Hash Length Extension Attack Lab Report

Hayden Eubanks

School of Business, Liberty University

CSIS 463-B01

Dr. De Queiroz

September 30, 2023

Hash Length Extension Lab Report

Introduction:

One of the applications of hashing algorithms in modern cryptography can be seen in the verification of message integrity (Upadhyay et al., 2022). This message integrity is often applied through message authentication codes (MAC) which can be appended to the end of a message and compared through hashing against the received message hash (Al-Odat, Khan, & Al-Qtiemat, 2022). If the hashed values are the same, a high degree of certainty can be placed on the fact that the message has not been altered. However, some hashing algorithms based on the Merkle Damgård (MD) framework such as MD4, MD5, SHA1, SHA2, and SHA256 have been discovered to be vulnerable to length extension attacks due to the way they perform block padding (Al-Odat, Khan, & Al-Qtiemat, 2022). This vulnerability is a serious concern for security professionals as it allows a malicious actor to append information to the end of a message without knowing the original message's contents and then produce the correct hash value for the new message (Cortez, 2020). The consequences of a length extension attack could then result in the receiver believing that they can verify that a message has not been altered when in fact the message has been intercepted and data appended to it (Upadhyay et al., 2022). Understanding the implications of this vulnerability then highlights the importance of the risk of length extension attacks to be mitigated, and one implementation to accomplish this is the use of hash-based message authentication codes (HMACs) (Cortez, 2020). Exploring the dangers of improper MAC usage and how they can be mitigated through HMACs can then allow a security professional to understand the risks associated with length extension attacks as well as implementations to mitigate these attacks in their work promoting message integrity.

To examine the dangers posed by extension attacks, an understanding of how they are performed must first be obtained. Length extension attacks of hashing algorithms are a form of man-in-the-middle attack where a malicious actor intercepts a message and its hashed value and then appends data to the end of the message before rehashing to a new and verifiable hashed value (Upadhyay et al., 2022). To accomplish a length extension attack, a malicious actor would need access to the original hashed value, the length of the initial message, and the hashing algorithm used (Cortez, 2020). Notably, this list does not include the secret key used for encryption highlighting a vulnerability of extreme concern. Further, the fact that this risk exists in several notable hashing algorithms denotes a lack of compliance with one of the core principles of hashing being the avalanche effect. The avalanche effect is a principle for secure hashing that states that small changes to a message's input should be representative of a massive alteration of output (Upadhyay et al., 2022), similar to an avalanche escalating in variance as it rolls down a hill. Length extension attacks can then be seen to exploit this principle as the appending of a message will still compute up to the hash of the initial message before passing that hash to continuing steps for correctly hashing the remaining message elements (Cortez, 2020). This appending of messages in length extension attacks can then be seen to result from the way that messages are padded and processed within vulnerable hashing algorithms (Al-Odat, Khan, & Al-Qtiemat, 2022). The relationship between length extension attacks and the padding of blocks is central to understanding the vulnerability and as such has led to this attack also being referred to as a padding attack (Upadhyay et al., 2022). For example, to create a MAC value a message is created and passed to a hashing algorithm where padding is added to create blocks of the correct size (Basta, 2018). These blocks are then processed one by one up until the end block. This means that the processing of blocks through the end of the message would be identical to the processing of blocks up until the start of the appended message in an altered form (Al-Odat, Khan, & Al-Qtiemat, 2022). A malicious attacker must then simply process the blocks of appended data with the given information to successfully produce a hashed value of the new message.

While length extension attacks pose a serious threat to MAC value integrity, several implementations exist to combat this risk and ensure that hashed values can be used to authenticate that a message has not been altered. One of the most notable implementations to accomplish this task can be seen through the switch from MAC to HMAC implementations (Chawla, 2021). HMACs function by taking the original message as well as the secret key used in encryption and hash them together to get the new HMAC value (Upadhyay et al., 2022). The primary difference between HMACs and signatures can then be seen in the fact that HMACs use private key encryption while signatures use public (Chawla, 2021). HMACs provide added security over their MAC counterparts by hashing the key and the message separately, resulting in a message that cannot be appended to without knowing the secret key (Chawla, 2021). The key is used in this process to initially append a series of bits to the front of the message and as the blocks are processed in order, the appending of this key will affect the hash of the entire message. This allows the hash to better comply with the avalanche effect principle and protects data from length extension attacks where the attacker does not know the secret key. Further, as long as these secret keys remain uncompromised, the hashed value can also seek to provide non-repudiation as only validated users should have access to the shared private key (Chawla, 2021). From this, the vast improvement of HMAC over MAC can be observed and a security professional can be better equipped to implement HMAC hashes for increased data security and message integrity.

