

CPS 472/572 Fall 2024

Prof: Zhongmei Yao

Lab 2 Report: Hash Length Extension Attack Lab

1. Dinesh Kumar Kala (kalad1@udayton.edu)

Student ID:101745354

2. Akhil Reddy Kotha (kothaa6@udayton.edu)

Student ID:101790043

1. Overview:

When a client communicates with a server, there's a risk of a Man-in-the-Middle (MITM) attack where an attacker can intercept and modify the client's request. To ensure data integrity, servers use a Message Authentication Code (MAC), which is derived from a secret key and the message itself. A common but flawed approach involves concatenating the key and message and then hashing them. This method is vulnerable to length extension attacks, allowing attackers to alter the message and generate a valid MAC without knowing the key. In this lab, students will explore how this attack works by creating and exploiting forged commands to manipulate a server program.

Takeaways: In this lab, we learned that a web server uses a Message Authentication Code (MAC) for validating client commands. The server is set up in Docker and is accessible via a specific URL. The MAC is generated using SHA-256 combined with a secret key. The lab demonstrates how a hash length extension attack can exploit weaknesses in this method. To mitigate this vulnerability, the `verify_mac()` function is improved by implementing HMAC, which offers a more secure way to compute the MAC and protect against such attacks.

2. Lab Environment:

For the 2nd Lab, we downloaded the Lab setup.zip from the link given in the Assignment. After unzipping the file. We build the Docker using the “dcbuild” command.

```
[09/08/24]seed@VM:~/.../kalakotha$ dcbuild
Building web-server
Step 1/4 : FROM handsonsecurity/seed-server:flask
---> 384199adf332
Step 2/4 : COPY app /app
---> 41d6ae0a2108
Step 3/4 : COPY bashrc /root/.bashrc
---> 823884cd4f2e
Step 4/4 : CMD cd /app && FLASK_APP=/app/www flask run --host 0.0.0.0 --port 80
&& tail -f /dev/null
---> Running in 52a5872725da
Removing intermediate container 52a5872725da
---> 79e823db2946

Successfully built 79e823db2946
Successfully tagged seed-image-flask-len-ext:latest
[09/08/24]seed@VM:~/.../kalakotha$
```

After Building successfully, we turned on the docker using the “dcup” command. Now, it is time to assign a web server to the IP Address: 10.9.0.80.

```
[09/08/24]seed@VM:~/.../kalakotha$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.80 ... done
Attaching to www-10.9.0.80
www-10.9.0.80 | * Serving Flask app "/app/www"
www-10.9.0.80 | * Environment: production
www-10.9.0.80 | WARNING: This is a development server. Do not use it in a pro
duction deployment.
www-10.9.0.80 | Use a production WSGI server instead.
www-10.9.0.80 | * Debug mode: off
www-10.9.0.80 | * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

Using “sudo nano /etc/hosts” command. We went into the hosts' file and mapped the container IP address to the website.

www.seedlab-hashlen.com — 10.9.0.80

```
[09/09/24]seed@VM:~/.../kalakotha$ sudo nano /etc/hosts
[09/09/24]seed@VM:~/.../kalakotha$
```

```
seed@VM: ~/.../kalakotha
GNU nano 4.8 /etc/hosts
# For XSS Lab
10.9.0.5      www.xsslabelgg.com
10.9.0.5      www.example32a.com
10.9.0.5      www.example32b.com
10.9.0.5      www.example32c.com
10.9.0.5      www.example60.com
10.9.0.5      www.example70.com

# For CSRF Lab
10.9.0.5      www.csrflabelgg.com
10.9.0.5      www.csrf lab-defense.com
10.9.0.105    www.csrf lab-attacker.com

# For Shellshock Lab
10.9.0.80     www.seedlab-shellshock.com

# For Webserver
10.9.0.80     www.seedlab-hashlen.com
```

Now after adding the ip address in hosts, we are ready to do the tasks for this Lab. At the beginning, we confused about how to set up this Lab but with the help of Linux commands like how to edit a file and save a file, we successfully completed the Lab set up for this Task.

Takeaways: In this lab, we use Docker Compose to set up containers, with key commands aliased for convenience. Our web server is hosted at www.seedlab-hashlen.com, and the server code and secret files are stored in specific directories. To send commands, we must include a valid MAC and our real name in the request. The MAC ensures the server verifies and securely processes our commands.

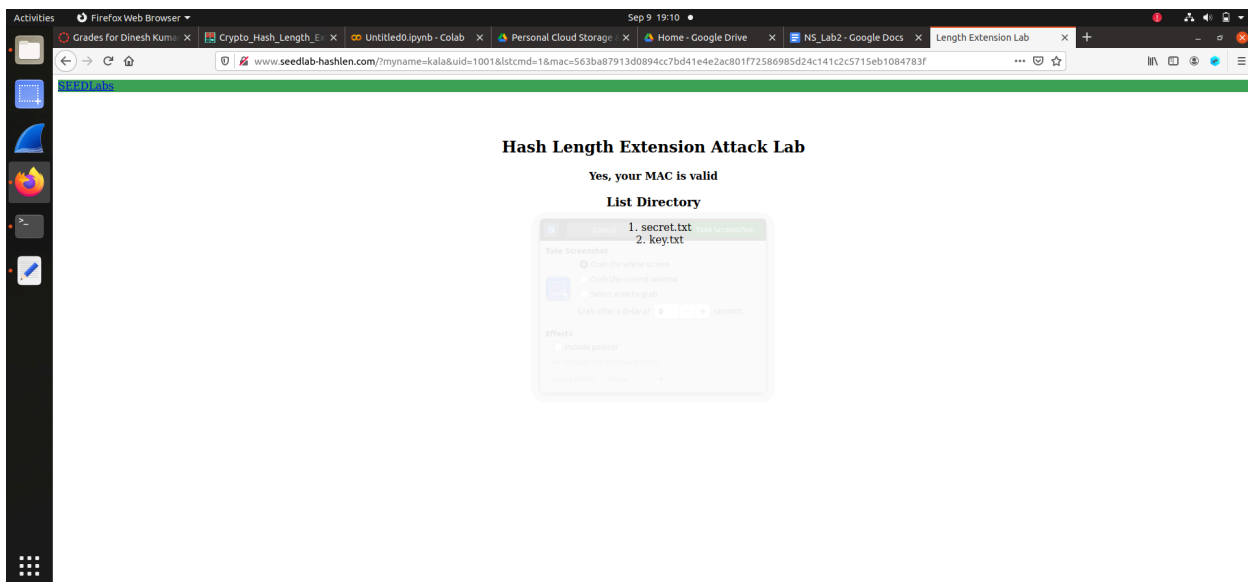
3. Tasks

3.1 Task 1: Send Request to list Files:

To do this task, we need key, name and uid associated with the key. We picked up the Key and uid from key.txt from labhome directory and for “myname=” i used my last name “kala”. Now we are all set to generate a “MAC”.

