Faculty of Engineering and Technology

Electrical & Computer Engineering Department

ENCS4320

Applied Cryptography

Hash Length Extension Attack Lab

Prepared by:

Al-Ayham Maree 1191408

Sara Ammar 1191052

Supervised by:

Dr. Ahmad Alsadeh

Section:

2

12-2-2023

Abstract

The purpose of this lab exercise is to gain an understanding of verifying the authenticity of requests through the use of a Message Authentication Code (MAC) generated with a one-way hash and a key. Additionally, the lab will cover the Hash Length Extension attack, which allows an attacker to alter a message while still being able to produce a valid MAC based on the modified message, even without access to the secret key. The significance of padding in these types of attacks will also be emphasized. Finally, the lab will teach how to prevent the Hash Length Extension attack by using a keyed hash message authentication code (HMAC).

Contents

Abstract 2
Theory 5
    SHA-256 5
    Padding 5
    Hash Length Extension Attack 6
    HMAC 7
Procedure with results of tasks 8
    Lab Environment 8
    Task 1: Send Request to List Files 10
    Task2: Create Padding 12
    Task 3: The Length Extension Attack 13
    Task 4: Attack Mitigation using HMAC 16
Conclusion 21
References 22

Table of Figures:

Figure 1: Hash Function Example 5
Figure 2: Hash Length Extension Attack 7
Figure 3: HMAC description 7
Figure 4: Settings for utilizing a server program 8
Figure 5: Building Container Image 9
Figure 6: Starting Container Image 9
Figure 7: Calculating MAC 10
Figure 8: Server Response 11
Figure 9: Calculating MAC 11
Figure 10: Server Response 12
Figure 11: Python Code for padding 12
Figure 12: Padding Output 13
Figure 13: Calculating MAC 13
Figure 14: Padding Message 13
Figure 15: Generate New MAC code 14
Figure 16: New MAC Code after executed C code 14
Figure 17: Server Response 15
Figure 18: Lab.py file where the verify function was changed 16
Figure 19: HMAC.py file 17
Figure 20: Creating a MAC address by running HMAC.py file 17
Figure 21: The response of the request 18
Figure 22: HMAC.py file with new message 18
Figure 23: Creating a new MAC address by running HMAC.py file 19
Figure 24: The response of the new request 19

Theory

SHA-256

SHA-256 is a mathematical algorithm that converts an input of any length into a fixed-length 256-bit output known as a hash. This one-way function ensures that it is very difficult to determine the original input from the hash output. This property makes it useful for applications such as digital signatures, password protection, and ensuring the integrity of digital data. It is considered to be secure and widely used in various security-sensitive applications.

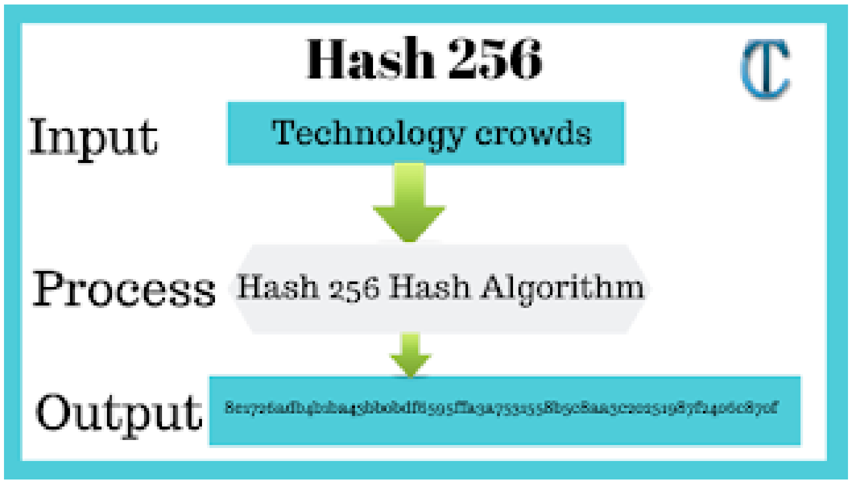


Figure 1: Hash Function Example

Notice that SHA-256 ensures the reliability of data transmission by allowing both parties to verify the authenticity of a message. The recipient device calculates a hash value of the original message and compares it to the hash value received from the sender. If the two hash values match, it confirms that the message has remained unchanged during transmission and can be trusted as coming from the intended source.

Padding

Padding is a method used in cryptography to make data processing consistent and secure. It is used to extend the length of a message or data to meet certain requirements, such as block size in block ciphers. Padding also helps conceal the true length of a message, making it harder for attackers to determine its size and contents. By adding padding, cryptographic systems can improve their security and reliability.

SHA-256 is a hash function that operates on fixed-sized blocks of data. The block size of SHA-256 is 64 bytes, so when a message is hashed with SHA-256, it is padded to a multiple of 64 bytes. This padding ensures that the message meets the required length for processing by the hash function.

According to RFC 6234, the padding consists of one byte of \x80, followed by many zeros, and then a 64-bit length field that represents the length of the original message in bits. For example, if the original message is 22 bytes long, the padding would consist of 42 bytes of zeros, followed by the 8-byte length field.

It is important to note that the length field uses big-endian byte order, meaning that the most significant byte is stored first. For example, if the length of the original message is 0x012345, the length field in the padding would be "\x00\x00\x00\x00\x00\x01\x23\x45".

In some applications, such as sending data in URLs, the hexadecimal numbers in the padding need to be encoded using percent encoding, where each hexadecimal number is replaced with its equivalent percent-encoded form. For example, "\x80" in the padding would be encoded as "%80". On the server side, the percent-encoded data would be decoded back into its binary representation.

Hash Length Extension Attack

The Hash Length Extension Attack is a security vulnerability in which an attacker can modify a message in transit and still generate a valid MAC based on the modified message, without knowing the secret key. The attack takes advantage of the way that certain hash algorithms work, such as SHA-256, by being able to extend the original message without affecting the validity of the MAC. The importance of padding in these kinds of attacks is crucial, as it can be used to mitigate the attack by making it harder for an attacker to extend the original message. One way to mitigate this attack is by using HMAC, which is a keyed hash message authentication code.

An example about the Hash Length Extension Attack:

Suppose the original message is "This is a test message" and the secret key is "secret". The sender computes the SHA-256 hash of the message and the key to generate the MAC, which is a unique identifier for the message. The message, the MAC, and the key are then sent to the recipient.

An attacker intercepts the message in transit and modifies it to "This is a test message, but I changed it". The attacker also computes the SHA-256 hash of the modified message and the key, but since the attacker doesn't know the secret key, they can't generate a valid MAC for the modified message.

However, the attacker can take advantage of the way that the SHA-256 algorithm works and extend the original message to include the additional text, "but I changed it". This allows the attacker to generate a valid MAC for the modified message, even though the message has been tampered with. The recipient would receive the modified message and the attacker-generated MAC, and since the MAC is valid, the recipient would believe that the message came from the sender and has not been tampered with.

This is where padding becomes important, as it can be used to mitigate the attack by making it harder for an attacker to extend the original message. By using HMAC, which is a keyed hash message authentication code, the attack can be further mitigated, as the secret key is used in the hash calculation, making it much harder for an attacker to generate a valid MAC.

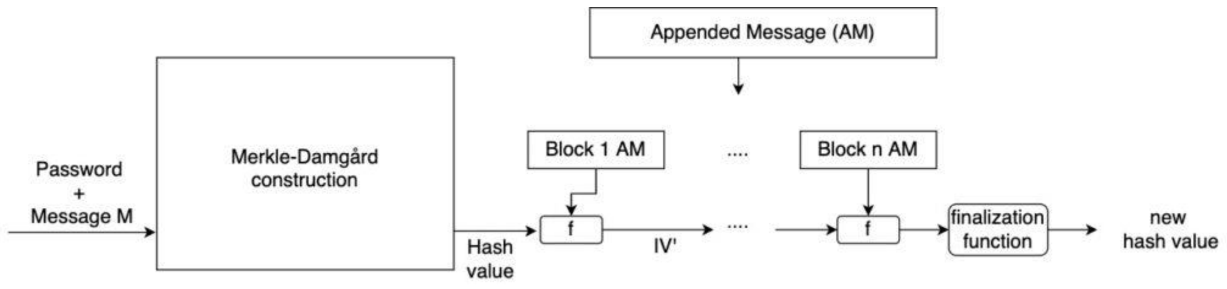


Figure 2: Hash Length Extension Attack

HMAC

HMAC (Hash-based Message Authentication Code) is a mechanism used to authenticate the integrity of a message in a communication between two parties. It is a type of keyed hash function that combines a secret key with a hash function to produce a unique message authentication code. The recipient of the message can then use the same secret key to calculate the HMAC of the received message and compare it with the one sent by the sender. If the calculated HMAC matches the sent HMAC, it proves that the message has not been altered during transmission and provides assurance that the message actually came from the sender. HMAC is more secure than other hash-based authentication methods because it involves a secret key, which makes it harder for an attacker to impersonate the sender or alter the message.

