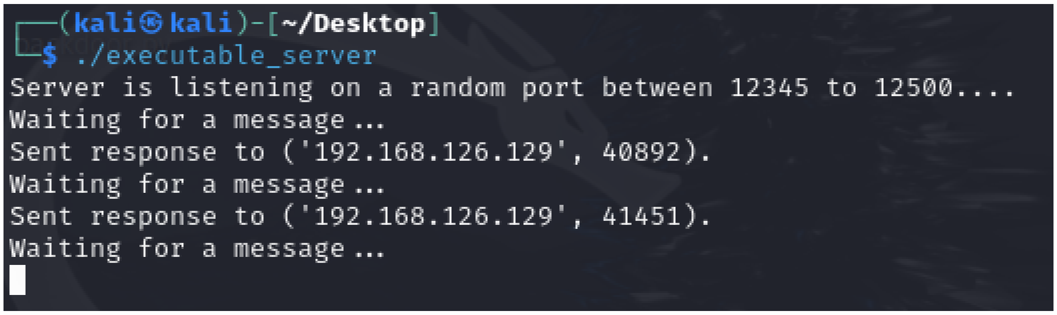
```python
from scapy.all import *
import random


def get_voucher_code(client_id):

        # Server IP and port
        server_ip = "192.168.126.129" # Change it to your kali IP address
        server_port = 12431  # Replace with the discovered port (e.g., 12449)

        # Generate a random source port
        source_port = random.randint(10024, 65535)

        # Create a UDP packet
        packet = IP(dst=server_ip)/UDP(sport=source_port, dport=server_port)/Raw(load=client_id)

        # Send the packet and wait for a response
        response = sr1 (packet, timeout=3)

        # Check if a response was received
        if response:
                # Extract and print the gift voucher code from the response
                gift_voucher_code = response[Raw].load.decode()
                print(response.show())
        else:
                print("No response received. The server may be down or not responding.")

# Define client ID (use your UOW student number)
client_id = "8039276"  # Replace with your 7-digit UOW student number
get_voucher_code(client_id)

