**HASH EXTENSION SEED LAB**

**Submitted by:**

**Abdullah Altassan**

**Id:**

**201969370**

**Ics 344 section 3**

[Lab environment setup: 3](#_Toc25454)

[Steps 3](#_Toc31709)

[Task 1: Create and send a harmless request to the server: 4](#_Toc31369)

[Steps 4](#_Toc19085)

[Task 2: Create a complement for a specific message: 5](#_Toc19100)

[Step 5](#_Toc15760)

[1. Use Python3 to calculate message padding using following commands as shown below: 5](#_Toc254)

[6](#_Toc16578)

[Task 3: Compile and run “calculate\_mac.c”: 6](#_Toc1656)

[Steps 6](#_Toc103)

[1. Write the program calculate\_mac.c: 6](#_Toc19438)

[6](#_Toc2049)

[2. Compile the program in C: 6](#_Toc20802)

[3. Run the compiled program: 7](#_Toc12484)

[Task 4: Create a query with an unknown MAC key: 7](#_Toc1375)

[Steps 7](#_Toc32598)

[Task 5: Understanding HMAC: 9](#_Toc30345)

[Hashed Message Authentication Code: 9](#_Toc15451)

[Steps 9](#_Toc6291)

[Conclusion for HMAC 10](#_Toc11024)

# Lab environment setup:

In this phase, we configure the lab environment by downloading the necessary files and configuring the Docker container. These files contain the resources and tools necessary to perform cryptographic hash length expansion attacks. Docker provides a consistent and isolated environment for running experiments without impacting the host system.

## Steps

1. Download the lab configuration files: https://seedsecuritylabs.org/Labs\_20.04/Files/Crypto\_Hash\_Length\_Ext/Labsetup.zip -o Labsetup.zip
2. Unzip the downloaded file:

