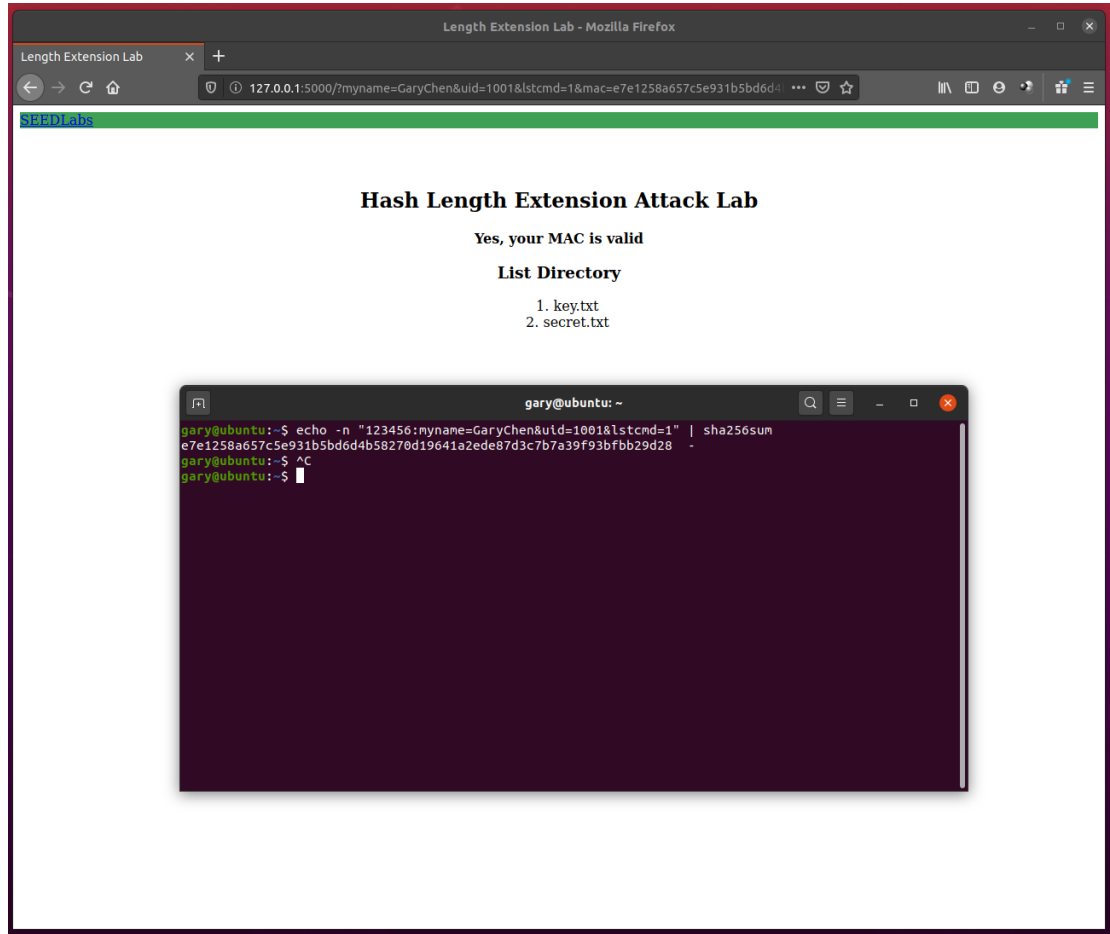


資訊安全 LAB

40647027S 陳冠穎

Task1:



Task2:

123456:myname=GaryChen&uid=1001&lstcmd=1: 40 Bytes

$40 * 8 = 320 \text{ bits} = 0x140 \text{ bits}$

"123456:myname=GaryChen&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\x01\x40"

Task3:

MAC: 5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc

Request:

http://127.0.0.1:5000/?myname=GaryChen&uid=1001&lscmd=1%80%01%40&download=secret.txt&mac=5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc

The screenshot shows a development environment with a code editor, a terminal, and a web browser. The code editor contains a C program named `calculate_mac.c` that calculates a SHA256 MAC for a given request. The terminal shows the execution of the program, which outputs the MAC value `5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc`. The web browser shows the output of the request, which is a page titled "Hash Length Extension Attack Lab" with the message "Yes, your MAC is valid" and the file content "TOP SECRET. DO NOT DISCLOSE."

```
1 /* calculate_mac.c */
2 #include <stdio.h>
3 #include <openssl/sha.h>
4
5 int main(int argc, const char *argv[])
6 {
7     SHA256_CTX c;
8     unsigned char buffer[SHA256_DIGEST_LENGTH];
9     int i;
10    SHA256_Init(&c);
11    SHA256_Update(&c,
12                 "123456:myname=GaryChen&uid=1001&lscmd=1"
13                 "\x80"
14                 "\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
15                 "\x00\x00\x00\x00\x00"
16                 "\x00\x00\x00\x00\x00\x01\x40"
17                 "download=secret.txt",
18                 64 + 20);
19    SHA256_Final(buffer, &c);
20    for (i = 0; i < 32; i++)
21    {
22        printf("%02x", buffer[i]);
23    }
24    printf("\n");
25    return 0;
26 }
```

```
ary@ubuntu:~/Downloads$ cd Downloads/
ary@ubuntu:~/Downloads$ ls
calculate_mac  code_1.45.0-158866285_and64.deb  pycharm-professional-2020.1.1.tar.gz  server  server.zip  test.py  Untitled-3.c
ary@ubuntu:~/Downloads$ gcc Untitled-3.c -o calculate_mac -lcrypto
ary@ubuntu:~/Downloads$ ./calculate_mac
cf683b9c5cc427d1e122221d13e7ab1f9b334087d51301dab67af666bd3970e6
ary@ubuntu:~/Downloads$ ./calculate_mac
5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc
ary@ubuntu:~/Downloads$
```