```
seed@VM: ~/.../kalakotha
[09/09/24]seed@VM:~/.../kalakotha$ echo -n "123456:myname=kala&uid=1001&lstcmd=1" | sha256sum
563ba87913d0894cc7bd41e4e2ac801f72586985d24c141c2c5715eb1084783f -
[1]+  Exit 127                  http://www.seedlab-hashlen.com/?myname=kala
[09/09/24]seed@VM:~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kala&uid=1001&lstcmd=1&mac=563ba87913d0894cc7bd41e4e2ac801f72586985d24c141c2c5715eb1084783f
[1] 18947
[2] 18948
[3] 18949
bash: http://www.seedlab-hashlen.com/?myname=kala: No such file or directory
[1] Exit 127                  http://www.seedlab-hashlen.com/?myname=kala
[2]- Done                    uid=1001
[3]+ Done                    lstcmd=1
[09/09/24]seed@VM:~/.../kalakotha$
```

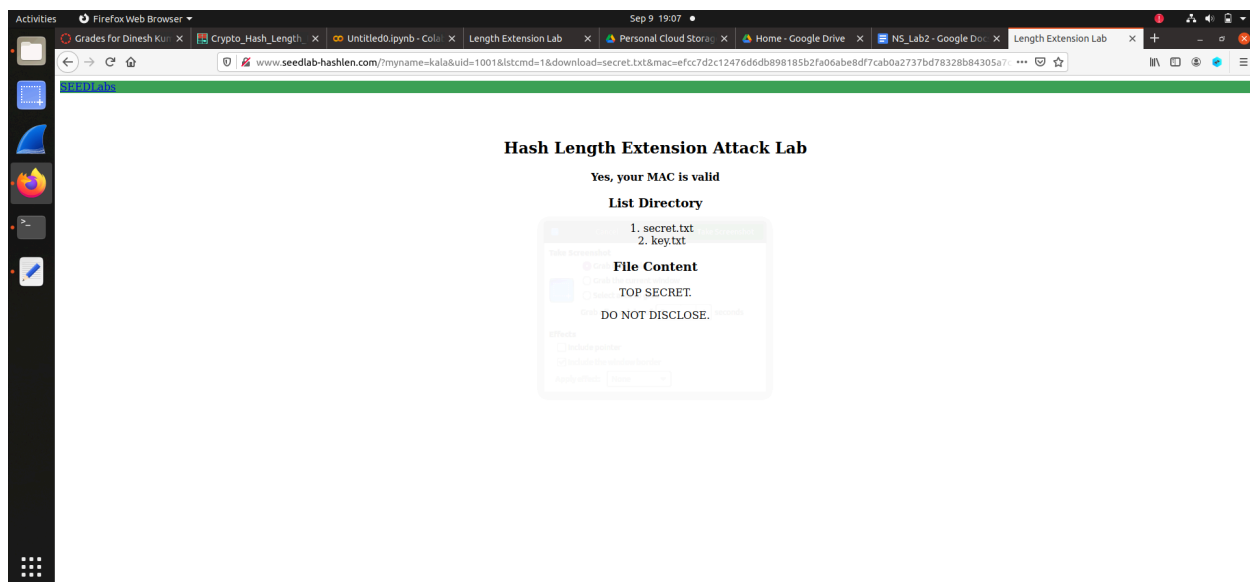
To, generate a MAC, we used echo -n “key:myname=<my_last_name>&uid=1001&lstcmd=1” message and **SHA256sum** hash function to generate a MAC. After Getting MAC, we constructed a complete request to send to the server.



Then we copy pasted the url in the firefox and we got the desired output. Above image shows the list of files.