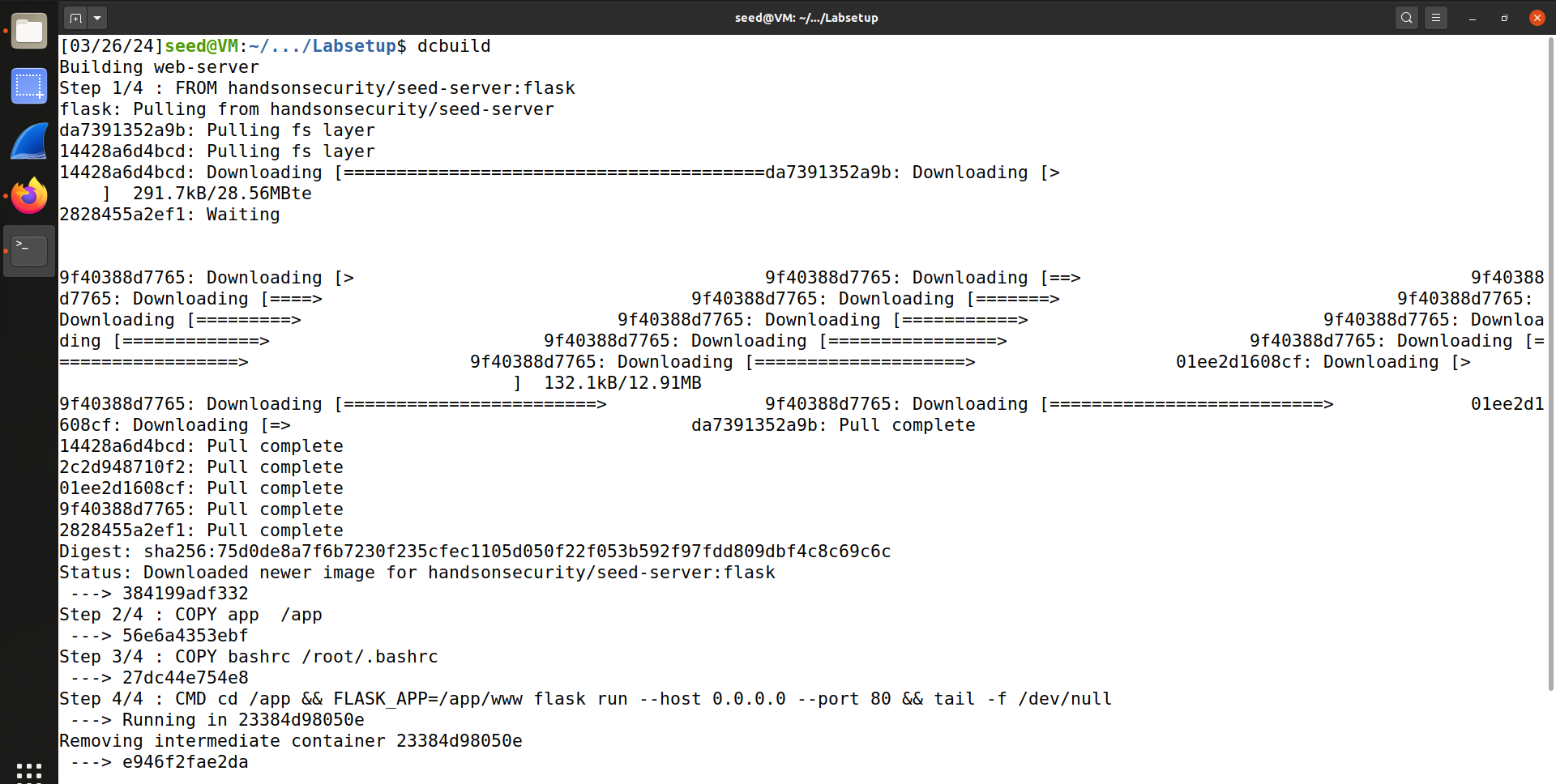
Unpack Labsetup.zip

1. Go to the Lab setup directory:

Laboratory setup for CDs

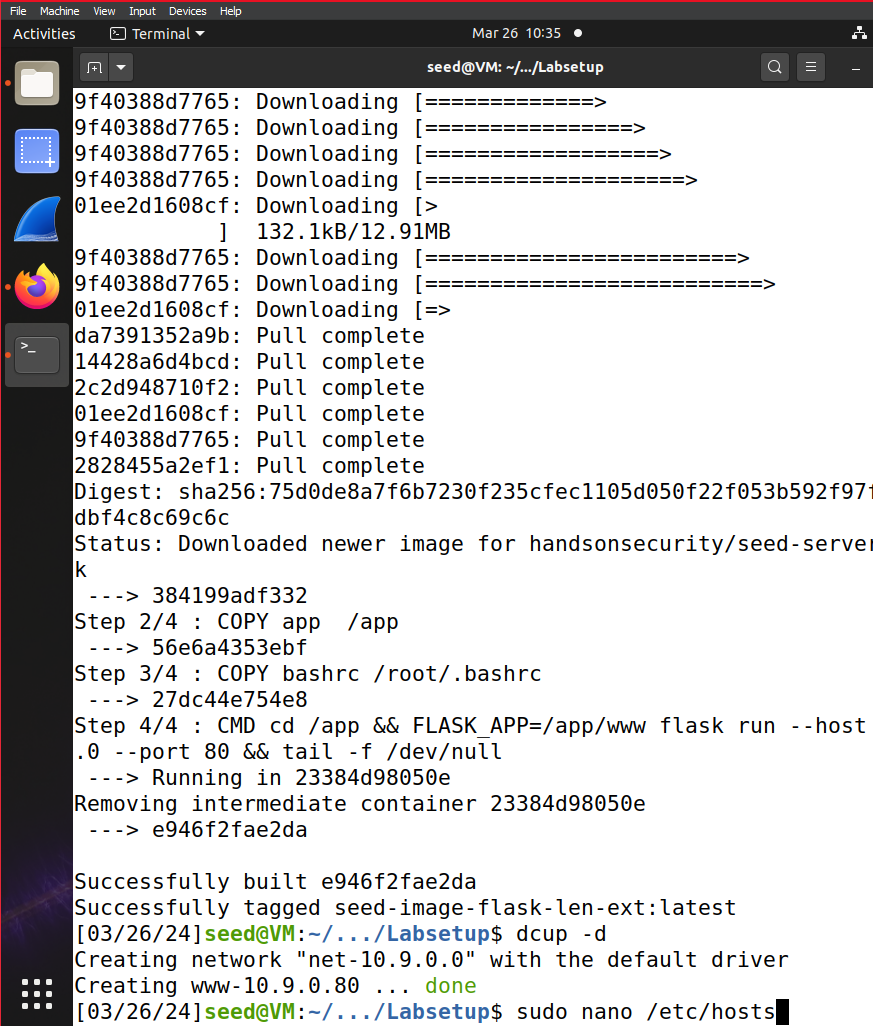
1. Create a Docker container:

dcbuild



1. Run a Docker container in the background:

dcup -d

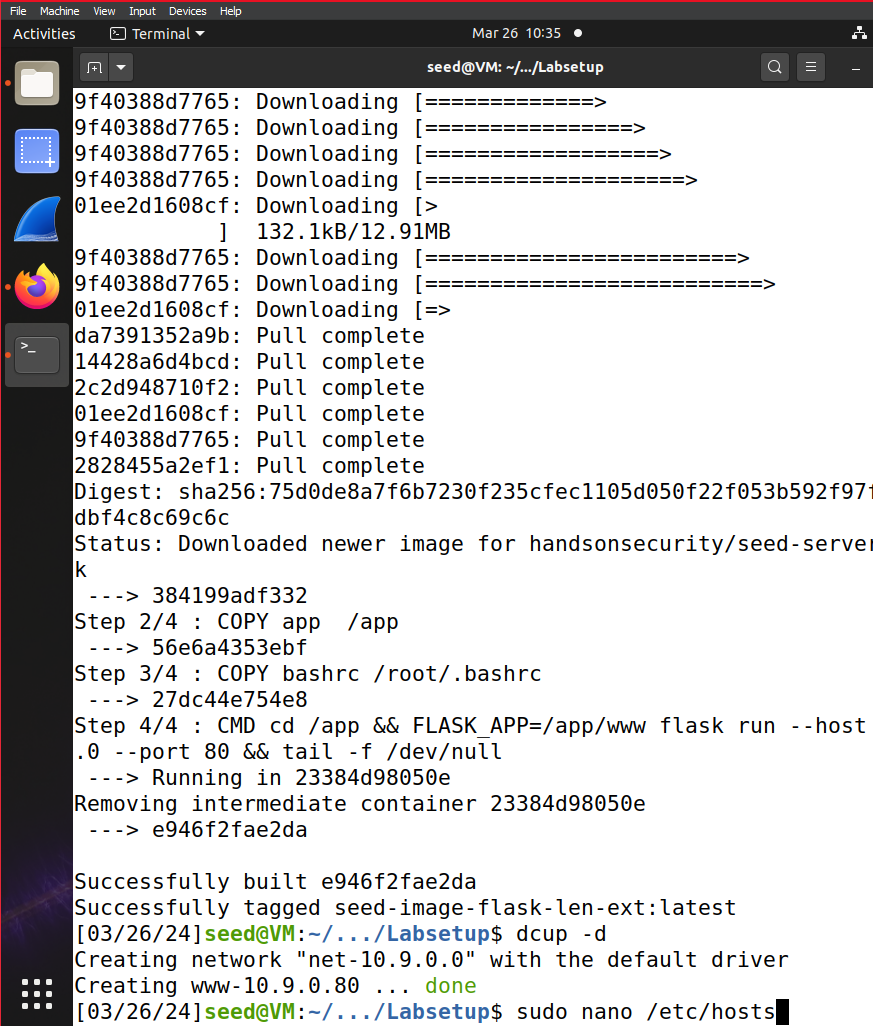


1. If necessary, get the ID of the running container using dockps and then start a shell on that container using :

Docksh [container name]