Task4:

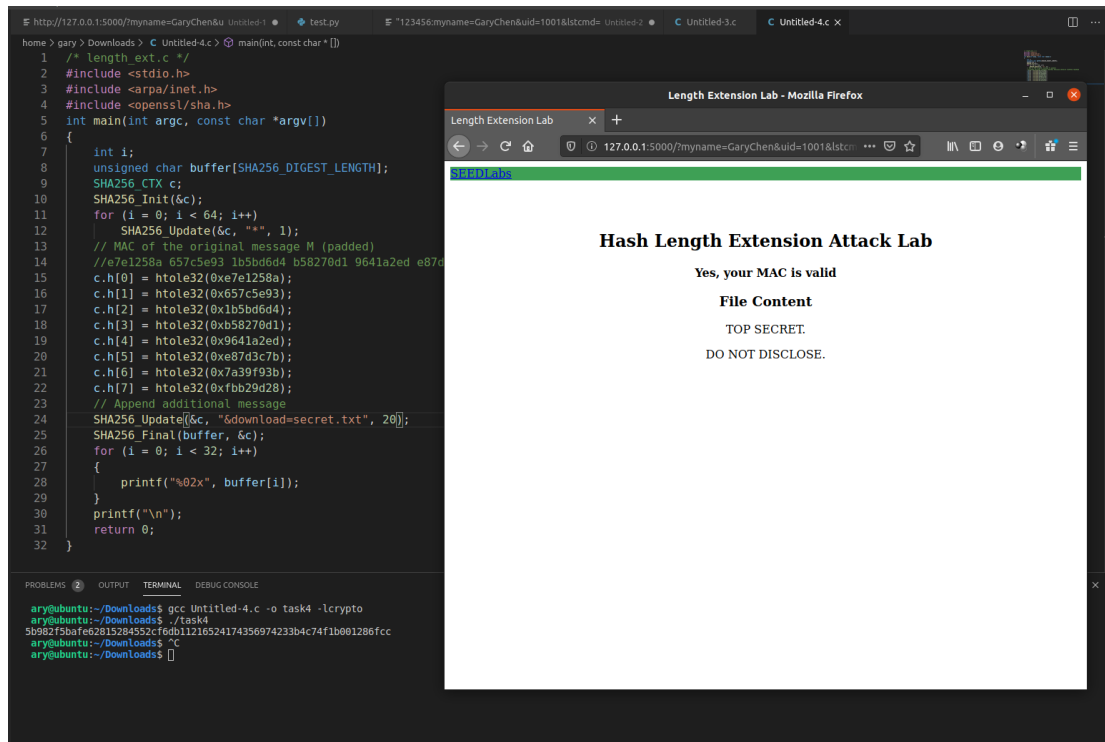
算出來的 MAC 會跟 Task3 知道 Key 之下算出來的一樣。

MAC: 5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc

Request:

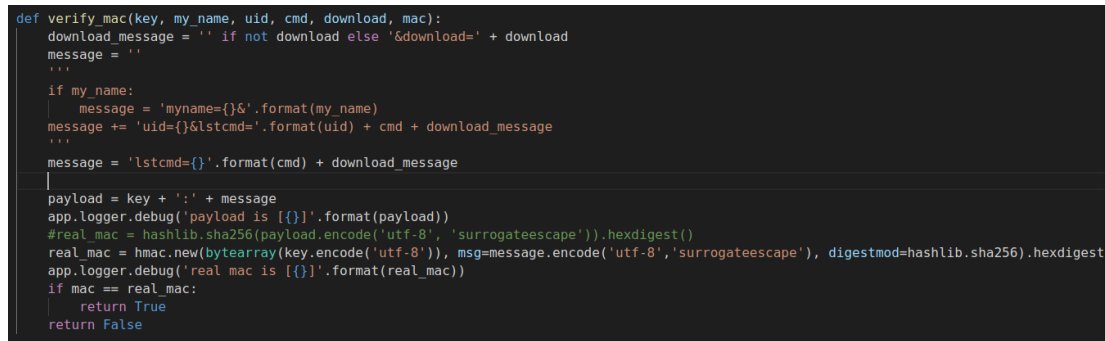
http://127.0.0.1:5000/?myname=GaryChen&uid=1001&lscmd=1%80%01%40&download=secret.txt&mac=5b982f5baf62815284552cf6db11216524174356974233b4c74f1b001286fcc

