Network Implementation & Security

Hash Length Extension Attack Lab

Jin Jung, Joshua Barrs, John Park

1 Introduction

When a client and a server communicate over the internet, they are subject to MITM attacks. An attacker can intercept the request from the client. The attacker may choose to modify the data and send the modified request to the server. In such a scenario, the server needs to verify the integrity of the request received. The standard way to verify the integrity of the request is to attach a tag called MAC to the request. There are many ways to calculate MAC, and some of the methods are not secure.

MAC is calculated from a secret key and a message. A naive way to calculate MAC is to concatenate the key with the message and calculate the one way hash of the resulting string. This method seems to be fine, but it is subject to an attack called length extension attack, which allows attackers to modify the message while still being able to generate a valid MAC based on the modified message, without knowing the secret key.

The objective of this lab is to help students understand how the length extension attack works. Students will launch the attack against a server program; they will forge a valid command and get the server to execute the command.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM.

2 Lab Setup

We have set up a server for this lab. A client can send a list of commands to this server. Each request must attach a MAC computed based on a secret key and the list of commands. The server will only execute the commands in the request if the MAC is verified successfully.

Setting up the hostname. We use the domain *www.seedlablenext.com*to host the server program. Since we only use one VM for this lab, we map this hostname *to localhost(127.0.0.1)*. This can be achieved by adding the following entry to the */etc/hosts* file.

```
127.0.0.1 www.seedlablenext.com
```

The server program. The server program (server.zip) can be found on Blackboard. After downloading this zip file, uncompress it and check its contents:

```
$ unzip server.zip
$ cd Server
$ ls
LabHome run_server.sh www
```

The www directory contains the server code, and the *LabHome* directory contains a secret file and the key used for computing the MAC. We can run the following command to start the server program.

```
$ chmod +x run_server.sh
$ ./run_server.sh
```

The server uses a Python module named Flask. By the time this lab is officially released, the VM should have this module installed. If you see an error message that says “Flask not found”, you can use the command below to install Flask.

```
$ sudo pip3 install Flask==1.1.1
```

Sending requests. The server program accepts the following commands:

- The *lstcmd* command: the server will list all the files in the *LabHome*folder.

- The *download* command: the server will return the contents of the specified file from the *LabHome* directory.

A typical request sent by the client to the server is shown below. The server requires a uid argument to be passed. It uses uid to get the MAC key from *LabHome/key.txt*. The command in the example below is *lstcmd*, and its value is set to 1. It requests the server to list all the files. The last argument is the MAC computed based on the secret key (shared by the client and the server) and the command arguments. Before executing the command, the server will verify the MAC to ensure the command's integrity.

```
http://www.seedlablenext.com:5000/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

Students should replace the value *JohnDoe*in the *myname*field with their actual names (no space is allowed). This parameter is to make sure that different students' results are different, so students cannot copy from one another. The server does not use this argument, but it checks whether the argument is present or not. Requests will be rejected if this field is not included. Instructors can use this argument to check whether students have done the work by themselves. No point will be given if students do not use their real names in this task.

The following shows another example. The request includes two commands: list all the files and download the file *secret.txt*. Similarly, a valid MAC needs to be attached, or the server will not execute these commands.

```
http://www.seedlablenext.com:5000/?myname=JohnDoe&uid=1001&lstcmd=1
&download=secret.txt
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

3 Tasks

3.1 Task 1: Send Request to List Files

In this task, we will send a benign request to the server so we can see how the server responds to the request. The request we want to send is as follows:

```
http://www.seedlabhashlengthext.com:5000/?myname=<name>&uid=<need-to-fill>&lstcmd=1&m
ac=<need-to-calculate>
```

To send such a request, other than using our real names, we need to fill in the two missing arguments. Students need to pick a uid number from the *key.txt*in the *LabHome*directory. This file contains a list of colon-separated uid and key values. Students can use any uid and its associated key value. For example, students can use uid 1001 and its key 123456.