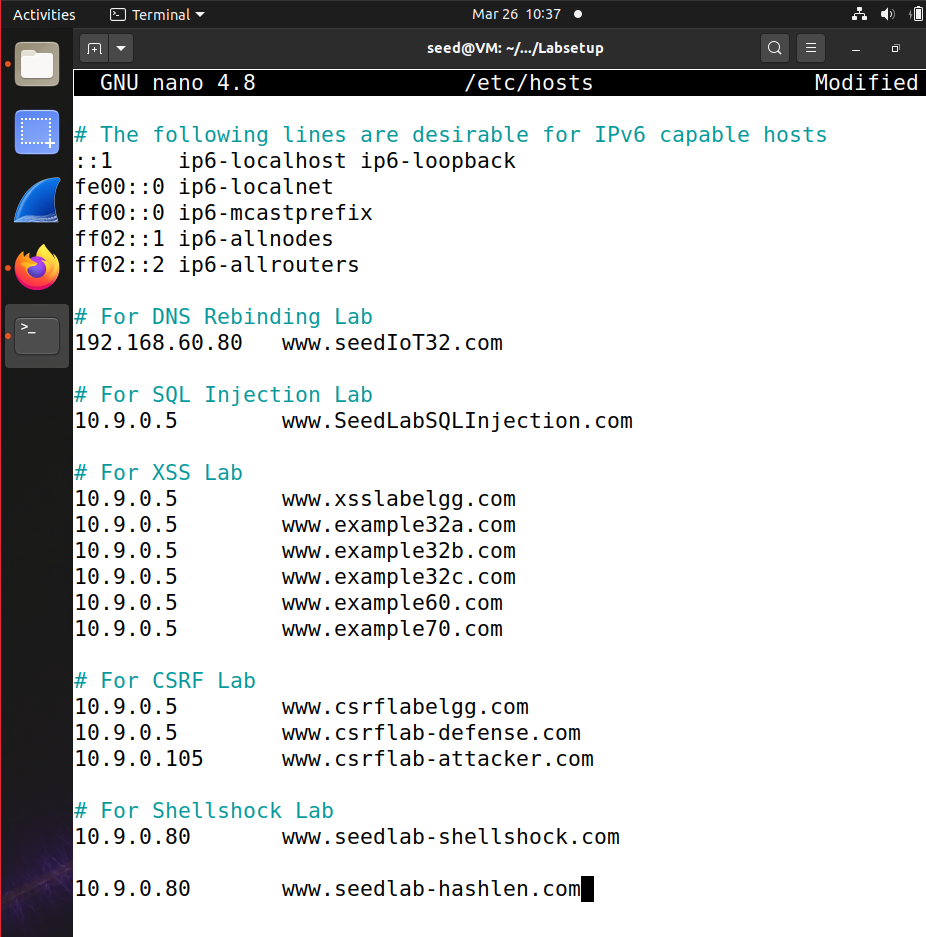
1. Add the following entry to /etc/hosts:

sudo nano /etc/hosts



1. Add a line:

10.9.0.80 [www.seedlab-hashlen.com](http://www.seedlab-hashlen.com)



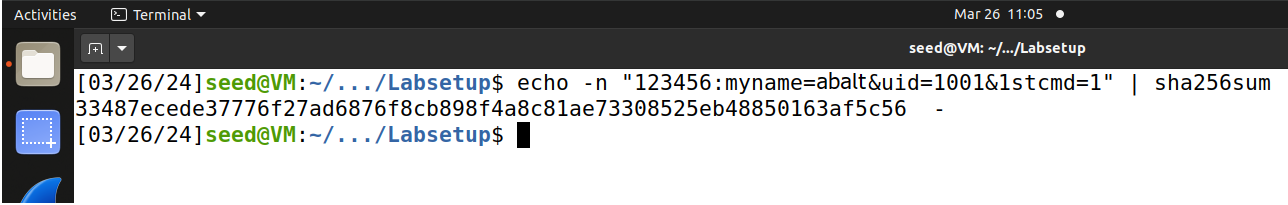
# Task 1: Create and send a harmless request to the server:

This task creates a harmless HTTP request to the server with the specified parameters. We obtain the user ID (UID) and the associated key from a predefined file. We use this information to calculate the Message Authentication Code (MAC) and add it to the request URL. Finally, we send a request to interact with the server via a web browser. We use abalt as an example for [ my name ] variable.

