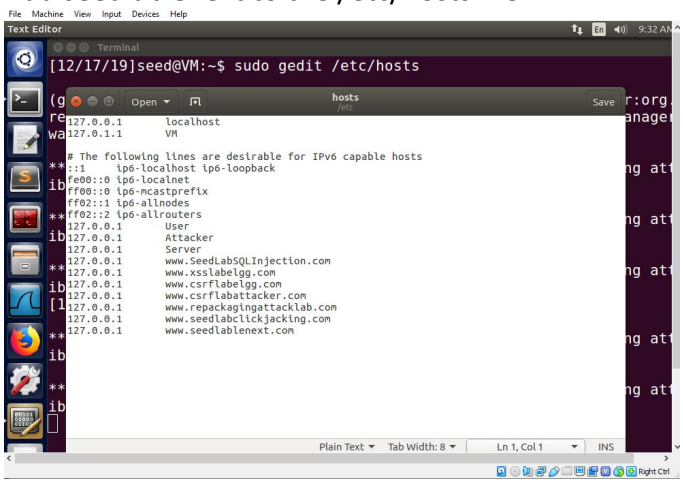
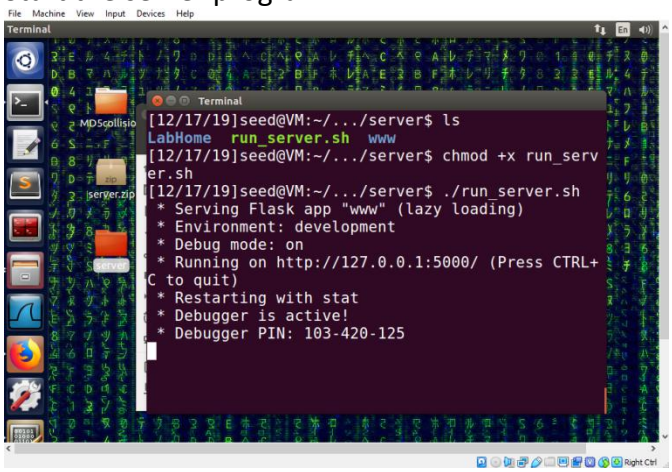
Hash Length Extension Attack Lab

Liangyu W

Add seedlablenext to the /etc/hosts file:

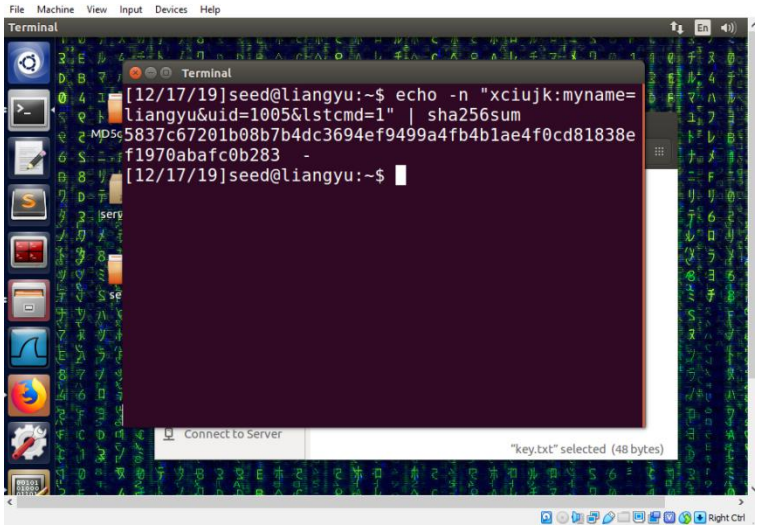


Start the server program

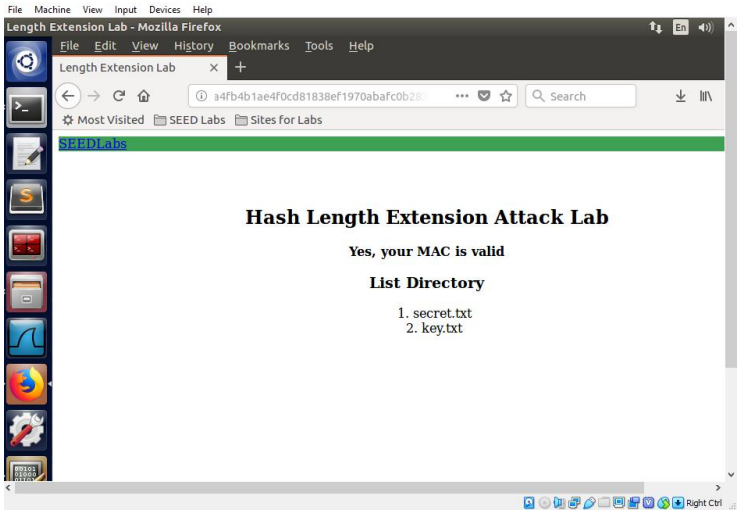


Task 1: Send Request to List Files

We choose uid 1005 and its corresponding key value xciujk from the LabHome directory.
Generate a MAC using the command:
echo -n "xciujk:myname=liangyu&uid=1005&lstcmd=1" | sha256sum