```
seed@VM: ~/.../kalakotha$ echo -n "123456:myname=kala&uid=1001&lscmd=1&download=secret.txt" | sha256sum
efcc7d2c12476d6db898185b2fa06abe8df7cab0a2737bd78328b84305a7c023 -
[09/09/24] seed@VM: ~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kala&uid=1001&lscmd=1&download=secret.txt&mac=efcc7d2c12476d6db898185b2fa06abe8df7cab0a2737bd78328b84305a7c023
[1] 18771
[2] 18772
[3] 18773
[4] 18774
bash: http://www.seedlab-hashlen.com/?myname=kala: No such file or directory
[2] Done uid=1001
[3]- Done lscmd=1
[4]+ Done download=secret.txt
[09/09/24] seed@VM: ~/.../kalakotha$
```

We got the expected output for the above task. Now its time to see what is inside secret.txt file. For that, we added “&download=secret.txt” to the above message and using the SHA256sum we generated the MAC. We constructed the request to send to the server.



After pasting the URL in the firefox, we successfully fetched the content inside secret.txt file. Above image shows the result.

Takeaways: From this task, I learned how to construct and send a request to a web server by filling in specific arguments. We used a uid and key and calculated a MAC (Message Authentication Code) to ensure the request's integrity. The MAC was computed by concatenating the key with the request parameters and then applying a hashing function. After constructing the full request with our real name, the correct uid, and the MAC, we sent it to the server and observed its response.

3.2 Task 2: Create Padding:

In the Last Lab (Secret Key Encryption), we learned what is the use of padding and how to use it. This time we are using SHA256sum HASH function, which means it takes input as 64 bytes. so, we need to make sure each block is of length 64bytes. If the message not 64 bytes we need to add the padding to make it as a 64 bytes.

But SHA256 consists of `\x80` (one byte) which refers as starting of padding and ends with a length field which is the number of bits in the message.

Example : “msg + \x80 (starting of padding) + padding + length filed(total bits in message) = 64 bytes

Now, we have a message “123456:myname=kala&uid=1001&lstcmd=1” which is not a 64 bytes. But for the SHA256sum HASH function, the input needs to be a 64 bytes. This is where we need padding to make it 64 bytes.

[illegible]

To make my message as a 64 bytes, i need padding but i don't know how much padding i required to make it 64 bytes. Using the above python code, i can get the padding required for my message. But i need padding for url.

[illegible]

To get padding in URL format, I used above Python code where we gave our message as a input and this python code converts my message into padding (URL format). Now i can use this padding in URL.

Takeaways: We learned how to add padding to SHA-256, the message is padded with a 'x80' byte, followed by zeros, and ends with a 64-bit length field. The length is encoded in bits, and hexadecimal values in the padding are represented in URL format as '%' followed by the hex value (e.g., 'x80' becomes '%80'). This approach allows us to extend the message and generate a valid MAC for a new request, even without knowing the MAC key.

Doubt: we got a doubt, how this padding helps hacker to hack? So, we asked ChatGpt.

Answer Given by ChatGpt: “This standardized padding ensures that the message fits into the required block size for hashing. Hackers exploit this predictable padding to extend the original message by appending additional data. By knowing the hash of the original message and its padding structure, attackers can forge new messages with valid hashes, effectively bypassing security measures that rely on hash verification. This means they can manipulate requests and perform actions that the server would consider legitimate, even without access to the original key”.

3.3 Task 3: The Length Extension Attack:

To do this task, I made my friend as a Hacker, so we used his Last name “kotha” as a myname and used different Key and uid.

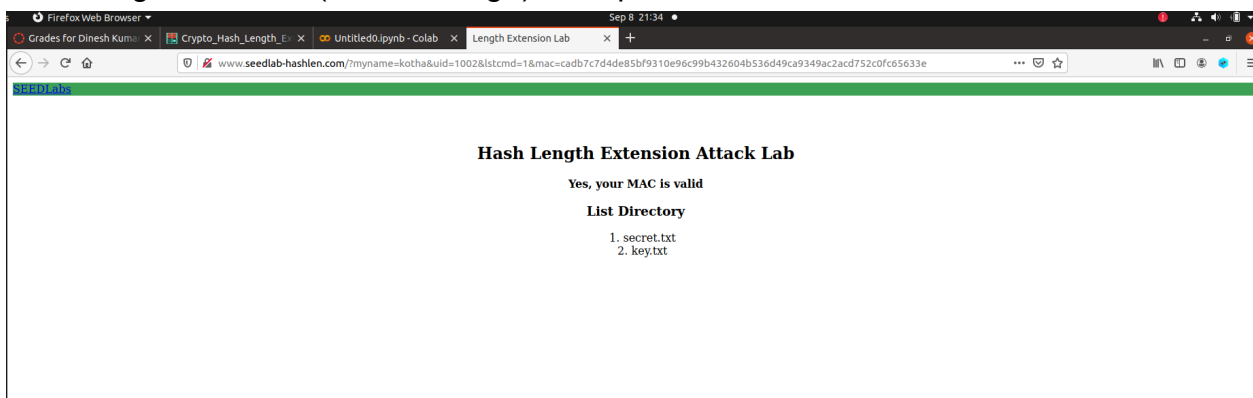
Initially we generated a MAC for the “Istcmd” and assembled the key, myname, uid, MAC, and then sent request to the server.