The second missing argument is the MAC, which can be calculated by concatenating the key with the contents of the requests R (the argument part only), with a colon added in between. See the following example:

```
Key:R = 123456:myname=JohnDoe&uid=1001&lstcmd=1
```

The MAC will be calculated using the following command:

```
$ echo -n "123456:myname=JohnDoe&uid=1001&lstcmd=1" | sha256sum
7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3 -
```

Terminal Cmd below was entered to calculate the Message Authentication Code:

```
[03/18/20]seed@VM:~/.../LabHome$ echo -n "123456:myname=JinJung&uid=1001&lstcmd=1&download=secret.txt" | sha
256sum
c262b265d8c421af2d24e28f43cb44c1b50906d0a5dc1cadfb79cf39d208eb88  -
```

We can then construct the complete request and send it to the server program using the browser:

```
http://www.seedlablenext.com:5000/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3
```

Task. Please send a download command to the server, and show that you can get the results back.

Entering the url below:

```
http://www.seedlablenext.com:5000/?
myname=JinJung&uid=1001&lstcmd=1&download=secret.txt&mac=c262b265d8c421af2d24e28f43cb44c1b50906d0a5dc1cadfb79cf39d208eb88
```

Yields:

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.

3.2 Task 2: Create Padding

To conduct the hash length extension attack, we need to understand how padding is calculated for one-way hash. The block size of SHA-256 is 64 bytes, so a message M will be padded to the multiple of 64 bytes during the hash calculation. According to RFC 6234, paddings for SHA256 consist of one byte of \x80, followed by a many 0's, followed by a 64-bit (8 bytes) length field (the length is the number of bits in the M).

Assume that the original message is M = *"This is a test message".* The length of M is 22 bytes, so the padding is 64 - 22 = 42 bytes, including 8 bytes of the length field. The length of M in term of bits is 22 * 8 = 176 = 0xB0. SHA256 will be performed in the following padded message:

```
"This is a test message"
"\x80"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\xB0"
```

It should be noted that the length field uses the Big-Endian byte order, i.e., if the length of the message is 0x012345, the length field in the padding should be:

```
"\x00\x00\x00\x00\x00\x01\x23\x45"
```

Task. Students need to construct the padding for the following message (the actual value of the <*key*>and <*uid*>should be obtained from the *LabHome/key.txt*file.

```
<key>:myname=<name>&uid=<uid>&lstcmd=1
```

```
[03/18/20]seed@VM:~/.../LabHome$ echo -n "123456:myname=JinJung&uid=1001&lstcmd=1" > msg.txt
[03/18/20]seed@VM:~/.../LabHome$ cat msg.txt
123456:myname=JinJung&uid=1001&lstcmd=1[03/18/20]seed@VM:~/.../LabHome$ ls -ld msg.txt
-rw-rw-r-- 1 seed seed 39 Mar 18 16:18 msg.txt
[03/18/20]seed@VM:~/.../LabHome$
```

64 Bytes = 39 byte msg + 1 byte 0x80 + 16 bytes repeating 0x00 + 7 bytes repeating 0x00 + 1 byte 0x27

The message is 39 bytes in size so we need to pad it with: 1 byte of 0x80 followed by 16 bytes of 0x00 and lastly 7 additional bytes of 0x00 and 1 byte 0x27

So the entire message with padding should be:

"123456:myname=JinJung&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\x00\x00\x00”
“\x27

3.3 Task 3: Compute MAC using Secret Key

In this task, we will add an extra message N = "Extra message" to the padded original message M = "This is a test message", and compute its hash value. The program is listed below.

```
/* calculate_mac.c */
#include <stdio.h> #include
<openssl/sha.h>

int main(intargc, const char *argv[])
{
      SHA256_CTX c; unsigned char
      buffer[SHA256_DIGEST_LENGTH]; inti;

      SHA256_Init(&c);
      SHA256_Update(&c,
            "This is a test message"
            "\x80"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
            "\x00\x00\x00"
            "\x00\x00\x00\x00\x00\x00\x00\xB0"
            "Extra message",
            64+13);
      SHA256_Final(buffer, &c);

      for(i = 0; i< 32; i++) {
          printf("%02x", buffer[i]);
      } printf("\n");
      return 0;
}
```