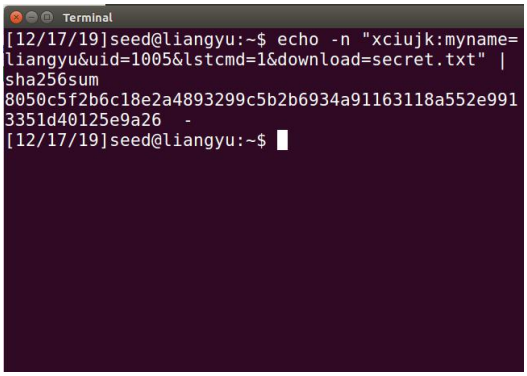


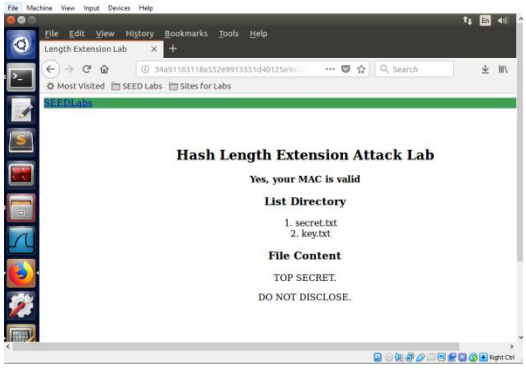
We send the following request to the server:
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&lstcmd=1&mac=5837c67201b08b7b4dc3694ef9499a4fb4b1ae4f0cd81838ef1970abafc0b283



The server verifies the MAC and lists the files in its directory.

Similarly, we use the following request to download file from the server:
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&lstcmd=1&download=secret.txt&mac=8050c5f2b6c18e2a4893299c5b2b6934a91163118a552e9913351d40125e9a26



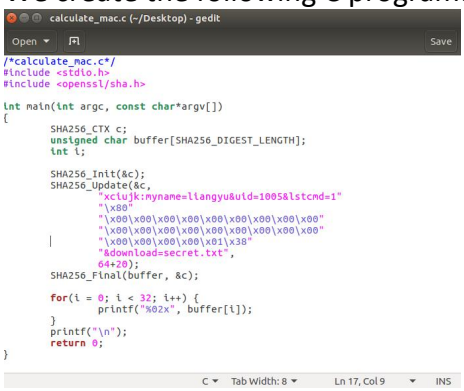


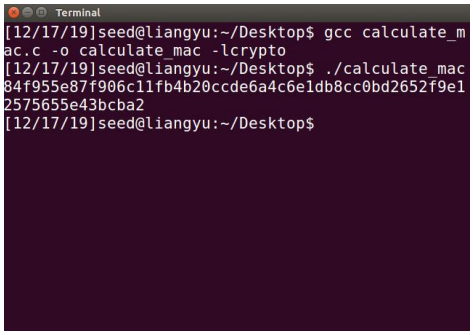
Task 2: Create Padding

The message “xciujk:myname=liangyu&uid=1005&lstcmd=1” has a length of 39. So 25 bytes need to be padded.
Thus for “xciujk:myname=liangyu&uid=1005&lstcmd=1”, the padding is:
“\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\x27”

Task 3: Compute MAC using Secret Key

We create the following C program:

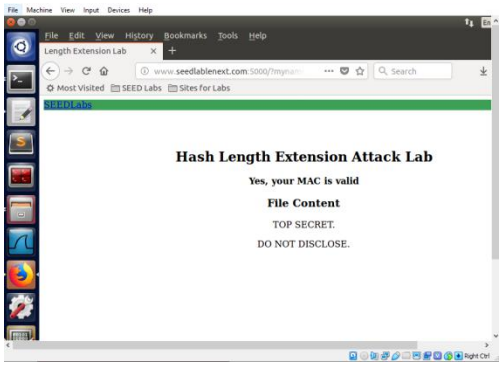




Compile and running the program generates our MAC.

We send the following request to the server:
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&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%00%01%38&download=secret.txt&mac=84f955e87f906c11fb4b20ccde6a4c6e1db8cc0bd2652f9e12575655e43bcba2



Task 4: The Length Extension Attack

We previously generated the valid MAC
5837c67201b08b7b4dc3694ef9499a4fb4b1ae4f0cd81838ef1970abafc0b283
For the request
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&lstcmd=1

We create the following length_ext.c program, it computes a new MAC based on the previously generated MAC and the added message:

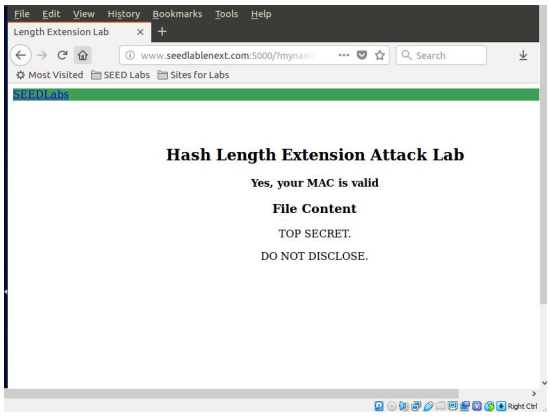
```c
/*length_ext.c*/
#include <stdio.h>
#include <arpa/inet.h>
#include <openssl/sha.h>

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(0x5837c672);
        c.h[1] = htole32(0x01b08b7b);
        c.h[2] = htole32(0x4dc3694e);
        c.h[3] = htole32(0xf9499a4f);
        c.h[4] = htole32(0xb4b1ae4f);
        c.h[5] = htole32(0x0cd81838);
        c.h[6] = htole32(0xef1970ab);
        c.h[7] = htole32(0xafc0b283);

        // 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;
}
```