```

[09/08/24]seed@VM:~/.../kalakotha$ echo -n "983abe:myname=kotha&uid=1002&lstcmd=
1" | sha256sum
cadb7c7d4de85bf9310e96c99b432604b536d49ca9349ac2acd752c0fc65633e -
[09/08/24]seed@VM:~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kotha&
uid=1002&lstcmd=1&mac=cadb7c7d4de85bf9310e96c99b432604b536d49ca9349ac2acd752c0fc
65633e
[1] 16377
[2] 16378
[3] 16379
[2]- Done uid=1002
[3]+ Done lstcmd=1
bash: http://www.seedlab-hashlen.com/?myname=kotha: No such file or directory
[09/08/24]seed@VM:~/.../kalakotha$ █

```

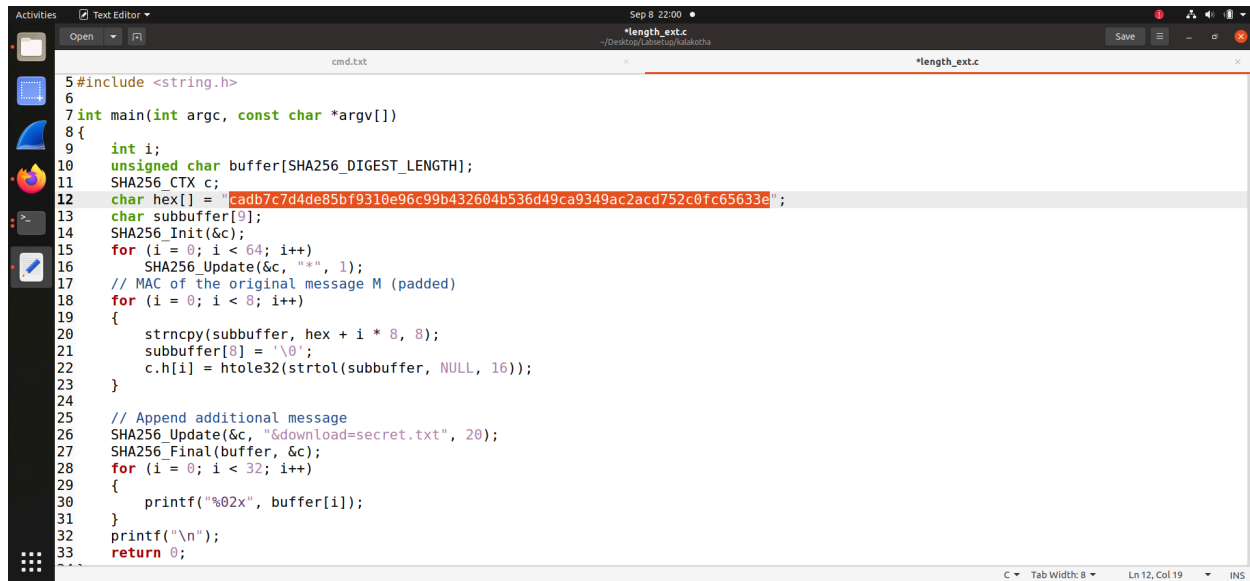
Now, we got the result(Below Image) as expected.



Hackers can only see the URL which is (myname, uid, padding, and MAC). But the hacker didn't know the key.

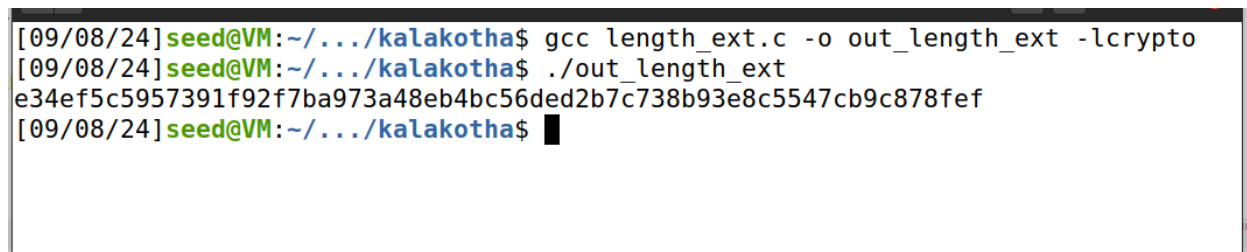
Now it's time to hack!!

Hacker know the URL and don't know the key. But using the Message and MAC. he can add extra message and generate a new MAC.

A screenshot of a text editor window titled 'length_ext.c' showing C code for calculating a SHA256 MAC. The code includes `<string.h>` and defines a `main` function. It initializes a SHA256 context `c` and a buffer of size `SHA256_DIGEST_LENGTH`. A hexadecimal string `hex` is defined, and a subbuffer is used to process it in 8-byte chunks. The code then appends the message `"&download=secret.txt"` to the context and calculates the final MAC, which is printed in hexadecimal format.