```

In Terminal b, by running the program, we got the gift voncher code.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo python3 Gift_Voucher.py
Begin emission:
Finished sending 1 packets.
.*
Received 2 packets, got 1 answers, remaining 0 packets
###[ IP ]###
  version   = 4
  ihl       = 5
  tos       = 0x0
  len       = 148
  id        = 6305
  flags     = DF
  frag      = 0
  ttl       = 64
  proto     = udp
  chksum    = 0xa364
  src       = 192.168.126.129
  dst       = 192.168.126.129
  \options   \
###[ UDP ]###
     sport     = 12431
     dport     = 41451
     len       = 128
     chksum    = 0x7ee5
###[ Raw ]###
        load      = b'Your client ID, 8039276, has been transformed into a se
cret! Here is your voucher code: 218b9a79afd071d8c7291ed3be603c93'
```

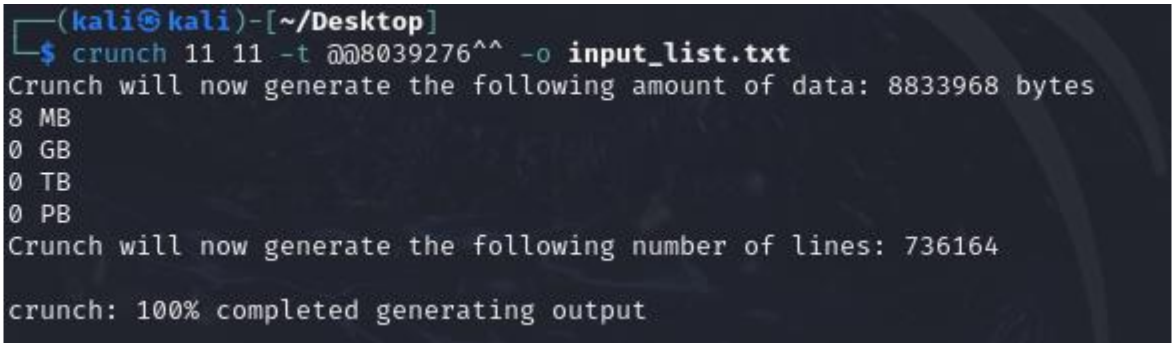
In Terminal a, there is a change.



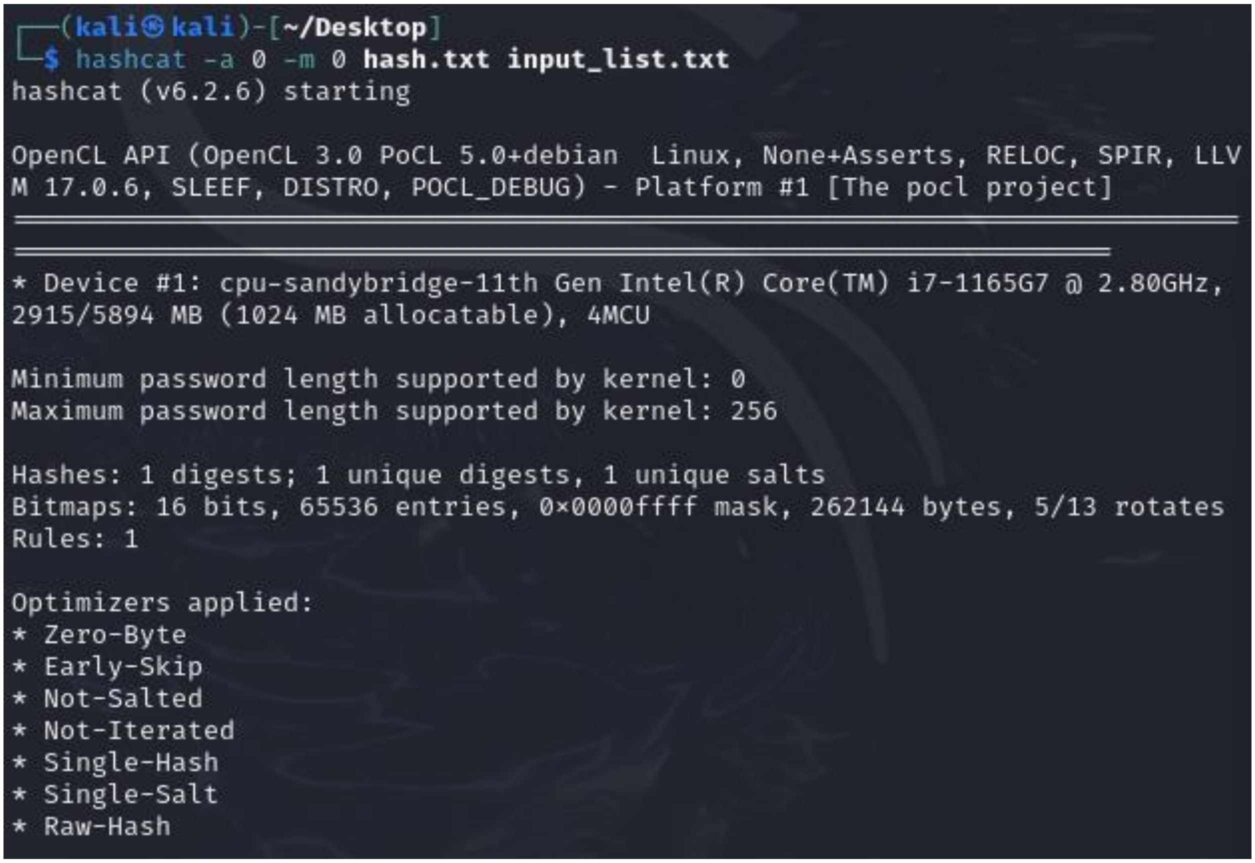
```
┌──(kali㉿kali)-[~/Desktop]
└─$ ./executable_server
Server is listening on a random port between 12345 to 12500....
Waiting for a message ...
Sent response to ('192.168.126.129', 40892).
Waiting for a message ...
Sent response to ('192.168.126.129', 41451).