In this lab, length extension attacks are explored to give the user an understanding of the importance of securing machine codes. The lack of security regarding these codes could result in man-in-the-middle attacks being performed where a malicious actor can append contents to the end of a message while maintaining a valid hashed code (Upadhyay et al., 2022). In order to explore these attacks, the core pieces of information needed to perform the attack are examined within the context of an attack allowing the user to gain an overview of how a length extension attack is performed. This begins by first providing an examination of calculating a MAC from an input message. After this, the next element being message padding is explored to understand the role played by block buffering in the SHA256 algorithm. With this knowledge, a mock length extension attack can be performed to append data to the end of the message passed to the server. This lab uses the example of appending codes sent to the server to get the server to perform functionality unintended by the original sender. This highlights the dangers associated with length extension attacks and as such points to the importance for a security professional to cover associated vulnerabilities. This then leads to the final portion of the lab where the mitigation strategy of implementing HMAC over MAC is examined and the benefits explored. Using HMAC involves including a private key to encryption to better secure the hashed values and as such increases the security of the message and mitigates the ability for length extension to occur (Chawla, 2021). Mitigation of risk and the covering of vulnerabilities are primary goals for security professionals, and through examining attack vectors such as length extension attacks, a security professional can be better equipped to implement security using hashed values within a system.

Lab Procedure:

The first task of this lab involved sending requests to the server and examining the effects of sending requests with valid and invalid MAC values. The server uses the MAC value to validate message integrity and as such, a message with an incorrect or missing MAC value will result in a failure of command execution. To generate a valid MAC value, the file of possible user IDs and key pairs must be examined (Screenshot 1) and this data used to update the fields of the message to be sent. This code could then be sent to the SHA256 hashing algorithm (Screenshot 2) and this hashed value could then be used as the MAC value for the message. Sending this message to the server will then result in a verification of message integrity and the contents of the server will be displayed (Screenshot 3). This same procedure can then be performed for the download command by first generating a MAC value for the command (Screenshot 4) and then sending the command to the server to download the intended file (Screenshot 5).

The next lab task then instructed the user to explore the implementation of block padding within the SHA256 algorithm and the impact that this padding plays on length extension attacks. To begin this task, a new message value was generated (Screenshot 6) which served as the basis for calculating the message padding. In SHA 256, the messages are processed in blocks of 64 bytes in size (Al-Odat, Khan, & Al-Qtiemat, 2022) with the original message only taking 46 bytes of that space. This means that 17 bytes of padding will be added to the message. Further, the length of the initial method also determines the final eight bytes of the padding which will read “\x00 \x00 \x00 \x00 \x00 \x00 \x01 \x70” representative of the original message’s size. The remaining padded bytes were then filled with the hexadecimal value of “\x00”. Appending the padding to the message then results in the final hexadecimal representation of the padded value

(Screenshot 7). However, each hexadecimal prefix of “\x” must be replaced with the percent symbol “%” to ensure that it can be passed to the browser with commands. The padding is an essential aspect of length extension attacks as the padding value records the length of the message it pads (Upadhyay et al., 2022). To accomplish a length extension attack, a new padding value must be generated and appended to the message with the updated message length included. This highlights the importance for a security professional to understand the role of padding in MAC values and gives insight into the mitigation of length extension attacks.

After this, the lab then directed the user toward the simulation of a length extension attack to gain insight into how these attacks are performed. To accomplish this, a new test message was generated and hashed with the SHA256 algorithm to create a MAC value for the initial message as would be sent from an initial sender (Screenshot 8). A program could then be used to append additional data onto this message by processing each block present in the original message and then running the update function to add on the new blocks present from the additional data (Screenshot 9). When compiling this code, several warnings of the deprecation of the SHA256 functions will be presented as these functions have been deprecated due to security concerns (Screenshot 10) further highlighting the importance of exploring attacks such as the length extension attack on SHA256. Executing this code then returns the updated MAC value for the new extended message (Screenshot 11). With a simple example performed, another extension attack could then be performed to be sent to the server. To begin this, the original server message could be created, and the MAC value generated for that message (Screenshot 12). The program could then be updated, including the updated values for the message blocks and the addition of the download command added as the extended message (Screenshot 13). Again, this code could be compiled (Screenshot 14) and executed to generate the new MAC value from the extended message (Screenshot 15). Passing the extended message with the new MAC value should then allow the download command to be executed on the server successfully performing a length extension attack. In my attempt at this lab step, I was unable to get the download command to execute correctly (Screenshot 16), but the new MAC value was generated and appended to the message sent to the server to accomplish the download.