## Steps

1. Select the UID with the corresponding key value in “Labsetup/image\_flask/app/LabHome/key.txt”, for example “1001:123456” to send the request to the server.
2. Calculate the MAC of the key associated with the body of the R request:

echo -n "123456:myname=abalt&uid=1001&lstcmd=1" sum sha256



1. Create the full request URL and access it using a web browser:

http://www.seedlab-hashlen.com/?myname=koji&uid=1001&lstcmd=1&mac=mac address from last step

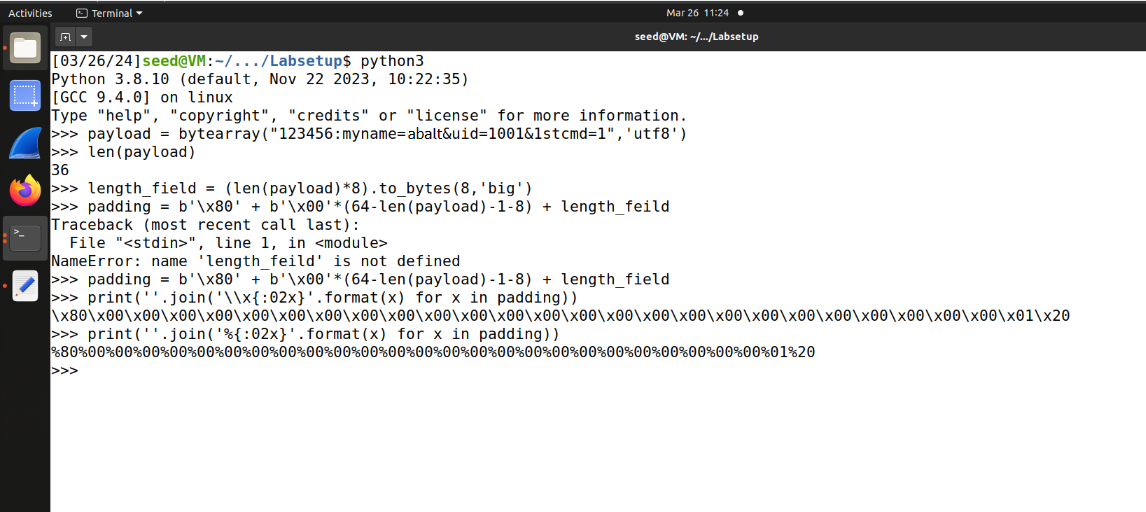


# Task 2: Create a complement for a specific message:

Padding is crucial in cryptographic operations, especially hash length expansion attacks. In this task, we use Python to calculate the required spacing for a specific message. This padding ensures that the length of the message matches the block size of the cryptographic hash function.

## Step

1. Use Python3 to calculate message padding using following commands as shown below:



# Task 3: Compile and run “calculate\_mac.c”:

To calculate the MAC of the message, we use a C program that uses the OpenSSL library. This program takes the message as padded input and calculates the corresponding MAC address using the SHA-256 hash function. Compiling and running this program shows how MAC addresses are generated in the real world.

## Steps

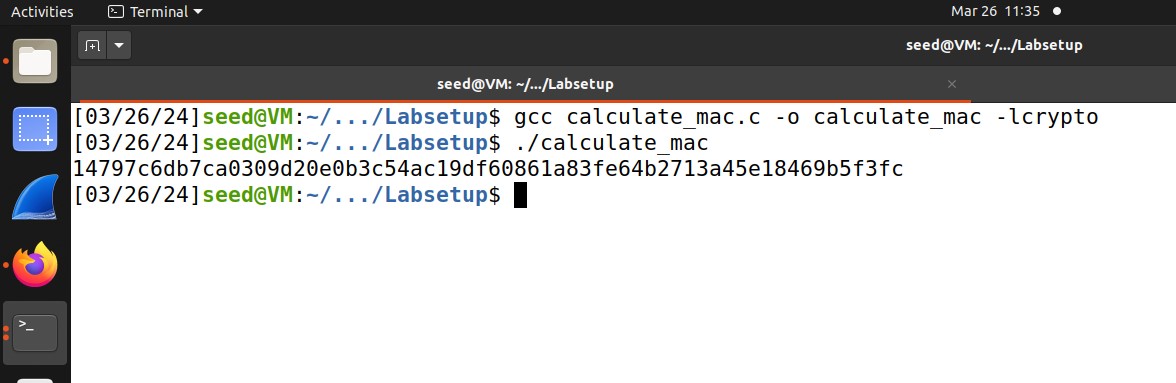
1. Write the program calculate\_mac.c:



This program takes the padding bytes calculated before as a second argument.