```
5#include <string.h>
6
7int main(int argc, const char *argv[])
8{
9    int i;
10   unsigned char buffer[SHA256_DIGEST_LENGTH];
11   SHA256_CTX c;
12   char hex[] = "cadb7c7d4de85bf9310e96c99b432604b536d49ca9349ac2acd752c0fc65633e";
13   char subbuffer[8];
14   SHA256_Init(&c);
15   for (i = 0; i < 64; i++)
16       SHA256_Update(&c, hex + i * 8, 8);
17   // MAC of the original message M (padded)
18   for (i = 0; i < 8; i++)
19   {
20       strncpy(subbuffer, hex + i * 8, 8);
21       subbuffer[8] = '\0';
22       c.h[i] = htobe32(strtol(subbuffer, NULL, 16));
23   }
24
25   // Append additional message
26   SHA256_Update(&c, "&download=secret.txt", 20);
27   SHA256_Final(buffer, &c);
28   for (i = 0; i < 32; i++)
29   {
30       printf("%02x", buffer[i]);
31   }
32   printf("\n");
33   return 0;
```

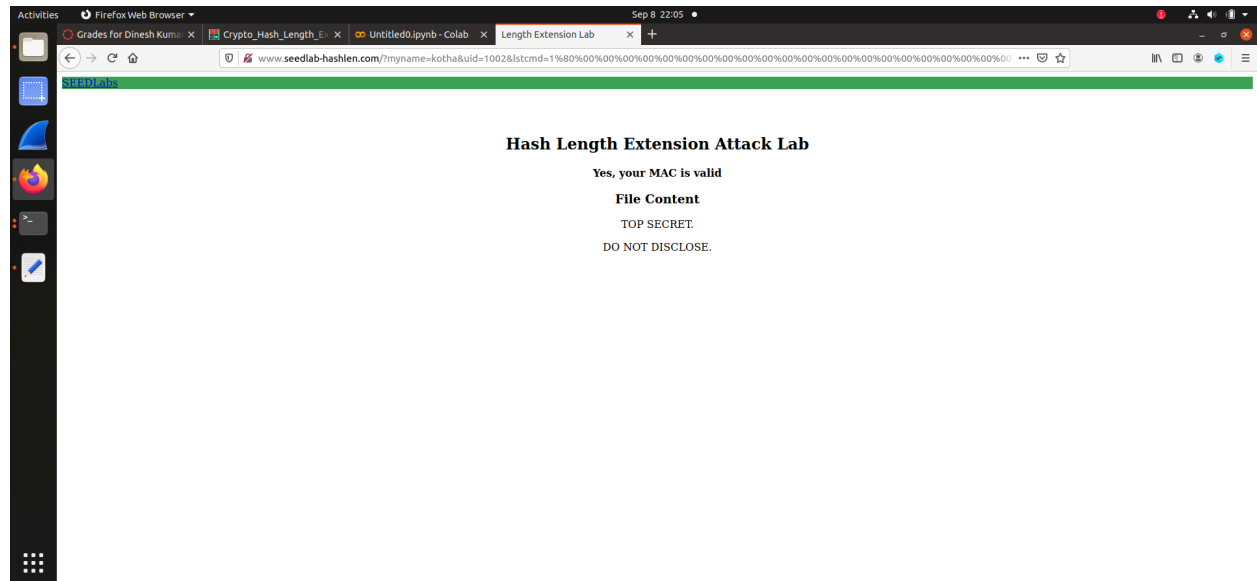
I used the above .c code to add extra message and to generate a new MAC.

A screenshot of a terminal window showing the compilation and execution of the C program. The user runs `gcc length_ext.c -o out_length_ext -lcrypto` and then `./out_length_ext`. The output is a 64-character hexadecimal string representing the new MAC.

```
[09/08/24] seed@VM:~/.../kalakotha$ gcc length_ext.c -o out_length_ext -lcrypto
[09/08/24] seed@VM:~/.../kalakotha$ ./out_length_ext
e34ef5c5957391f92f7ba973a48eb4bc56ded2b7c738b93e8c5547cb9c878fef
[09/08/24] seed@VM:~/.../kalakotha$
```

I ran the .c file in the terminal, which gave me a new MAC (generated using the old MAC and Extra Message which is download=secret.txt). Now I got a new MAC, and I need padding.

I got the new MAC, PAdding, Extra Message. Now i gathered them all sent request to server to check.



The URL is working which means, we can do the Length extension attack, without knowing the key.

Takeaways: We learned how to perform a hash length extension attack by using the SHA-256 padding process to append additional data to an original message. By leveraging the known MAC and padding details, we generated a valid MAC for the modified message. This allowed us to forge a URL with extra content and access protected resources, demonstrating a vulnerability in handling MACs and padding.

3.4 Task 4: Attack Mitigation using HMAC:

Since, without knowing the Key, Hacker using the Length extension attack he can hack the server using the MAC and Message. To Defeat the Hacker, we are using the HMAC (Hash-based Message Authentication Code).

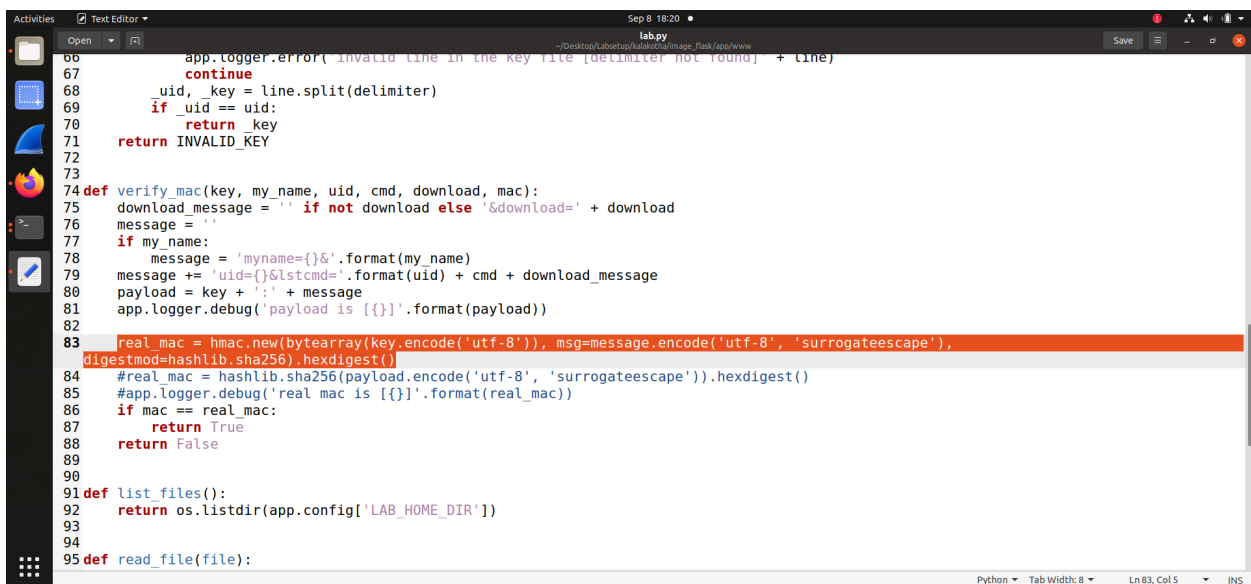
How HMAC works?

We went through the slides, and referred some websites and we finalized this "HMAC, or Hash-based Message Authentication Code, ensures message integrity and authenticity by combining a secret key with a message and applying a cryptographic hash function like SHA-256. The key is processed, XORed with inner and outer paddings, and hashed twice, once with the message and once with the intermediate hash. The result is a unique code (MAC) that verifies the message's authenticity and confirms it hasn't been altered".

Now, to do this task, First I shut down the docker using the “dcdowndown” command.