也就是說我只要知道原本合法 request 與 MAC，再把 padding 算出來，之後再後面加別的指令也可以在不知道 key 的情況下算出合法的 MAC。

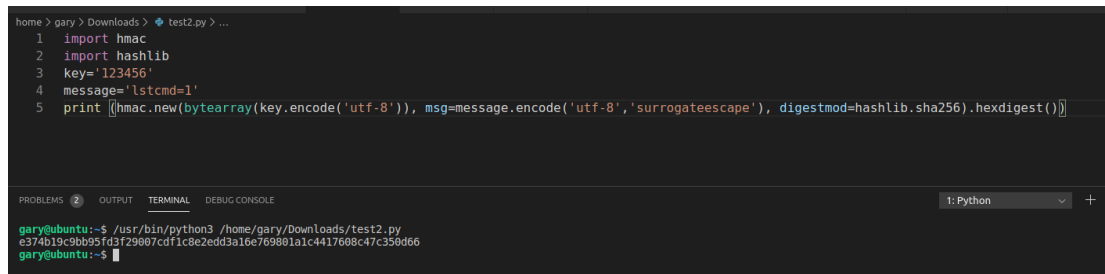


Task5:

將 verify_mac 更改如下

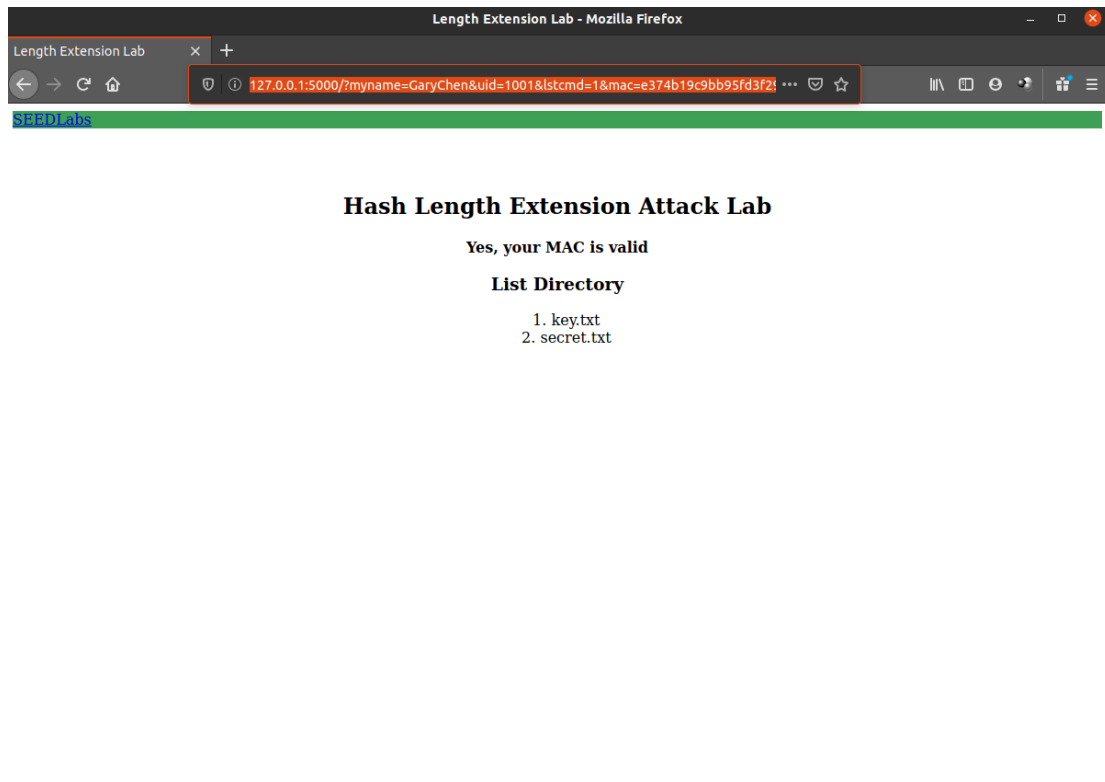


使用新的方法算出 MAC



Request:

<http://127.0.0.1:5000/?myname=GaryChen&uid=1001&lscmd=1&mac=e374b19c9bb95fd3f29007cdf1c8e2edd3a16c769801a1c4417608c47c350d66>



成功。

因為 HMAC 的作用方式會先將 key+message 做一次 hash 後，再做第二次的 hash，因此攻擊者無法再用長度擴展攻擊任意更改 message。