Waiting for a message ...
█
```

**c) Values of A and B**

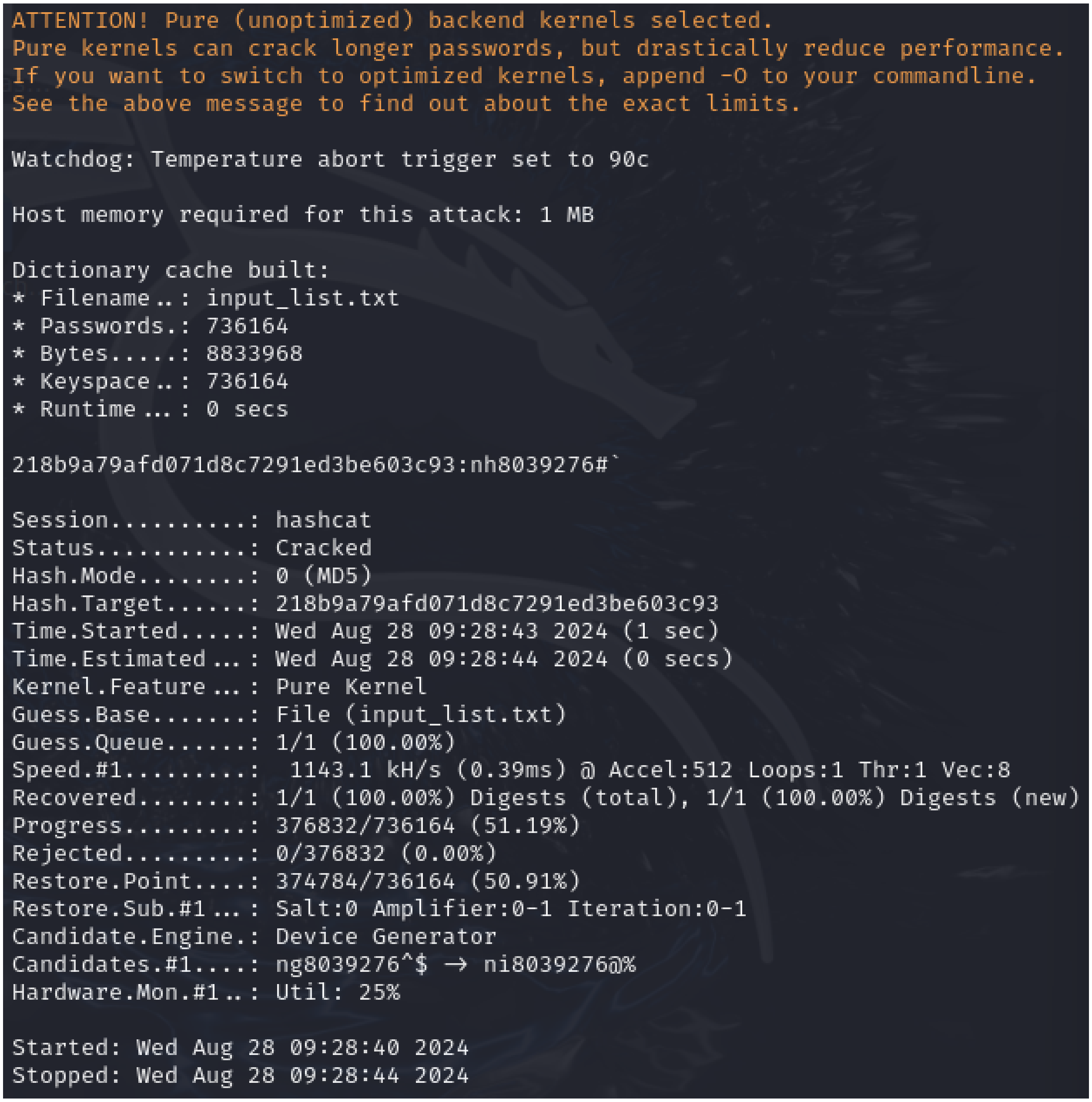
In Terminal b, crunch is used to get the values of A and B. @@ represents lowercase letters and ^^ represents all possible symbols. The output is saved in input_list.txt.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ crunch 11 11 -t @@8039276^^ -o input_list.txt
Crunch will now generate the following amount of data: 8833968 bytes
8 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 736164

crunch: 100% completed generating output
```



```
┌──(kali㉿kali)-[~/Desktop]
└─$ hashcat -a 0 -m 0 hash.txt input_list.txt
hashcat (v6.2.6) starting

OpenCL API (OpenCL 3.0 PoCL 5.0+debian  Linux, None+Asserts, RELOC, SPIR, LLV
M 17.0.6, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=============================================================================
====================================================================

* Device #1: cpu-sandybridge-11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz,
2915/5894 MB (1024 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Hash
* Single-Salt
* Raw-Hash
```