Students can compile and run the above program as follows:

```
$ gcccalculate_mac.c -o calculate_mac -lcrypto
$ ./calculate_mac
```

Task. Students should change the code in the listing above and compute the MAC for the following request (assume that we know the secret MAC key):

```
http://www.seedlablenext.com:5000/?myname=<name>&uid=<uid>&lstcmd=1<padding>&downlo
ad=secret.txt
&mac=<hash-value>
```

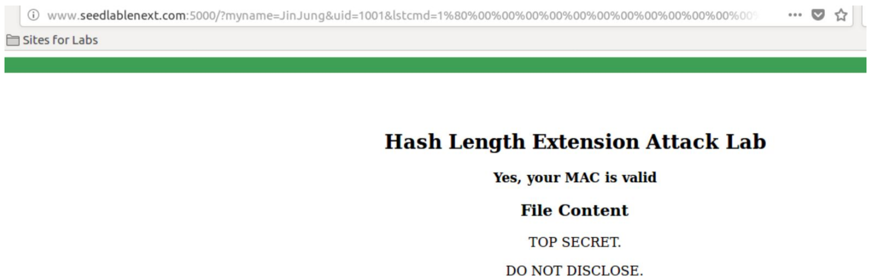
Just like the previous task, the value of <name>should be your actual name. The value of the <uid>and the MAC key should be obtained from the LabHome/key.txt file. Please send this request to the server, and see whether you can successfully download the secret.txt file.

It should be noted that in the URL, all the hexadecimal numbers in the padding need to be encoded by changing \x to %. For example, \x80 in the padding should be replaced with %80 in the URL above. On the server side, encoded data in the URL will be changed back to the hexadecimal numbers.

Modifying the calculate_mac.c file so that the correct mac will be calculated for http://www.seedlablenext.com:5000/?myname=<name>&uid=<uid>&lstcmd=1<padding>&download=secret.txt&mac =<hash-value> :

```c
/*calculate_mac.c*/
#include <stdio.h>
#include <openssl/sha.h>
#include <string.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=JinJung&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\x00\x00\x00"
       "\x27"
       "Extra message"
       "&download=secret.txt", 64 + 13 + 20);
    SHA256_Final(buffer, &c);

    for(i=0; i<32; i++) {
        printf("%02x", buffer[i]);
    }
    printf("\n");
    return 0;

}

/*http://www.seedlablenext.com:5000/?myname=JinJung&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%27Extra%
20message&download=secret.txt&mac=080e05f2330ed26f888bafea548e7d02ef972e8d1b163544db80f65d46d908d9*/
```

```
[03/18/20]seed@VM:~/.../server$ gcc calculate_mac.c -o calculate_mac -lcrypto
[03/18/20]seed@VM:~/.../server$ ./calculate_mac
080e05f2330ed26f888bafea548e7d02ef972e8d1b163544db80f65d46d908d9
[03/18/20]seed@VM:~/.../server$
```



3.4 Task 4: The Length Extension Attack

In the previous task, we show how a legitimate user calculates the MAC (with the knowledge of the MAC key). In this task, we will do it as an attacker, i.e., we do not know the MAC key. However, we do know the MAC of a valid request R. Our job is to forge a new request based on R, while still being able to compute the valid MAC.