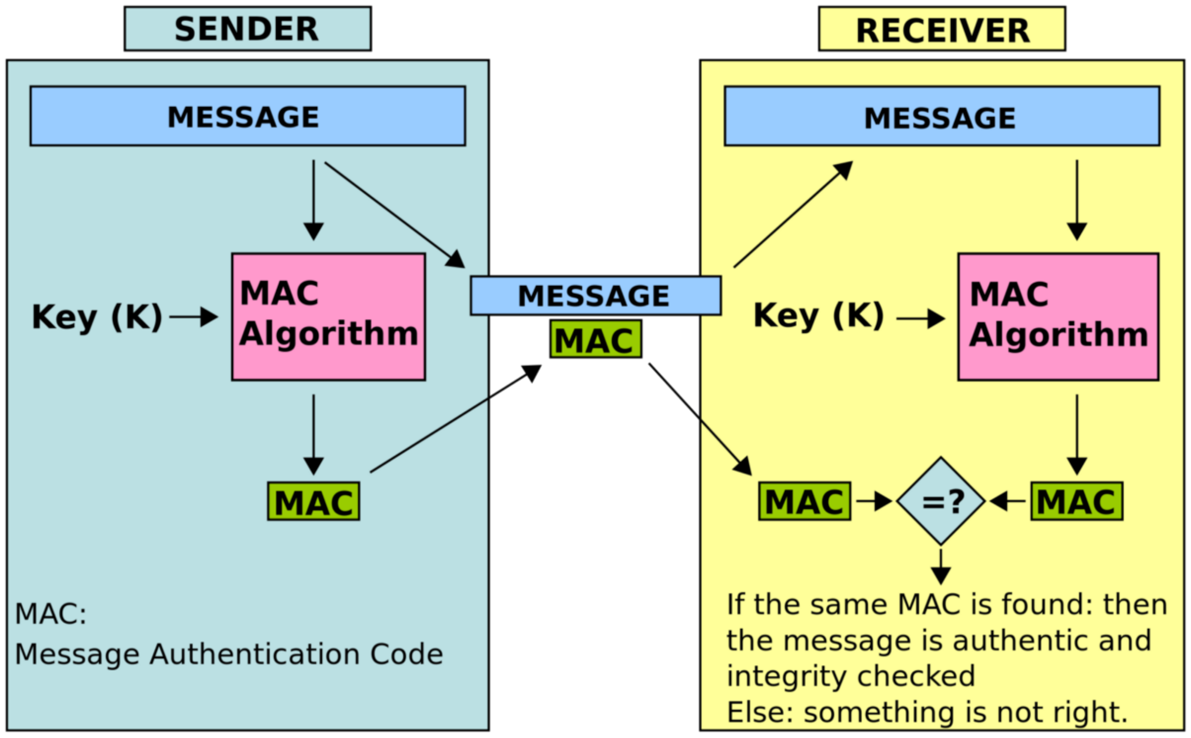
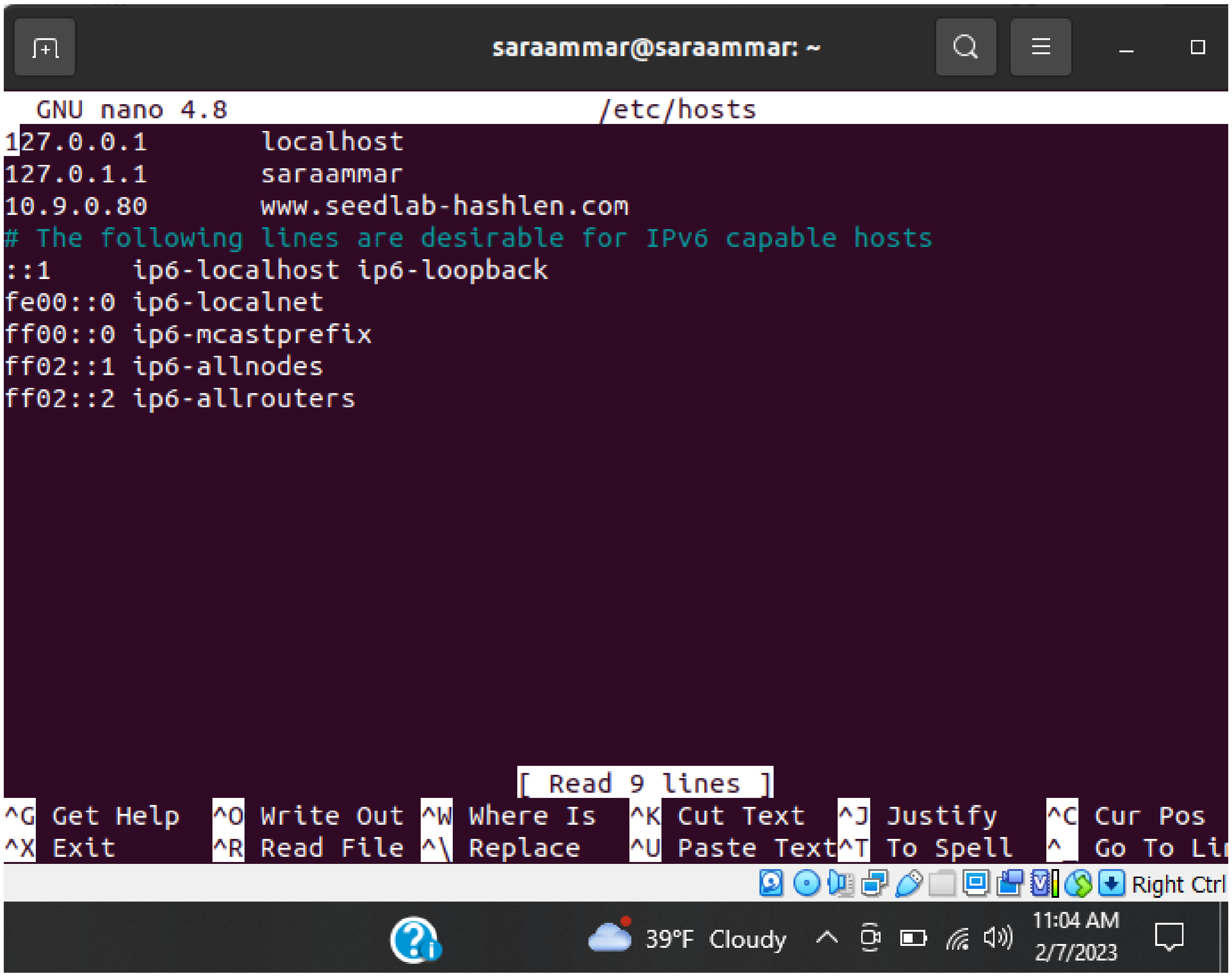

SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Figure 3: HMAC description

Procedure with results of tasks

Lab Environment

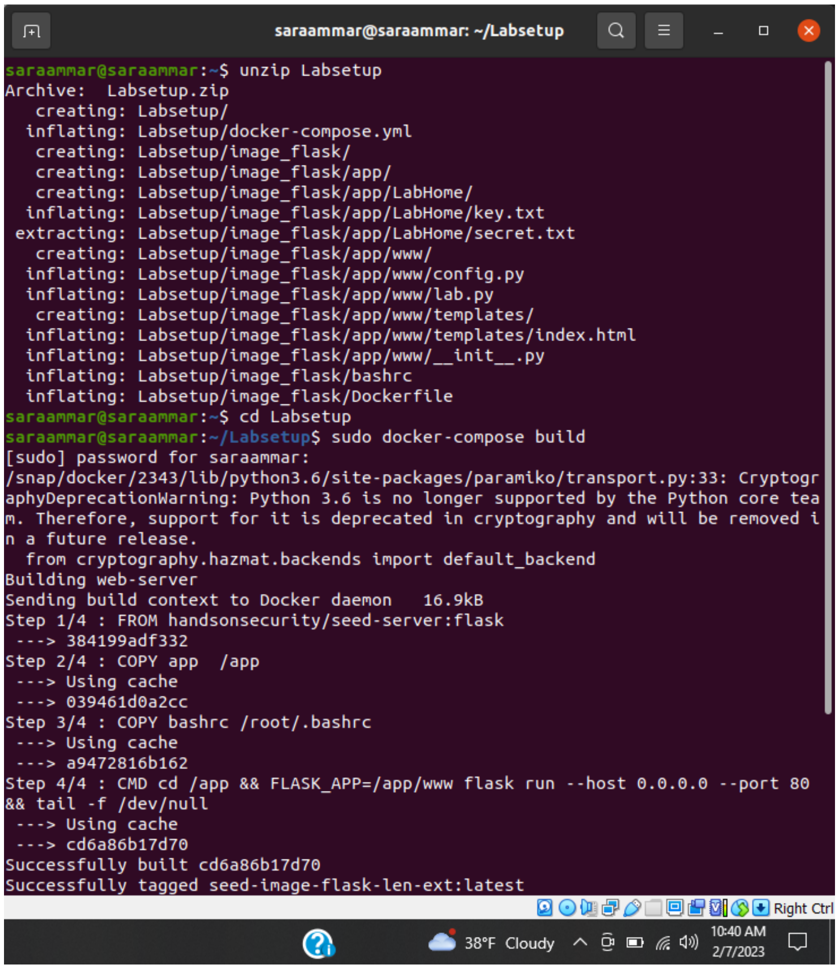
The website "www.seedlab-hashlen.com" was utilized for hosting a server program. This hostname was linked to the web server container located at "10.9.0.80" by adding an entry in the "/etc/hosts" file, as shown in Figure 4 below:



```
saraammar@saraammar: ~
  GNU nano 4.8                          /etc/hosts
127.0.0.1       localhost
127.0.1.1       saraammar
10.9.0.80       www.seedlab-hashlen.com
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters




                                [ Read 9 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text^T To Spell    ^_ Go To Li
```