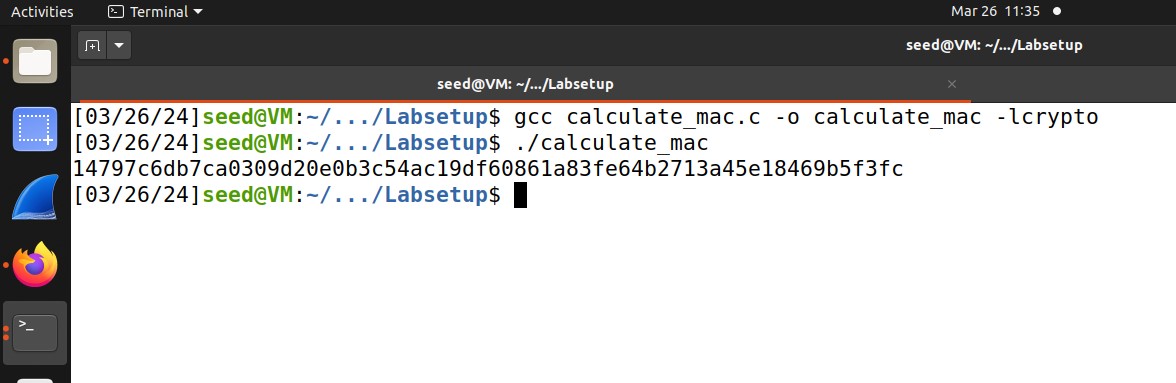
1. Compile the program in C:

gcc calculate\_mac.c -o calculate\_mac -lcrypto



1. Run the compiled program:

./calculate\_mac

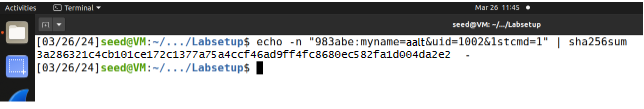


# Task 4: Create a query with an unknown MAC key:

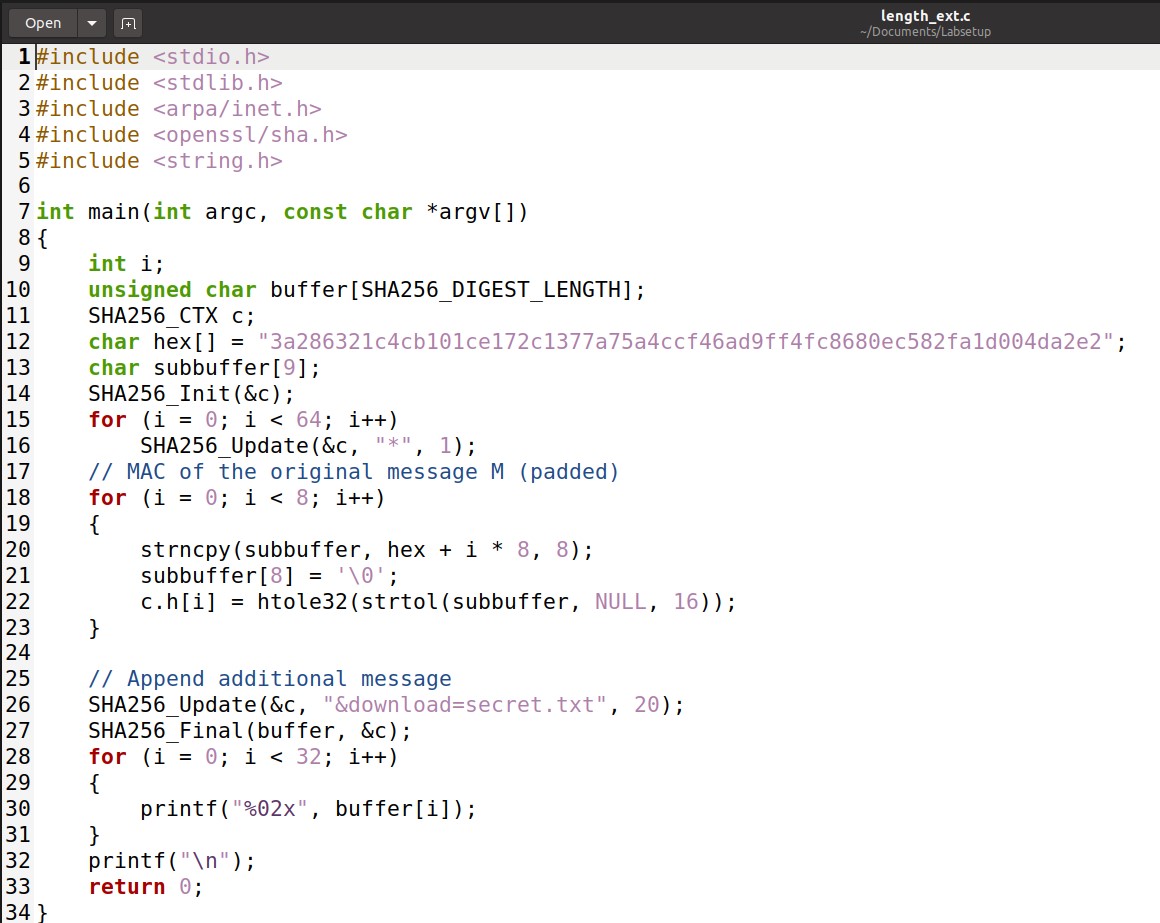
In this activity, we examine the scenario where the MAC key is unknown. Although we don't know the exact key, we looked at the full URL of the request, including the MAC address. Using a length expansion attack, we manipulate the query to include additional parameters while preserving the integrity of the original message.