Given the original message M="This is a test message" and its MAC value, we will show how to add a message "Extra message" to the end of the padded M, and then compute its MAC, without knowing the secret MAC key.

```
$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906
```

The program below can be used to compute the MAC for the new message:

```
/* length_ext.c */
#include <stdio.h>
#include <arpa/inet.h> #include
<openssl/sha.h>

int main(intargc, const char *argv[])
    { inti;
    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(0x6f343800);
    c.h[1] = htole32(0x1129a90c);
    c.h[2] = htole32(0x5b163792);
    c.h[3] = htole32(0x8bf38bf2);
    c.h[4] = htole32(0x6e39e57c);
    c.h[5] = htole32(0x6e951100);
    c.h[6] = htole32(0x5682048b);
    c.h[7] = htole32(0xedbef906);

    // Append additional message
    SHA256_Update(&c, "Extra message", 13); SHA256_Final(buffer, &c);

    for(i = 0; i< 32; i++) {
        printf("%02x", buffer[i]);
    } printf("\n");
    return 0;



}
```

Students can compile the program as follows:

```
$ gcclength_ext.c -o length_ext -lcrypto
```

Task. Students should first generate a valid MAC for the following request (where <uid>and the MAC key should be obtained from the LabHome/key.txt file):

```
http://www.seedlablenext.com:5000/?myname=<name>&uid=<uid>&lstcmd=1&mac=<mac>
```

So we want to generate a valid MAC for myname=<name>&uid=<uid>&lstcmd=1:

```
[03/18/20]seed@VM:~/.../server$ echo -n "123456:myname=JinJung&uid=1001&lstcmd=1" | sha256sum
0a1a4f8fb8c6dce9f781ff742bbd1042fbcd0c9194f52bb610a7b85b5bfa8a5d  -
```

Below is the calculated mac for the message "123456:myname=JinJung&uid=1001&lstcmd=1" <padding>:

```
[03/18/20]seed@VM:~/.../server$ gcc calculate_mac.c -o calculate_mac -lcrypto
[03/18/20]seed@VM:~/.../server$ ./calculate_mac
2d35b81b4c5df335f75b1fcc9441f0ab44f181f95db2dca752b7ae52565a9d5c
[03/18/20]seed@VM:~/.../server$
```