Figure 4: Settings for utilizing a server program

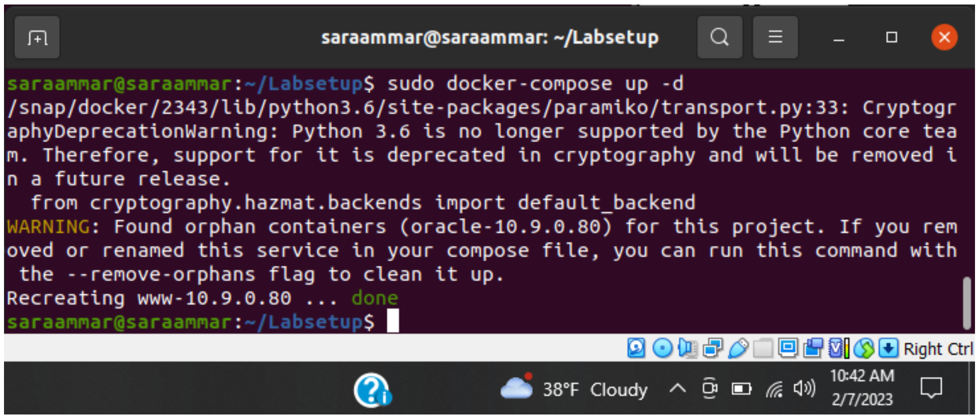
The container image was then created using the command: "$ sudo docker-compose build", as illustrated in Figure 5 below:



```
saraammar@saraammar:~$ unzip Labsetup
Archive:  Labsetup.zip
   creating: Labsetup/
  inflating: Labsetup/docker-compose.yml
   creating: Labsetup/image_flask/
   creating: Labsetup/image_flask/app/
   creating: Labsetup/image_flask/app/LabHome/
  inflating: Labsetup/image_flask/app/LabHome/key.txt
 extracting: Labsetup/image_flask/app/LabHome/secret.txt
   creating: Labsetup/image_flask/app/www/
  inflating: Labsetup/image_flask/app/www/config.py
  inflating: Labsetup/image_flask/app/www/lab.py
   creating: Labsetup/image_flask/app/www/templates/
  inflating: Labsetup/image_flask/app/www/templates/index.html
  inflating: Labsetup/image_flask/app/www/__init__.py
  inflating: Labsetup/image_flask/bashrc
  inflating: Labsetup/image_flask/Dockerfile
saraammar@saraammar:~$ cd Labsetup
saraammar@saraammar:~/Labsetup$ sudo docker-compose build
[sudo] password for saraammar:
/snap/docker/2343/lib/python3.6/site-packages/paramiko/transport.py:33: Cryptogr
aphyDeprecationWarning: Python 3.6 is no longer supported by the Python core tea
m. Therefore, support for it is deprecated in cryptography and will be removed i
n a future release.
  from cryptography.hazmat.backends import default_backend
Building web-server
Sending build context to Docker daemon   16.9kB
Step 1/4 : FROM handsonsecurity/seed-server:flask
 ---> 384199adf332
Step 2/4 : COPY app  /app
 ---> Using cache
 ---> 039461d0a2cc
Step 3/4 : COPY bashrc /root/.bashrc
 ---> Using cache
 ---> a9472816b162
Step 4/4 : CMD cd /app && FLASK_APP=/app/www flask run --host 0.0.0.0 --port 80
&& tail -f /dev/null
 ---> Using cache
 ---> cd6a86b17d70
Successfully built cd6a86b17d70
Successfully tagged seed-image-flask-len-ext:latest
```

Figure 5: Building Container Image

The container image was started using the command: "$ sudo docker-compose up -d", as illustrated in Figure 6 below:



```
saraammar@saraammar:~/Labsetup$ sudo docker-compose up -d
/snap/docker/2343/lib/python3.6/site-packages/paramiko/transport.py:33: Cryptogr
aphyDeprecationWarning: Python 3.6 is no longer supported by the Python core tea
m. Therefore, support for it is deprecated in cryptography and will be removed i
n a future release.
  from cryptography.hazmat.backends import default_backend
WARNING: Found orphan containers (oracle-10.9.0.80) for this project. If you rem
oved or renamed this service in your compose file, you can run this command with
 the --remove-orphans flag to clean it up.
Recreating www-10.9.0.80 ... done
saraammar@saraammar:~/Labsetup$
```

Figure 6: Starting Container Image

Task 1: Send Request to List Files

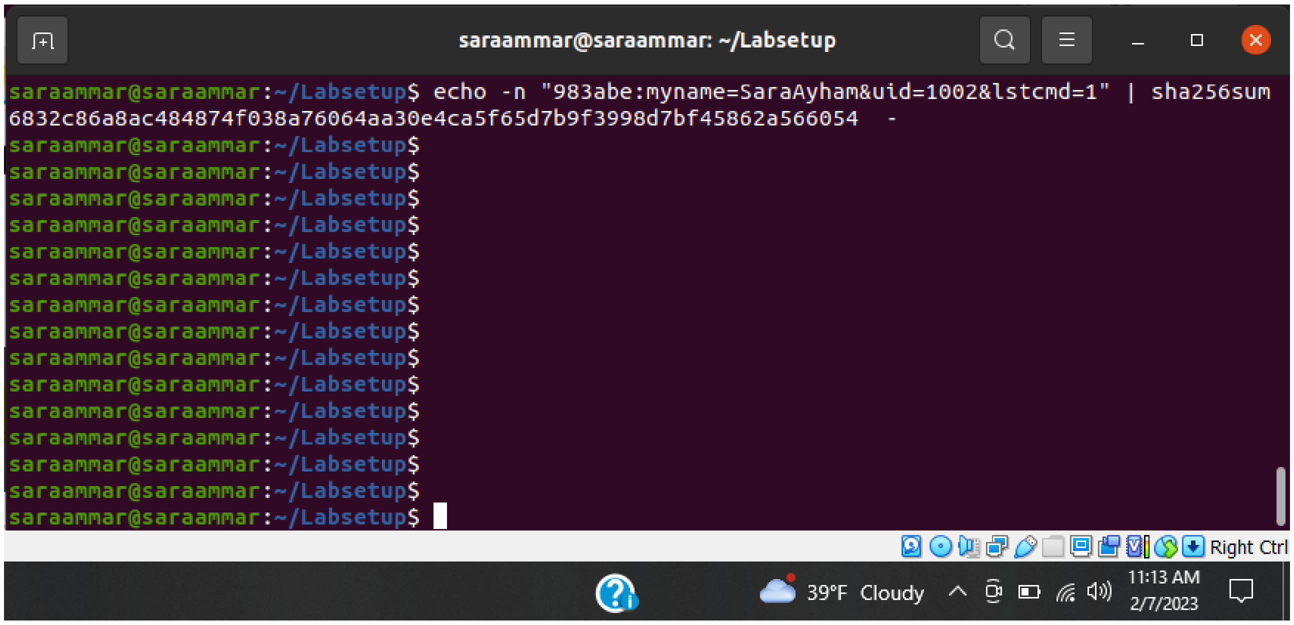
A request was sent to the domain www.seedlab-hashlen.com, which was hosted on a web server container (10.9.0.80) with a mapped hostname in the virtual machine. The container was built using the docker-compose build command, as shown in Figure 5. The request was made with the following format:

http://www.seedlab-hashlen.com/?myname=<name>&uid=<need-to-fill-> &lstcmd=1&mac=<need-tocalculate>

The user id used was 1002 and the key for this user was 983abe, with the name SaraAyham. To calculate the MAC, the following command was used:

$ echo -n "983abe:myname=SaraAyham&uid=1002&lstcmd=1" | sha256sum

As illustrated in Figure7 below:



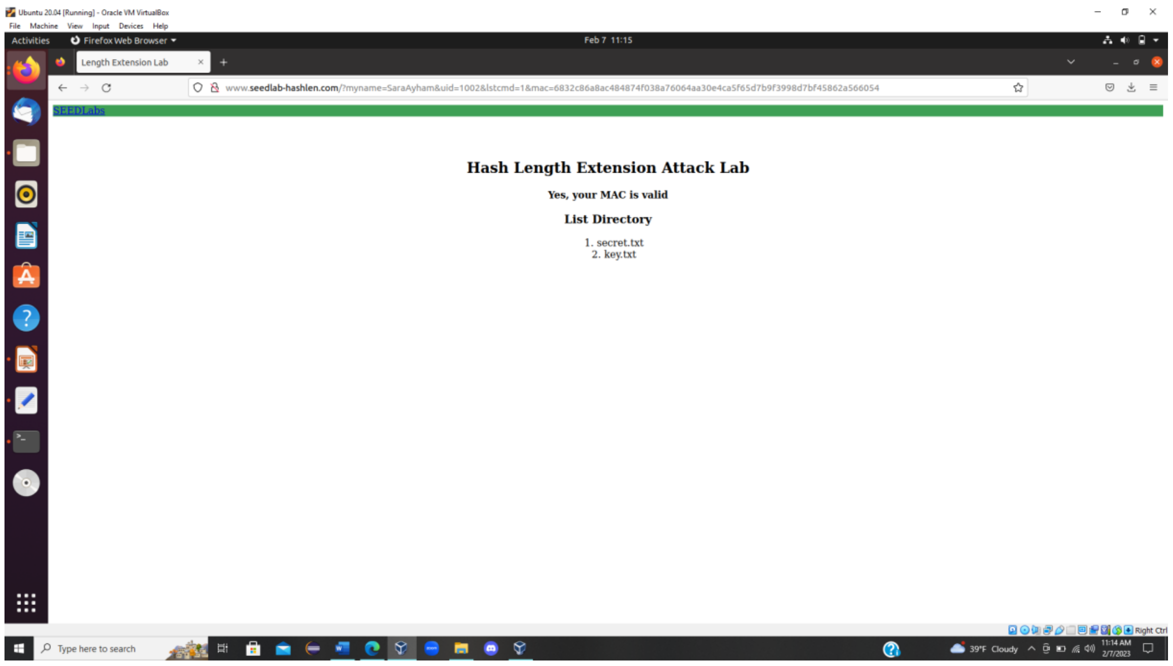
```
saraammar@saraammar:~/Labsetup$ echo -n "983abe:myname=SaraAyham&uid=1002&lstcmd=1" | sha256sum
6832c86a8ac484874f038a76064aa30e4ca5f65d7b9f3998d7bf45862a566054  -
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
```

Figure 7: Calculating MAC

The following URL was constructed to send the request:

http://www.seedlabhashlen.com/?myname=SaraAyham&uid=1002&lstcmd=1&mac=6832c86a8ac484874f028a76064aa30e4ca5f65d7b9f3998d7bf45862a566053

The request was sent using Firefox, and the response is displayed in Figure 8 below



```
www.seedlab-hashlen.com/?myname=SaraAyham&uid=1002&lstcmd=1&mac=6832c86a8ac484874f038a76064aa30e4ca5f65d7b9f3998d7bf45862a566054
```

SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

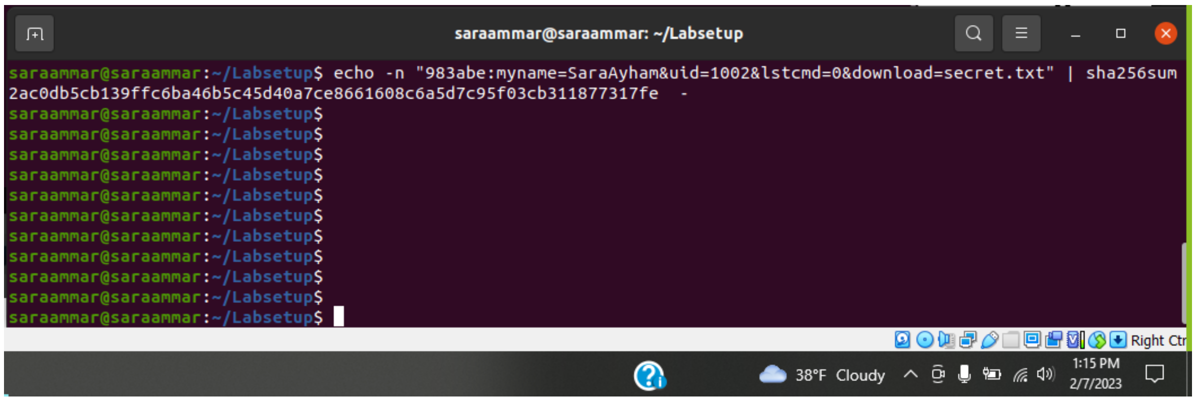
1. secret.txt
2. key.txt

Figure 8: Server Response

The next step was to send a download command to the server. To do this, the MAC was calculated by executing the following command:

$ echo -n "983abe:myname=SaraAyham&uid=1002& &lstcmd=0&download=secret.txt" |sha256sum

This command was used as shown in figure 9 below:



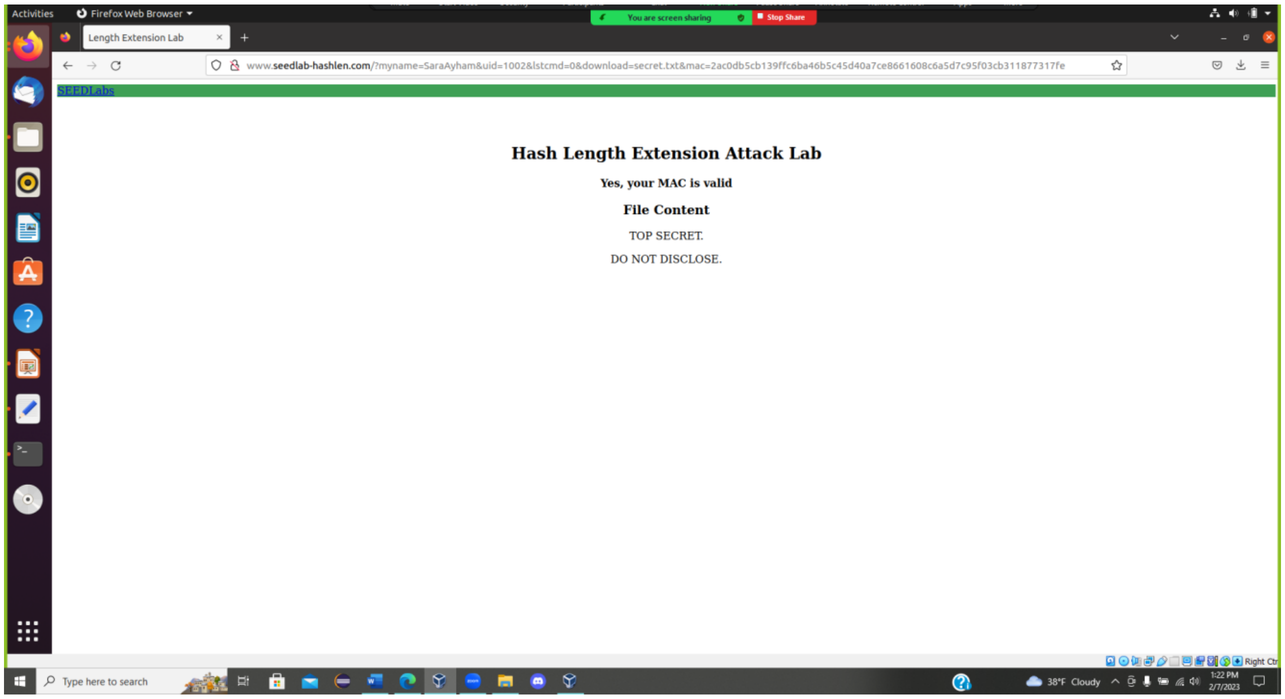
```
saraammar@saraammar:~/Labsetup$ echo -n "983abe:myname=SaraAyham&uid=1002&lstcmd=0&download=secret.txt" | sha256sum
2ac0db5cb139ffc6ba46b5c45d40a7ce8661608c6a5d7c95f03cb311877317fe  -
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
saraammar@saraammar:~/Labsetup$
```

Figure 9: Calculating MAC

The following URL was constructed to send the request:

http://www.seedlabhashlen.com/?myname=SaraAyham&uid=1002&lstcmd=0&download=secret.txt&mac=2ac0db5cb139ffc6ba46bSc45d0a7ce8661608c6a5d7c95f03cb311877317fe

The request was sent using Firefox, and the response is displayed in Figure 10 below


www.seedlab-hashlen.com/?myname=SaraAyham&uid=1002&lstcmd=0&download=secret.txt&mac=2ac0db5cb139ffc6ba46b5c45d40a7ce8661608c6a5d7c95f03cb311877317fe

SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

File Content

TOP SECRET.

DO NOT DISCLOSE.

Figure 10: Server Response

As demonstrated by the two requests, since the MAC's for both requests were deemed valid by the server, it responded and carried out the request, which in the first instance was a list request and in the second, a download request.