```
[09/08/24]seed@VM:~/.../kalakotha$ dcdowndown
Stopping www-10.9.0.80 ... done
Removing www-10.9.0.80 ... done
Removing network net-10.9.0.0
[09/08/24]seed@VM:~/.../kalakotha$
```

Next, we made changes to the code, commented on the original code, and then pasted the HMAC code(given in the PDF) and saved it successfully.



```
66         app.logger.error('invalid line in the key file [delimiter not found] + line)
67         continue
68         uid, _key = line.split(delimiter)
69         if _uid == uid:
70             return _key
71         return INVALID_KEY
72
73
74 def verify_mac(key, my_name, uid, cmd, download, mac):
75     download_message = '' if not download else '&download=' + download
76     message = ''
77     if my_name:
78         message = 'myname={}'.format(my_name)
79     message += 'uid={}&lscmd='.format(uid) + cmd + download_message
80     payload = key + ':' + message
81     app.logger.debug('payload is {}'.format(payload))
82
83     real_mac = hmac.new(bytearray(key.encode('utf-8')), msg=message.encode('utf-8', 'surrogateescape'),
84                          digestmod=hashlib.sha256).hexdigest()
85     #real_mac = hashlib.sha256(payload.encode('utf-8', 'surrogateescape')).hexdigest()
86     #app.logger.debug('real mac is {}'.format(real_mac))
87     if mac == real_mac:
88         return True
89     return False
90
91 def list_files():
92     return os.listdir(app.config['LAB_HOME_DIR'])
93
94
95 def read_file(file):
```

After changing the code, we rebuilt the container using the dcbuild.

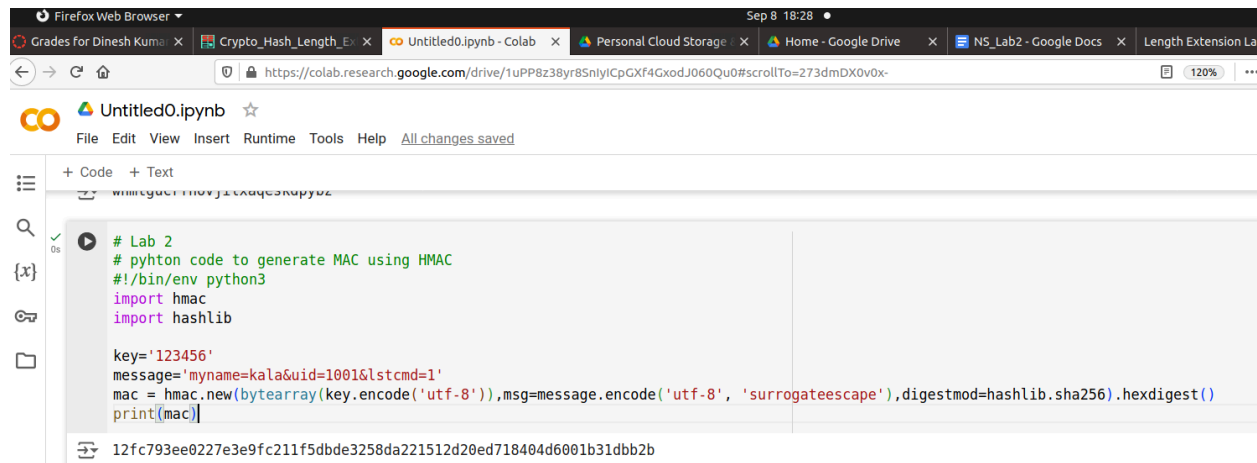
```