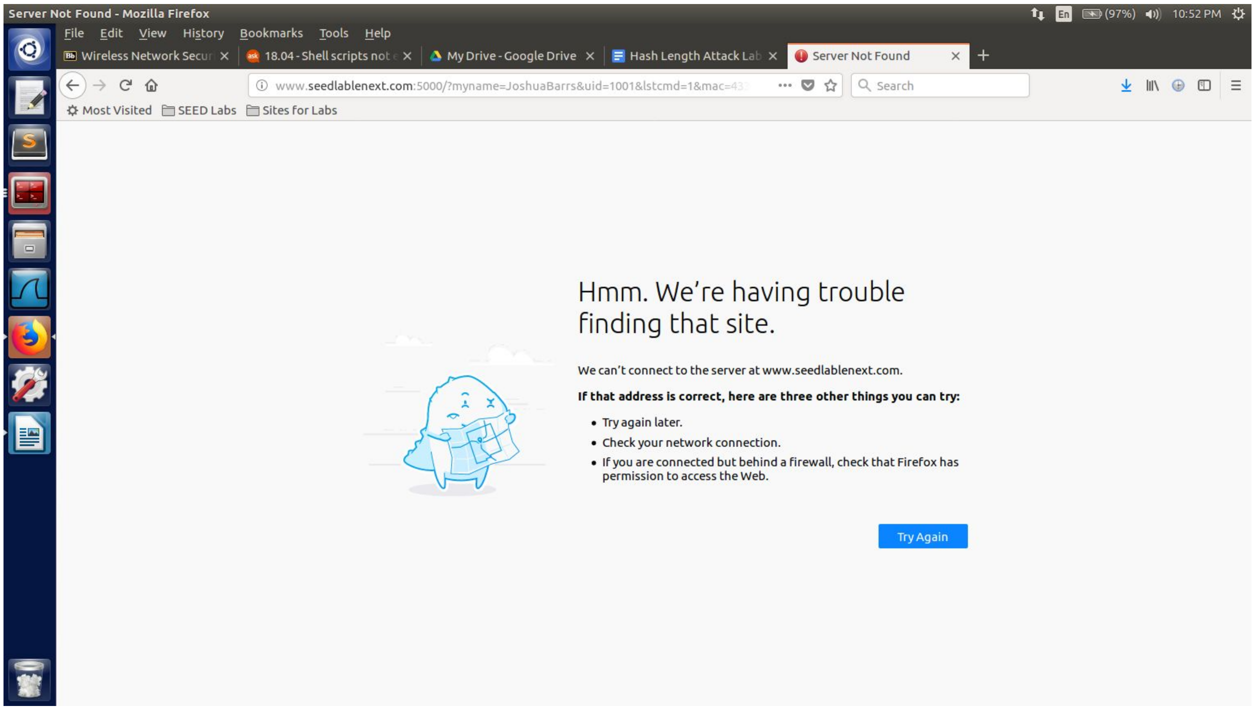
```c
/*length_ext.c*/

#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(0x2d35b81b);
    c.h[1] = htole32(0x4c5df335);
    c.h[2] = htole32(0xf75b1fcc);
    c.h[3] = htole32(0x9441f0ab);
    c.h[4] = htole32(0x44f181f9);
    c.h[5] = htole32(0x5db2dca7);
    c.h[6] = htole32(0x52b7ae52);
    c.h[7] = htole32(0x565a9d5c);
    //Append additional message
    SHA256_Update(&c, "Extra message"
    "&download=secret.txt"
    , 13 + 20);
    SHA256_Final(buffer, &c);

    for(i=0; i<32; i++) {
        printf("%02x", buffer[i]);
    }
    printf("\n");
    return 0;

}
```

```
[03/18/20]seed@VM:~/.../server$ gcc length_ext.c -o length_ext -lcrypto
[03/18/20]seed@VM:~/.../server$ ls
calculate_mac  calculate_mac.c  LabHome  length_ext  length_ext.c  run_server.sh  www
[03/18/20]seed@VM:~/.../server$ ./length_ext
9f6b5b733219f4a464f11569d63da0c9b5c6a297ae2c41e7d2d7ec72e613eede
[03/18/20]seed@VM:~/.../server$
```

Based on the <mac>value calculated above, please construct a new request that includes the download command. You are not allowed to use the secret key this time. The URL looks like below.

```
http://www.seedlablenext.com:5000/?myname=<name>&uid=<uid>&lstcmd=1<padding>&downlo
ad=secret.txt&mac=<new-mac>
```

Please send the constructed request to the server, and show that you can successfully get the content of the secret.txt file.



- The above image corresponds to me trying to access the server using the original MAC value (without the newly generated MAC). I could not seem to access the contents of the server with the generated MAC value. The server did not seem to recognize the command. This could possibly be due to an error in my MAC value or an error in the command.

3.5 Task 5: Attack Mitigation using HMAC

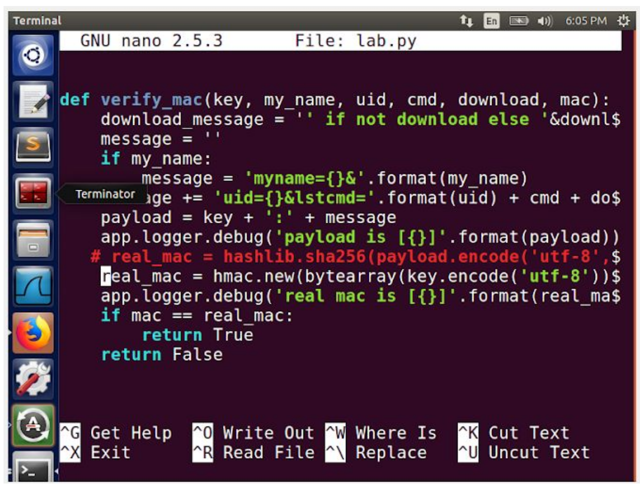
In the tasks so far, we have observed the damage caused when a developer computes a MAC in an insecure way by concatenating the key and the message. In this task, we will fix the mistake made by the developer. The standard way to calculate MACs is to use HMAC. Students should modify the server program's *verifymac*() function and use Python's hmacmodule to calculate the MAC. The function resides in lab.py. Given a key and message (both of type string), the HMAC can be computed as shown below:

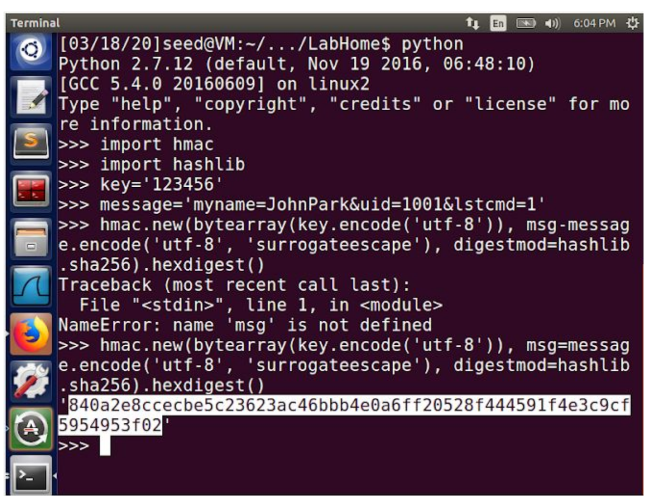
```
mac = hmac.new(bytearray(key.encode('utf-8')), msg=message.encode('utf-8',
          'surrogateescape'), digestmod=hashlib.sha256).hexdigest()
```

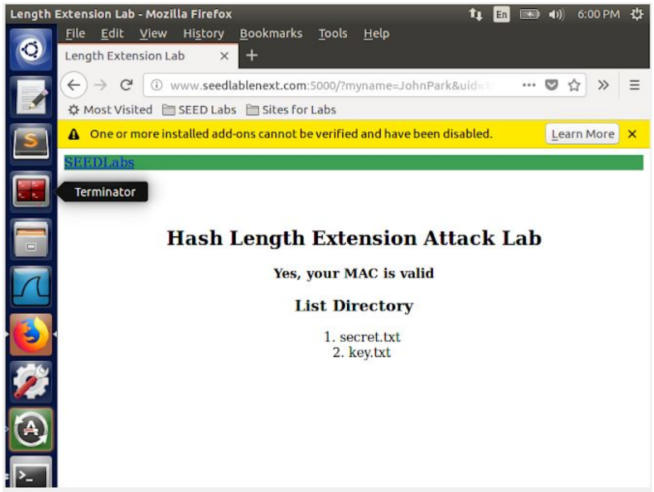
Students should repeat Task 1 to send a request to list files while using HMAC for the MAC calculation. Assuming that the chosen key is 123456, the HMAC can be computed in the following:

```
$ python
Python 3.5.2
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> import hmac
>>> import hashlib
>>> key='123456'
>>> message='lstcmd=1'
>>>hmac.new(bytearray(key.encode('utf-8')), msg=message.encode('utf-8',
    'surrogateescape'), digestmod=hashlib.sha256).hexdigest()
```

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







An attacker will fail MAC verification when the client and server are using HMAC because HMAC calculates MAC by hashing the key with the hashed value of the key and the message. In the more naive approach, the MAC is calculated by hashing the value of the key and the message only.

If the attacker has the inner hash value -- hash(key + message) -- and the client and server are using the naive approach, the attacker can implement a length-extension attack. However, the attacker does not have the outer hash value -- hash(key + hash(key + message)), and therefore cannot use length-extension.

4 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to explain the observations that are interesting or surprising.