Task2: Create Padding

The creation of a python program was accomplished for the purpose of adding padding to messages, in accordance with the specified algorithm. The padding method for SHA256 involves including one byte of \x80, followed by a series of zeros, and finally a 64-bit (8 bytes) length field, which represents the number of bits in the message (M). This can be seen in Figure 11 below:

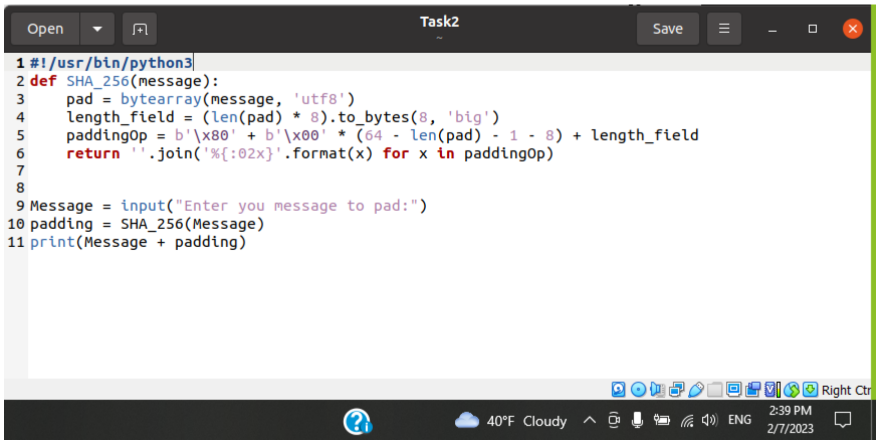
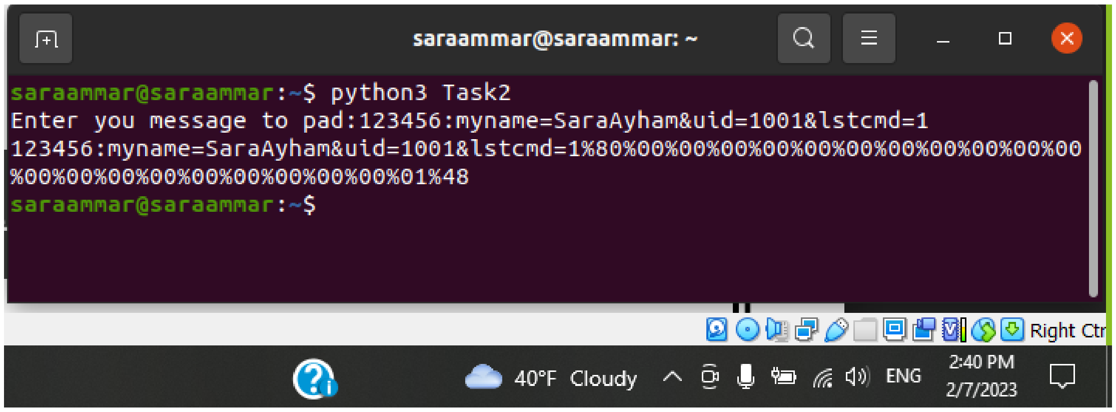

```python
#!/usr/bin/python3
def SHA_256(message):
    pad = bytearray(message, 'utf8')
    length_field = (len(pad) * 8).to_bytes(8, 'big')
    paddingOp = b'\x80' + b'\x00' * (64 - len(pad) - 1 - 8) + length_field
    return ''.join('%{:02x}'.format(x) for x in paddingOp)


Message = input("Enter you message to pad:")
padding = SHA_256(Message)
print(Message + padding)
```

Figure 11: Python Code for padding

In the code above, it’s important to note that in the URL, all hexadecimal numbers in the padding must be encoded by converting \x to % in line 6

The python code above was utilized to create padding for the message:

"123456:myname=SaraAyham&uid=1001&lstcmd=1", as demonstrated in Figure 12 below:



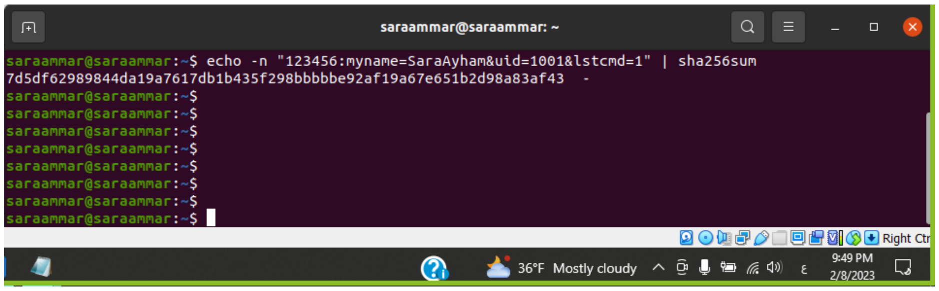
```
saraammar@saraammar:~$ python3 Task2
Enter you message to pad:123456:myname=SaraAyham&uid=1001&lstcmd=1
123456:myname=SaraAyham&uid=1001&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00
%00%00%00%00%00%00%00%00%00%01%48
saraammar@saraammar:~$
```

Figure 12: Padding Output

Task 3: The Length Extension Attack

The task of generating a valid MAC for a URL without knowing the MAC key was carried out. This was achieved by having a known MAC of a valid request R and knowing the size of the MAC key. To achieve this, the following steps were taken:

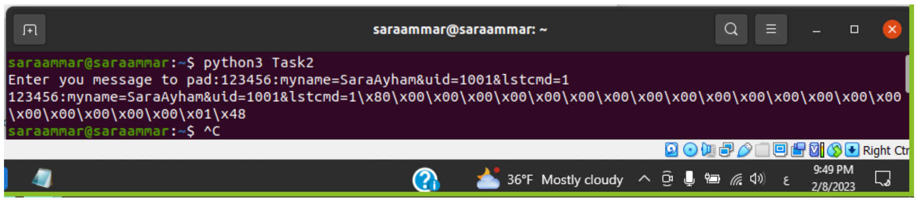
1. Generating the MAC for the message "123456:myname=SaraAyham&uid=1001&lstcmd=1" as shown in Orange in Figure 13 below:



```
saraammar@saraammar:~$ echo -n "123456:myname=SaraAyham&uid=1001&lstcmd=1" | sha256sum
7d5df62989844da19a7617db1b435f298bbbbbe92af19a67e651b2d98a83af43  -
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
```

Figure 13: Calculating MAC

2. Creating a padding for the same message, which is shown in yellow in Figure 14 below:

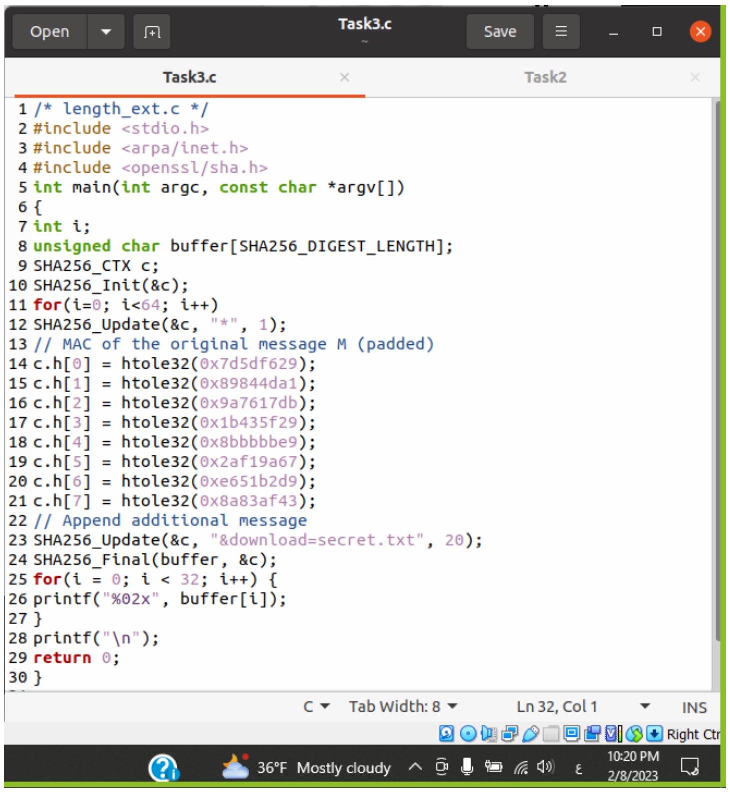


```
saraammar@saraammar:~$ python3 Task2
Enter you message to pad:123456:myname=SaraAyham&uid=1001&lstcmd=1
123456:myname=SaraAyham&uid=1001&lstcmd=1\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x01\x48
saraammar@saraammar:~$ ^C
```

Figure 14: Padding Message

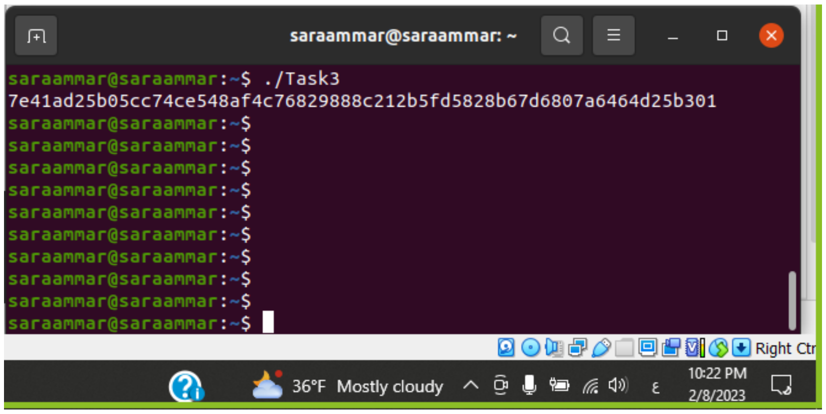
Notice that The MAC key was obtained from the file "LabHome/key.txt". The goal was to forge a new request based on the valid request R, while still being able to compute the valid MAC.