Compiling and running the program gives us the new MAC:
84f955e87f906c11fb4b20ccde6a4c6e1db8cc0bd2652f9e12575655e43bcba2

```
Terminal
[12/17/19]seed@liangyu:~/Desktop$ gcc length_ext.
c -o length_ext -lcrypto
[12/17/19]seed@liangyu:~/Desktop$ ./length_ext
84f955e87f906c11fb4b20ccde6a4c6e1db8cc0bd2652f9e1
2575655e43bcba2
[12/17/19]seed@liangyu:~/Desktop$
```

Using the new MAC we construct the following request:

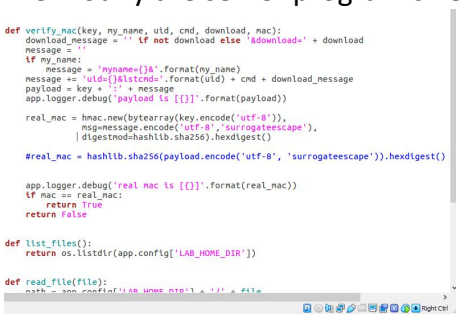
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&lstcmd=1%80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%38&download=secret.txt&mac=84f955e87f906c11fb4b20ccde6a4c6e1db8cc0bd2652f9e12575655e43bcba2



This allows us to successfully download the content of the file without knowing the secret key. But in order to construct the padding, we had to know the length of the secret key.

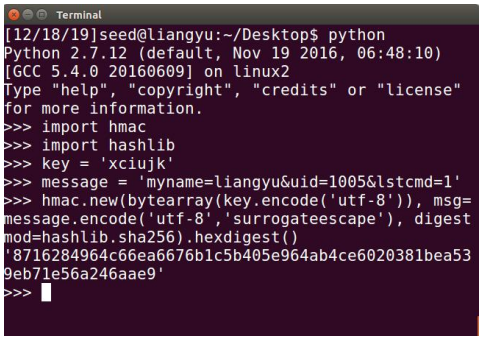
Task 5: Attack Mitigation using HMAC

We modify the server program's verify_mac() function as follows:



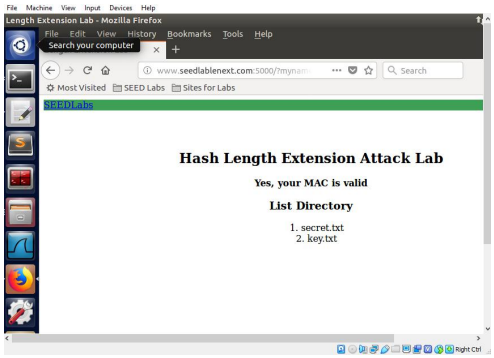
```
def verify_mac(key, my_name, uid, cmd, download, mac):
    download_message = '' if not download else '&download=' + download
    message = ''
    if my_name:
        message = 'myname={}&'.format(my_name)
    message += 'uid={}&lstcmd='.format(uid) + cmd + download_message
    payload = key + ':' + 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()

    #real_mac = hashlib.sha256(payload.encode('utf-8', 'surrogateescape')).hexdigest()


    app.logger.debug('real mac is [{}]'.format(real_mac))
    if mac == real_mac:
        return True
    return False


def list_files():
    return os.listdir(app.config['LAB_HOME_DIR'])


def read_file(file):
```

The hash calculated using HMAC is
8716284964c66ea6676b1c5b405e964ab4ce6020381bea539eb71e56a246aae9



```
Terminal
[12/18/19]seed@liangyu:~/Desktop$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>> import hmac
>>> import hashlib
>>> key = 'xciujk'
>>> message = 'myname=liangyu&uid=1005&lstcmd=1'
>>> hmac.new(bytearray(key.encode('utf-8')), msg=
message.encode('utf-8','surrogateescape'), digest
mod=hashlib.sha256).hexdigest()
'8716284964c66ea6676b1c5b405e964ab4ce6020381bea53
9eb71e56a246aae9'
>>>
```

Construct the new request using HMAC as follows:
http://www.seedlablenext.com:5000/?myname=liangyu&uid=1005&lstcmd=1&mac=8716284964c66ea6676b1c5b405e964ab4ce6020381bea539eb71e56a246aae9



Server successfully verifies the new HMAC.

The original MAC is calculated by hashing the key concatenated to the message: MAC = hash(key + message), HMAC is calculated using two rounds of hashing: HMAC = hash(key + hash(key + message)). This prevents the length extension attack demonstrated above.