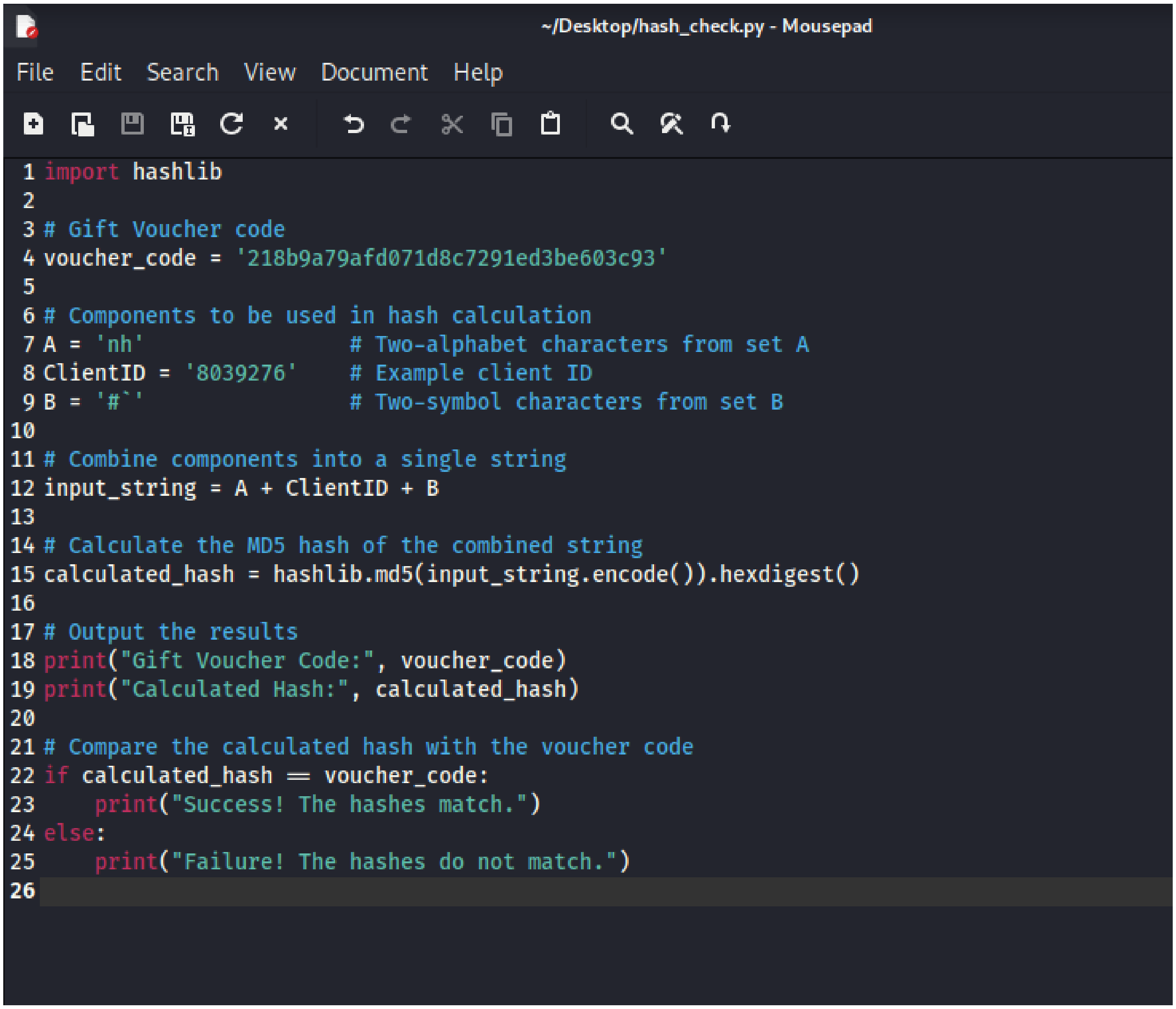
```
ATTENTION! Pure (unoptimized) backend kernels selected.
Pure kernels can crack longer passwords, but drastically reduce performance.
If you want to switch to optimized kernels, append -O to your commandline.
See the above message to find out about the exact limits.

Watchdog: Temperature abort trigger set to 90c

Host memory required for this attack: 1 MB

Dictionary cache built:
* Filename..: input_list.txt
* Passwords.: 736164
* Bytes.....: 8833968
* Keyspace..: 736164
* Runtime...: 0 secs

218b9a79afd071d8c7291ed3be603c93:nh8039276#`

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 0 (MD5)
Hash.Target......: 218b9a79afd071d8c7291ed3be603c93
Time.Started.....: Wed Aug 28 09:28:43 2024 (1 sec)
Time.Estimated...: Wed Aug 28 09:28:44 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (input_list.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1143.1 kH/s (0.39ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 376832/736164 (51.19%)
Rejected.........: 0/376832 (0.00%)
Restore.Point....: 374784/736164 (50.91%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: ng8039276^$ -> ni8039276@%
Hardware.Mon.#1..: Util: 25%

Started: Wed Aug 28 09:28:40 2024
Stopped: Wed Aug 28 09:28:44 2024
```