The next step was to generate a new MAC for a modified message using the original MAC. This was done by using the C code provided in the manual, as depicted in Figure 15 below:



```
/* length_ext.c */
#include <stdio.h>
#include <arpa/inet.h>
#include <openssl/sha.h>
int main(int argc, const char *argv[])
{
int i;
unsigned char buffer[SHA256_DIGEST_LENGTH];
SHA256_CTX c;
SHA256_Init(&c);
for(i=0; i<64; i++)
SHA256_Update(&c, "*", 1);
// MAC of the original message M (padded)
c.h[0] = htole32(0x7d5df629);
c.h[1] = htole32(0x89844da1);
c.h[2] = htole32(0x9a7617db);
c.h[3] = htole32(0x1b435f29);
c.h[4] = htole32(0x8bbbbbe9);
c.h[5] = htole32(0x2af19a67);
c.h[6] = htole32(0xe651b2d9);
c.h[7] = htole32(0x8a83af43);
// 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 15: Generate New MAC code



```
saraammar@saraammar:~$ ./Task3
7e41ad25b05cc74ce548af4c76829888c212b5fd5828b67d6807a6464d25b301
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
```

Figure 16: New MAC Code after executed C code

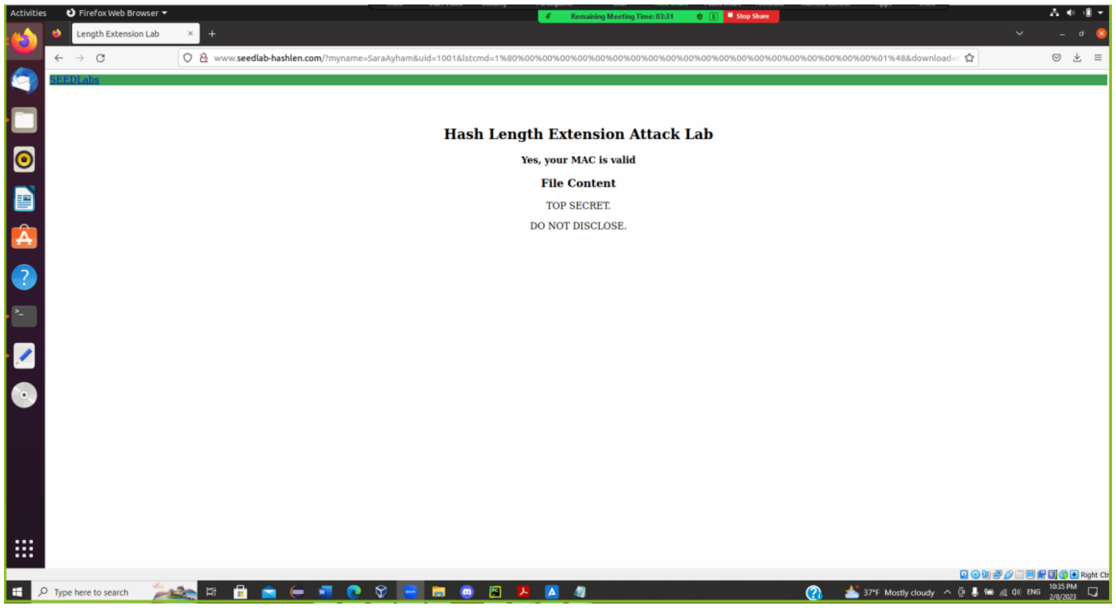
The C code was executed. Then, a new request was formulated using the following format:

http://www.seedlabhashlen.com/?myname=<name>&uid=<uid>&lstcmd=1<padding>&download=secret.txt&mac=<newmac>

With the obtained values of name, uid, padding, and the new MAC, a new request was constructed as follows:

http://www.seedlabhashlen.com/?myname=SaraAyham&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%00%00%00%00%00%00%01%48&downlad=secret.txt&mac=7e41ad25b05cc74ce548af4c76829888c212b5fd5828b67d6807a6464d25b301

The response to this request can be seen in Figure 16 below:


Activities
Firefox Web Browser
Length Extension Lab
www.seedlab-hashlen.com/?myname=SaraAyham&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%00%00%00%00%00%00%01%48&download=
SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

File Content

TOP SECRET.

DO NOT DISCLOSE.

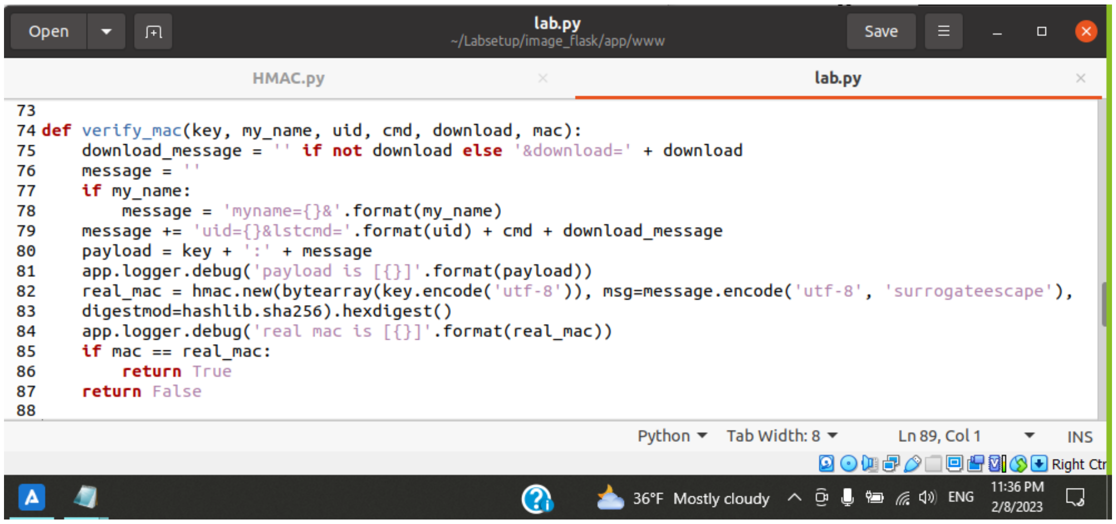
Figure 17: Server Response

As demonstrated in Figure 17, a process was carried out to successfully generate a valid MAC for a URL without having the knowledge of the MAC key. Using the MAC of a valid request and the size of the MAC key, a new request was constructed and the valid MAC was computed.

Task 4: Attack Mitigation using HMAC

We have seen the harm that results when a developer computes a MAC in an unsafe manner by concatenating the key and the message in the tasks thus far. In this task, we'll correct the developer's error. HMAC is the formula typically used to compute MACs. To determine the MAC address using Python's hmac function The verify mac() method was changed in the server program. The lab.py file contains the function. A key and message (both of type string) were provided, and the HMAC was calculated as follows:

```
real_mac = hmac.new(bytearray(key.encode('utf-8')), msg=message.encode('utf-8' 'surrogateescape'),
digestmod=hashlib.sha256).hexdigest()
```

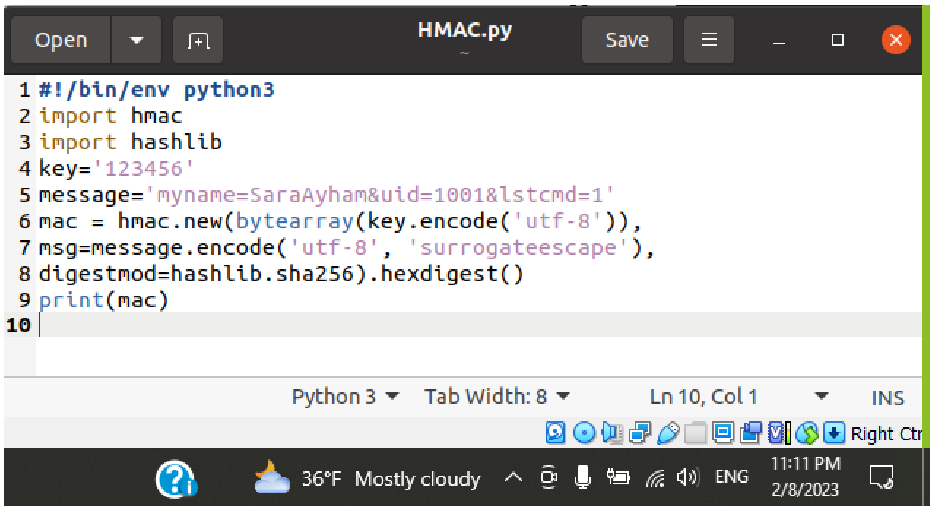


```
def verify_mac(key, my_name, uid, cmd, download, mac):
    download_message = '' if not download else '&download=' + download
    message = ''
    if my_name:
        message = 'myname={}&'.format(my_name)
    message += 'uid={}&lstcmd='.format(uid) + cmd + download_message
    payload = key + ':' + 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 18: Lab.py file where the verify function was changed

Following the modifications, all of the containers were stopped using the command "sudo docker-compose down", rebuilt using the command "sudo docker-compose build", and restarted using the command "sudo docker-compose up -d". Then the modification became effective.