[09/08/24]seed@VM:~/.../kalakotha$ dcbuild
Building web-server
Step 1/4 : FROM handsontsecurity/seed-server:flask
---> 384199adf332
Step 2/4 : COPY app /app
---> Using cache
---> 7191e29eb4fc
Step 3/4 : COPY bashrc /root/.bashrc
---> Using cache
---> 39135587862c
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
---> 86178f79c625

Successfully built 86178f79c625
Successfully tagged seed-image-flask-len-ext:latest
[09/08/24]seed@VM:~/.../kalakotha$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating www-10.9.0.80 ... done
Attaching to www-10.9.0.80
www-10.9.0.80 | * Serving Flask app "/app/www"
www-10.9.0.80 | * Environment: production
www-10.9.0.80 | WARNING: This is a development server. Do not use it in a pro

```

Now, its time to generate the MAC using the HMAC. for that we used python code. This time to generate the MAC we used, HMAC.



```

# Lab 2
# python code to generate MAC using HMAC
#!/bin/env python3
import hmac
import hashlib

key='123456'
message='myname=kala&uid=1001&lscmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),msg=message.encode('utf-8', 'surrogateescape'),digestmod=hashlib.sha256).hexdigest()
print(mac)

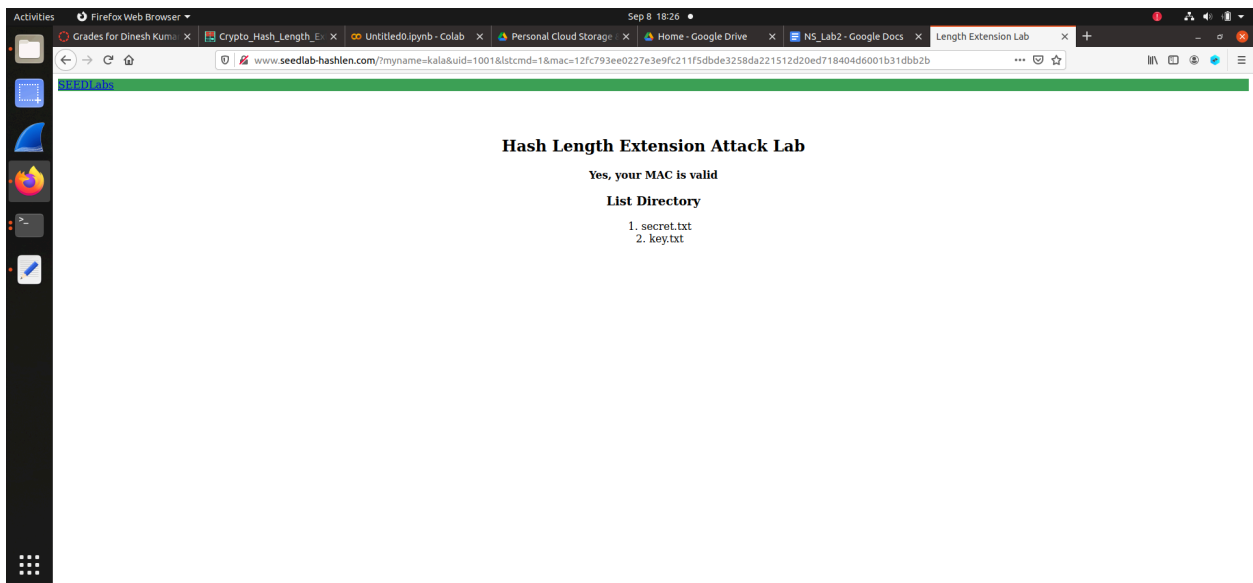
```

12fc793ee0227e3e9fc211f5dbde3258da221512d20ed718404d6001b31dbb2b

To test the HMAC MAC we went through the Task1 and executed the list command request followed by the download=secret.txt.

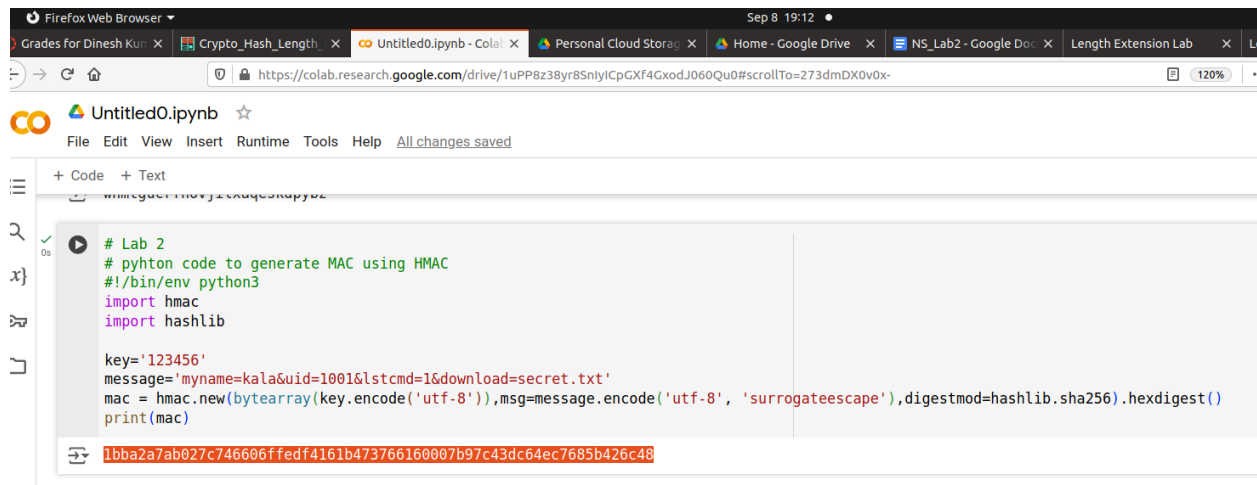
```
[09/08/24] seed@VM: ~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kala&uid=1001&lstcmd=1&mac=12fc793ee0227e3e9fc211f5dbde3258da221512d20ed718404d6001b31dbb2b
[1] 12093
[2] 12094
[3] 12095
bash: http://www.seedlab-hashlen.com/?myname=kala: No such file or directory
[1] Exit 127 http://www.seedlab-hashlen.com/?myname=kala
[2]- Done uid=1001
[3]+ Done lstcmd=1
[09/08/24] seed@VM: ~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kala&uid=1001&lstcmd=1&mac=12fc793ee0227e3e9fc211f5dbde3258da221512d20ed718404d6001b31dbb2b
```

Using the MAC given by the Python code, we send a request to the server to check the output and we got the expected response.



Used MAC generated by HMAC to fetch Text in secret.txt:

And after that we added an extra message (download=secret.txt) to original message and given as a input to the python code. This time we got different MAC.

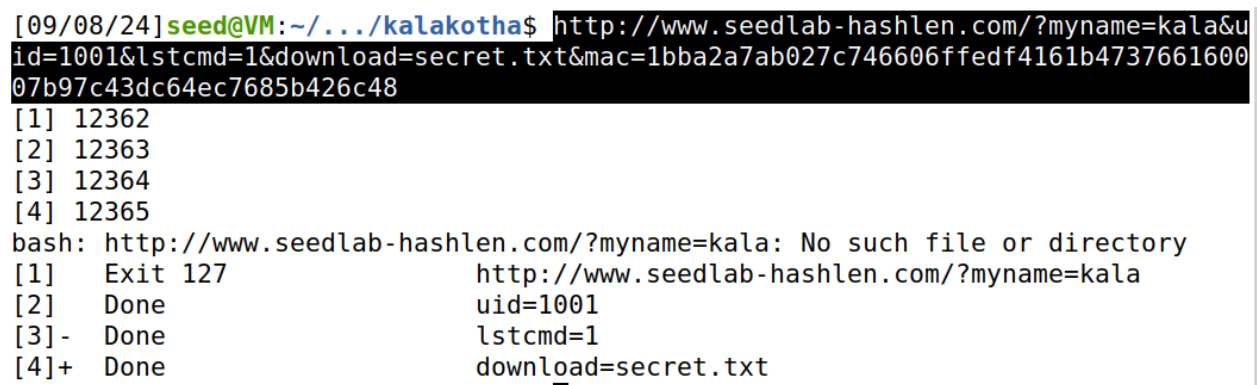