nh8039276#`

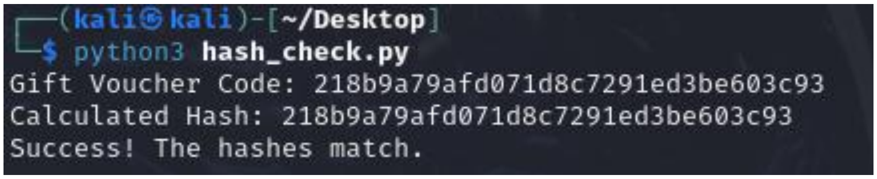
**The two-alphabet character from A = nh**

**The two-alphabet character from B = #`**

**d) Checking the hash**



```python
import hashlib

# Gift Voucher code
voucher_code = '218b9a79afd071d8c7291ed3be603c93'

# Components to be used in hash calculation
A = 'nh'                # Two-alphabet characters from set A
ClientID = '8039276'    # Example client ID
B = '#`'                # Two-symbol characters from set B

# Combine components into a single string
input_string = A + ClientID + B

# Calculate the MD5 hash of the combined string
calculated_hash = hashlib.md5(input_string.encode()).hexdigest()

# Output the results
print("Gift Voucher Code:", voucher_code)
print("Calculated Hash:", calculated_hash)

# Compare the calculated hash with the voucher code
if calculated_hash == voucher_code:
    print("Success! The hashes match.")
else:
    print("Failure! The hashes do not match.")

```



```
┌──(kali㉿kali)-[~/Desktop]
└─$ python3 hash_check.py
Gift Voucher Code: 218b9a79afd071d8c7291ed3be603c93
Calculated Hash: 218b9a79afd071d8c7291ed3be603c93
Success! The hashes match.
```

Values of A and B:

**A = nh**

**B = #`**