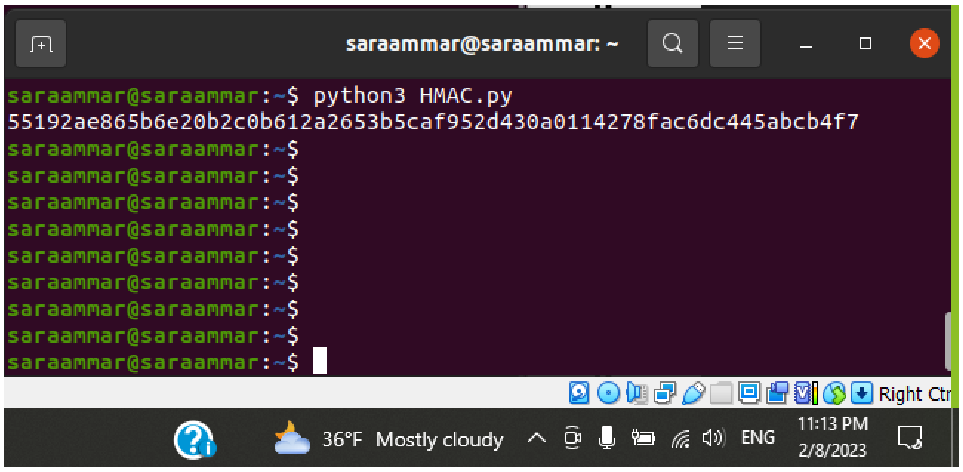
Then, using HMAC for the MAC computation, Task 1 was performed to make a request to list files. The following program in HMAC.py file was used to compute the MAC for the message "myname=SaraAyham&uid=1001&lstcmd=1 " assuming that the selected key is 123456:



```python
#!/bin/env python3
import hmac
import hashlib
key='123456'
message='myname=SaraAyham&uid=1001&lstcmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),
msg=message.encode('utf-8', 'surrogateescape'),
digestmod=hashlib.sha256).hexdigest()
print(mac)

```

Figure 19: HMAC.py file

The code was then run, and Figure-20 shows the MAC address that was created:



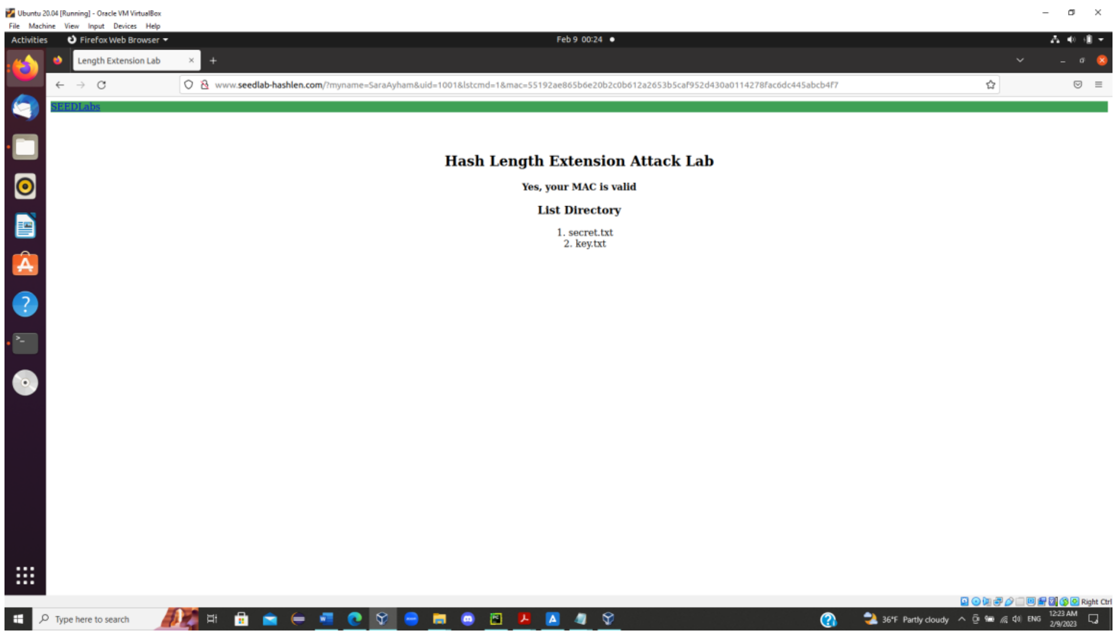
```
saraammar@saraammar:~$ python3 HMAC.py
55192ae865b6e20b2c0b612a2653b5caf952d430a0114278fac6dc445abcb4f7
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
```

Figure 20: Creating a MAC address by running HMAC.py file

Then the next request was created:

http://www.seedlab-hashlen.com/?myname=SaraAyham&uid=1001&lstcmd=1&mac=55192ae865b6e20b2c0b612a2653b5caf952d430a0114278fac6dc445abcb4f7

And its response was as follows:



```
www.seedlab-hashlen.com/?myname=SaraAyham&uid=1001&lstcmd=1&mac=55192ae865b6e20b2c0b612a2653b5caf952d430a0114278fac6dc445abcb4f7
```

SEEDLabs

Hash Length Extension Attack Lab

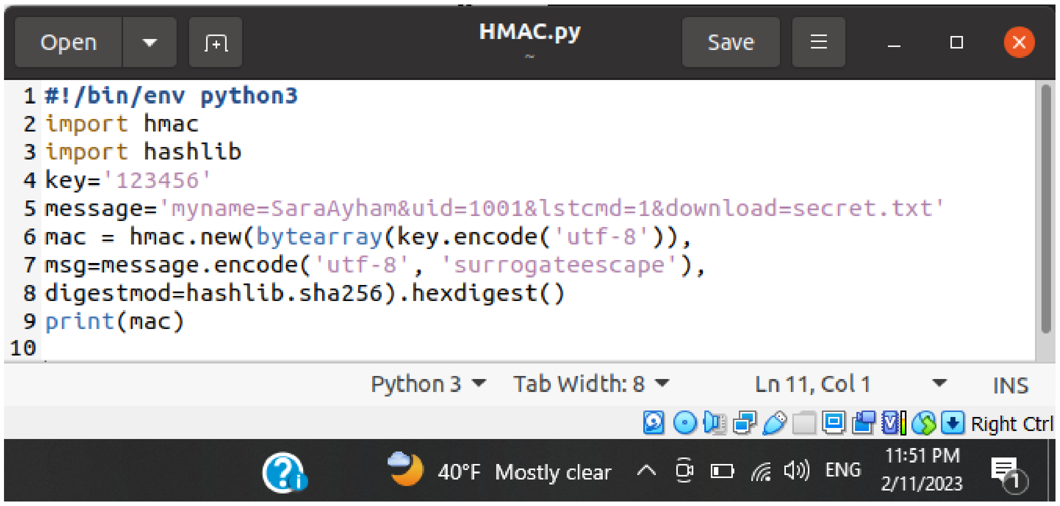
Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

Figure 21: The response of the request

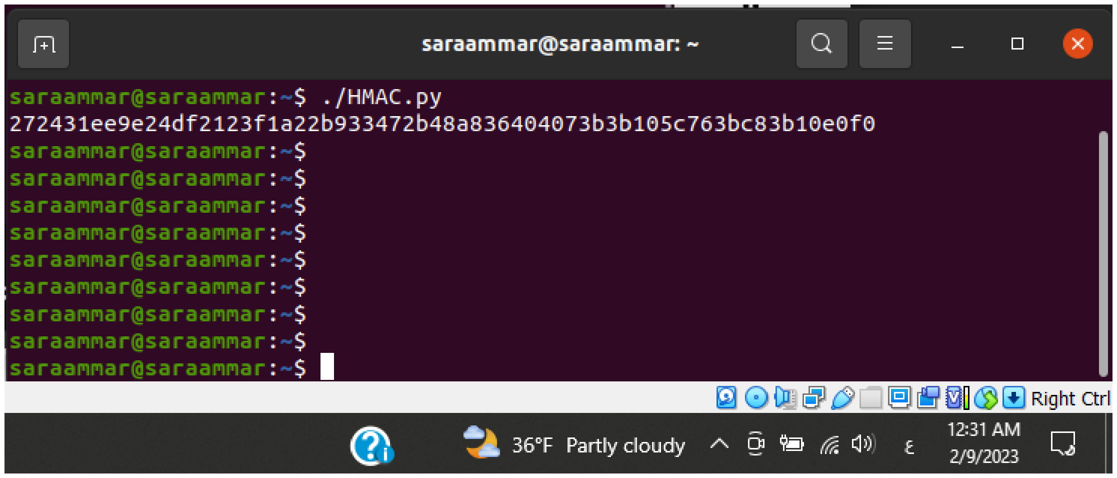
After that the program in HMAC.py file was used to compute the MAC for a new message "myname=SaraAyham&uid=1001&lstcmd=1&download=secret.txt":



```python
#!/bin/env python3
import hmac
import hashlib
key='123456'
message='myname=SaraAyham&uid=1001&lstcmd=1&download=secret.txt'
mac = hmac.new(bytearray(key.encode('utf-8')),
msg=message.encode('utf-8', 'surrogateescape'),
digestmod=hashlib.sha256).hexdigest()
print(mac)
```