```
# Lab 2
# python code to generate MAC using HMAC
#!/bin/env python3
import hmac
import hashlib

key='123456'
message='myname=kala&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)
```

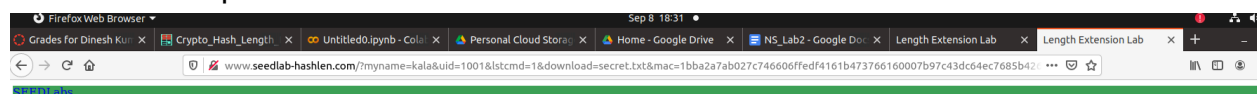
1bba2a7ab027c746606ffedf4161b473766160007b97c43dc64ec7685b426c48

Then again we added download=secret.txt to our message and Given the MAC from the above.



```
[09/08/24] seed@VM: ~/.../kalakotha$ http://www.seedlab-hashlen.com/?myname=kala&uid=1001&lstcmd=1&download=secret.txt&mac=1bba2a7ab027c746606ffedf4161b473766160007b97c43dc64ec7685b426c48
[1] 12362
[2] 12363
[3] 12364
[4] 12365
bash: http://www.seedlab-hashlen.com/?myname=kala: No such file or directory
[1] Exit 127 http://www.seedlab-hashlen.com/?myname=kala
[2] Done uid=1001
[3]- Done lstcmd=1
[4]+ Done download=secret.txt
```

Here is the Output.



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.

If we use this HMAC, to generate a MAC it's difficult for hackers to change/modify the message or MAC.

Question: describe why a malicious request using length extension and extra commands will fail MAC verification when the client and server use HMAC.

Answer: A malicious request using length extension or extra commands will fail HMAC verification because HMAC links the MAC closely to both the key and the message. When generating the MAC, HMAC processes the key and message together in a secure way. If someone tries to change the message or add extra data, the MAC will not match the altered message when checked. This ensures that any tampering with the message will be detected, making HMAC effective at preventing such attacks and keeping the message intact.

Takeaways: In this task, we learned that securely computing a MAC is crucial to prevent vulnerabilities like length extension attacks. By using HMAC with Python's `hmac` module and SHA-256, we improved security. HMAC ensures that any changes or additions to the message will cause MAC verification to fail, as it tightly binds the MAC to the specific key and message. Updating the server's `verify_mac()` function demonstrated that HMAC effectively protects message integrity and prevents unauthorized alterations.