## Steps

1. Now we take another uid and key from the file and calculate its hash.



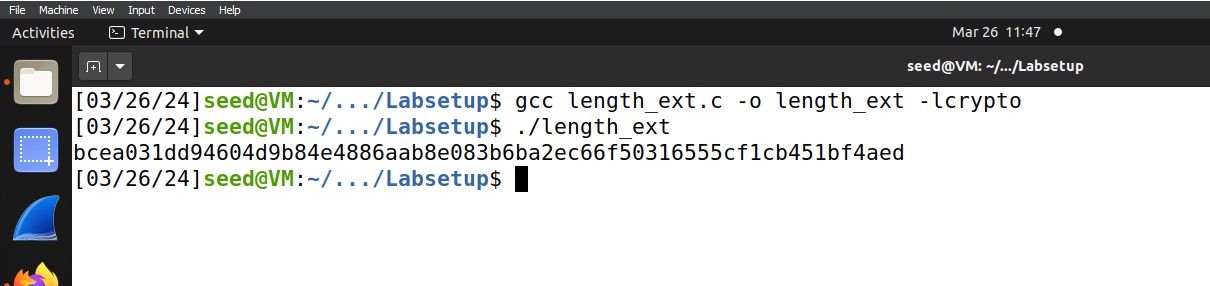
1. Now we write the program for length.c



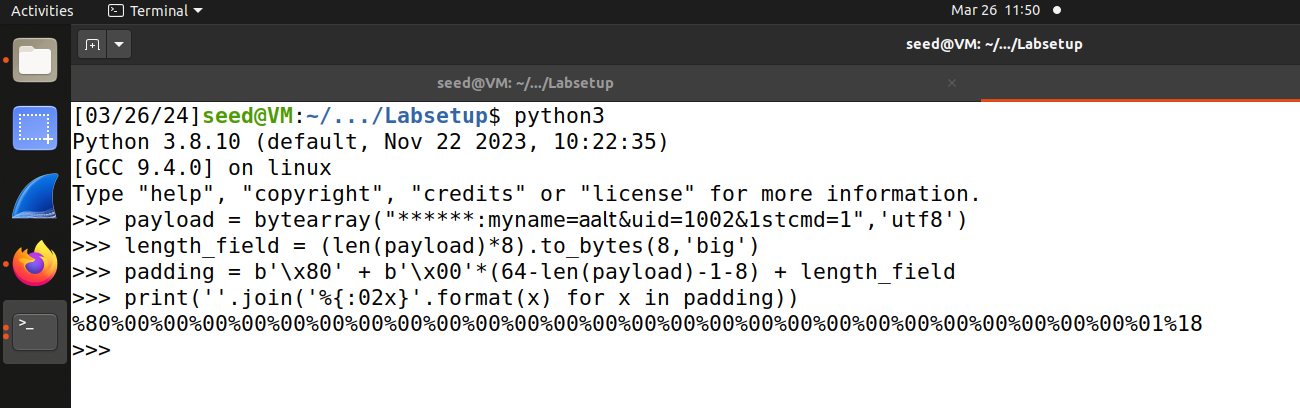
1. Compile and run length\_ext.c:

gcc extension\_length.c -o extension\_length -lcrypto

./ext\_length



1. Construct the padding of the original message using Python.



1. Now we use the data obtained above to visit the website.



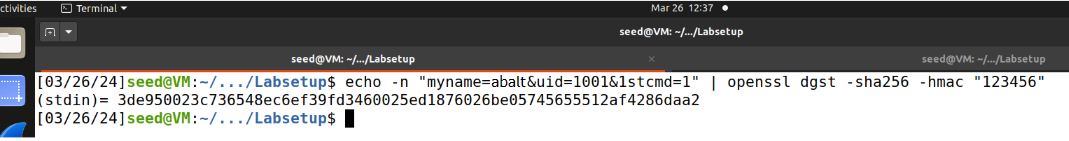
# Task 5: Understanding HMAC:

## Hashed Message Authentication Code:

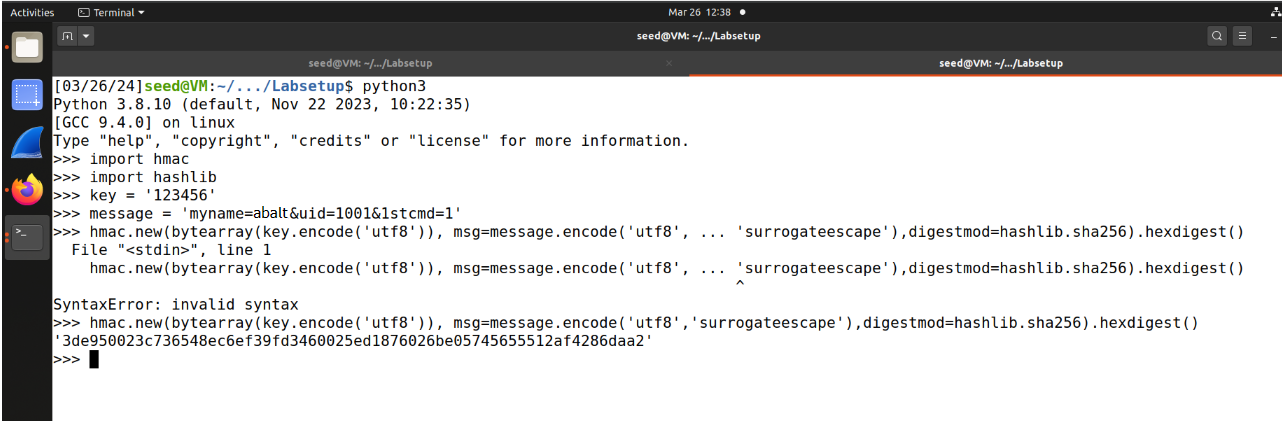
(HMAC) is a widely used cryptographic technique that ensures the integrity and authenticity of messages. In this assignment, we will explore the concept of HMAC and its importance for secure communication protocols. By understanding the inner workings of HMAC, we gain insight into its resilience to hash length extension attacks.

## Steps

1. Understand HMAC and its application in message authentication.The following step provides instructions for understanding HMAC. We can use following to find the hmac.



We can also calculate the above using python as well



## Conclusion for HMAC

The HMAC algorithm requires the use of a MAC key in both hash functions. The output of the internal hash serves as input to the external hash. Without knowledge of the intermediate results (internal hashing), the attacker cannot accurately calculate the final MAC. This sequential HMAC design prevents attackers from directly constructing the MAC address of the extended message only from the final MAC address of the legitimate request. Therefore, attempts to exploit hash length expansion vulnerabilities on servers implementing HMAC fail because the internal hash result remains hidden and is an integral part of the final MAC calculation.