Figure 22: HMAC.py file with new message

The code was then run, and Figure-22 shows the new MAC that was created:



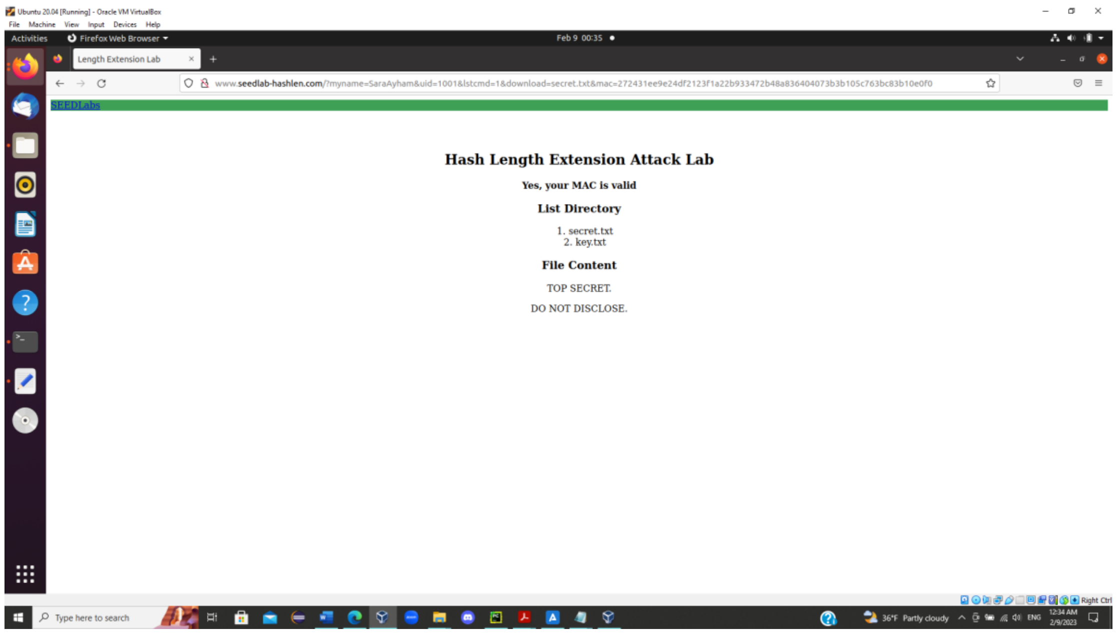
```
saraammar@saraammar:~$ ./HMAC.py
272431ee9e24df2123f1a22b933472b48a836404073b3b105c763bc83b10e0f0
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$
```

Figure 23: Creating a new MAC address by running HMAC.py file

Then the next request was created for the new message:

http://www.seedlab-hashlen.com/?myname=SaraAyham&uid=1001&lstcmd=1&download=secret.txt&mac=272431ee9e24df2123f1a22b933472b48a836404073b3b105c763bc83b10e0f0

And the response for the new message was as follows:


Ubuntu 20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
Firefox Web Browser
Feb 9 00:35
Length Extension Lab
www.seedlab-hashlen.com/?myname=SaraAyham&uid=1001&lstcmd=1&download=secret.txt&mac=272431ee9e24df2123f1a22b933472b48a836404073b3b105c763bc83b10e0f0
SEEDLabs
Type here to search
36°F Partly cloudy
Right Ctrl

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

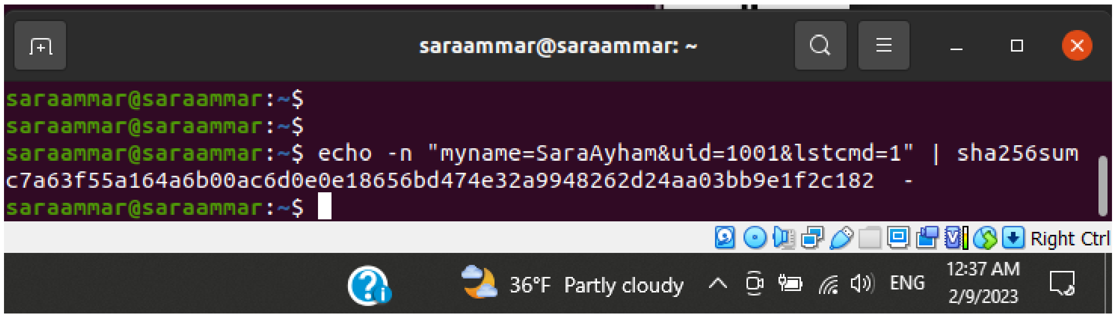
File Content

TOP SECRET.

DO NOT DISCLOSE.

Figure 24: The response of the new request

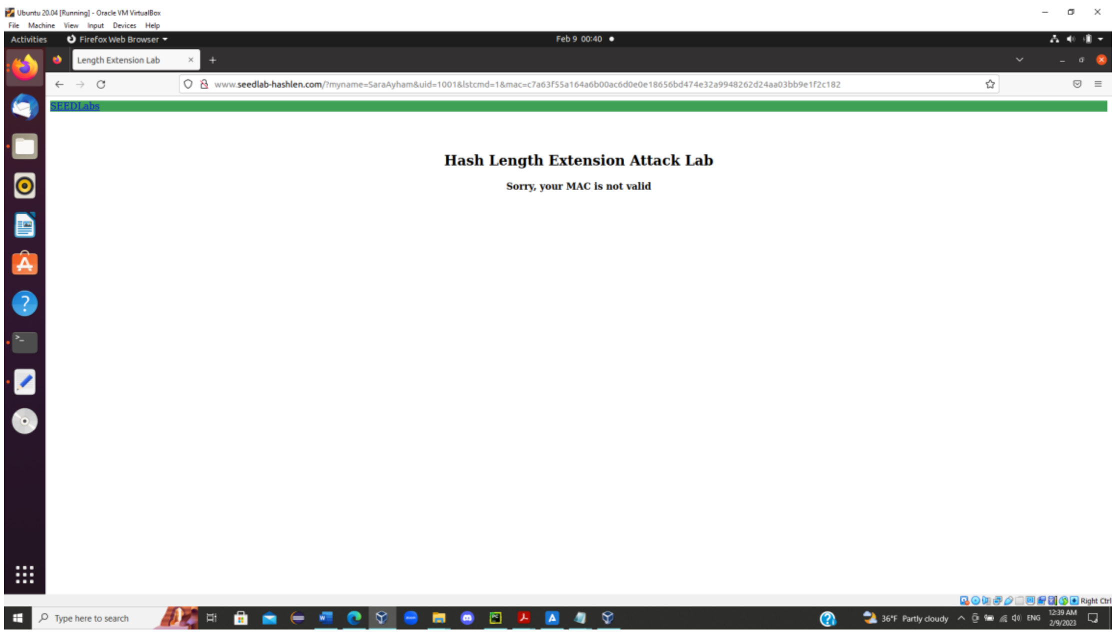
The MAC was then created using SHA256, similar to Task 1:



```
saraammar@saraammar:~$
saraammar@saraammar:~$
saraammar@saraammar:~$ echo -n "myname=SaraAyham&uid=1001&lstcmd=1" | sha256sum
c7a63f55a164a6b00ac6d0e0e18656bd474e32a9948262d24aa03bb9e1f2c182  -
saraammar@saraammar:~$
```

Kk

After that a new request was sent:



```
www.seedlab-hashlen.com/?myname=SaraAyham&uid=1001&lstcmd=1&mac=c7a63f55a164a6b00ac6d0e0e18656bd474e32a9948262d24aa03bb9e1f2c182
```

SEEDLabs

Hash Length Extension Attack Lab

Sorry, your MAC is not valid

Given that the server response indicates that the MAC is invalid, the Hash Length Extension attack will fail since HMAC is secure against such attacks and verifies integrity.

Conclusion

To summarize, we discovered the concept of MAC (Message Authentication Code) and its role in ensuring message integrity. We became familiar with the process of generating MACs through one-way hash functions such as SHA256. Additionally, we explored the Hash Extension attack on SHA256 and how it can be executed by manipulating padding, the original MAC, and creating a new MAC without knowledge of the key. Finally, we gained an understanding of HMAC (Hashed Message Authentication Code) and how it provides a stronger and more secure defense against such attacks.

References

1. https://manhhomienbienthuy.github.io/2015/09/30/hash-length-extension-attacks.html
2. https://www.technologycrowds.com/2019/10/compute-sha-256-hash-using-csharp-for-effective-secruity.html
3. https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Hash_Length_Ext/