The final lab task then challenged the user with exploring the mitigation provided from witching the MAC hashed values to HMAC values. The first step of this task involved accessing the verify_mac function for the server which is responsible for verifying the MAC value passed to the server (Screenshot 17). The code for this function was then changed to its HMAC counterpart (Screenshot 18) before rebuilding the server from the updated files (Screenshot 19). As an example, this code can be executed to produce a hash value and prove that the function has been implemented correctly (Screenshot 20). A new program could then be created to generate HMAC values (Screenshot 21) and the steps of task one repeated to prove the increased security provided by HMAC values. Rehashing the value according to the steps of task one (Screenshot 22) and then passing the value to the server will then show that access to the server is denied (Screenshot 23) highlighting the increased security of HMAC values. HMAC values provide additional security over MAC values as HMAC encryption includes a private key value in the encryption process (Chawla, 2021). This means that a malicious actor without access to the private key would then be unable to perform an extension attack while still generating the correct hashed value. This is a major improvement in security over the sole use of MAC values highlighting the importance for a security professional to understand HMAC implementations for the mitigation of length extension attacks. Length extension attacks pose a major threat to the integrity of data and a security professional should take great care in implementations using MAC values to ensure that integrity is maintained in light of potential attacks.

References

Al-Odat, Z. A., Khan, S. U., & Al-Qtiemat, E. (2022). A modified secure hash design to circumvent collision and length extension attacks. *Journal of Information Security and Applications*, *71*, 103376. https://doi.org/10.1016/j.jisa.2022.103376

Basta, A. (2018). *Oriyano, cryptography: Infosec pro guide.* McGraw-Hill Education. https://bookshelf.vitalsource.com/reader/books/9781307297003/pageid/14

Chawla, A. (August 31, 2021). *What is HMAC? (Hash based message authentication code)?*. Geeks for Geeks. https://www.geeksforgeeks.org/what-is-hmachash-based-message-authentication-code/

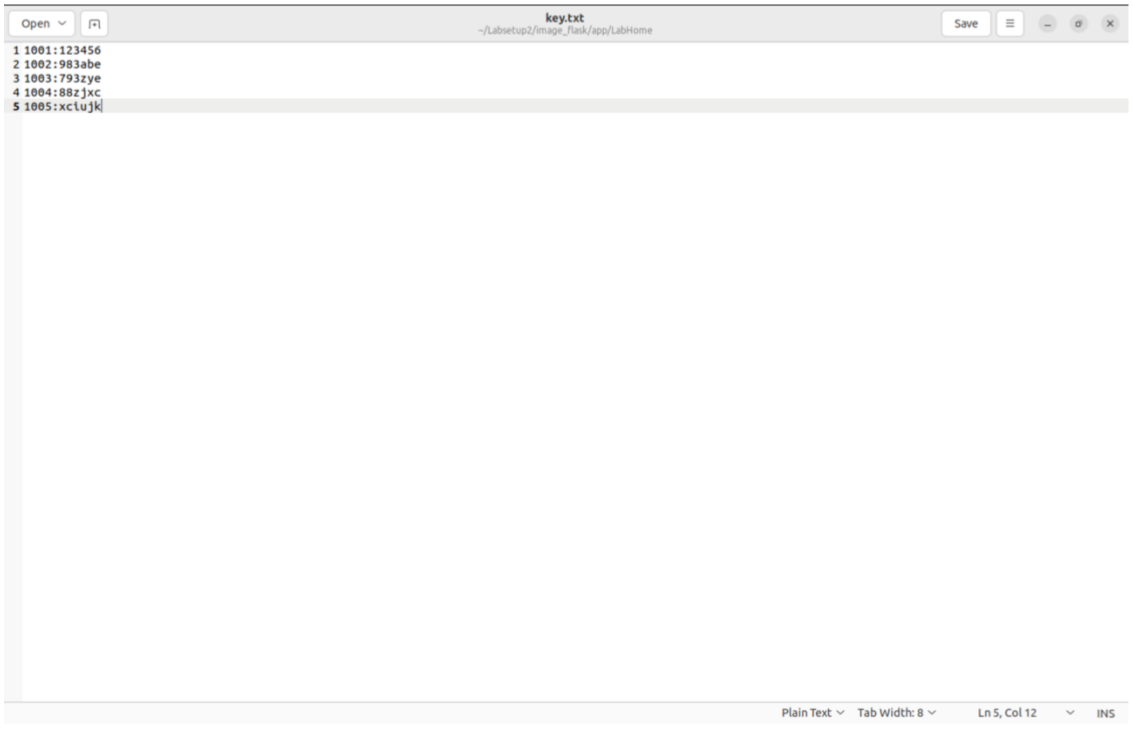
Cortez, D. M. A., Sison, A. M., & Medina, R. P. (April, 2020). Cryptographic randomness test of the modified hashing function of SHA256 to address length extension attack. *Proceedings of the 2020 8th International Conference on Communications and Broadband Networking* (pp. 24-28). https://dl.acm.org/doi/abs/10.1145/3390525.3390540

Python. (n.d.). *hmac- Keyed-hashing for message authentication.* Python. https://docs.python.org/3/library/hmac.html

Upadhyay, D., Gaikwad, N., Zaman, M., & Sampalli, S. (2022). Investigating the avalanche effect of various cryptographically secure hash functions and hash-based applications. *IEEE Access, 10*, 112472-112486. https://doi.org/10.1109/ACCESS.2022.3215778

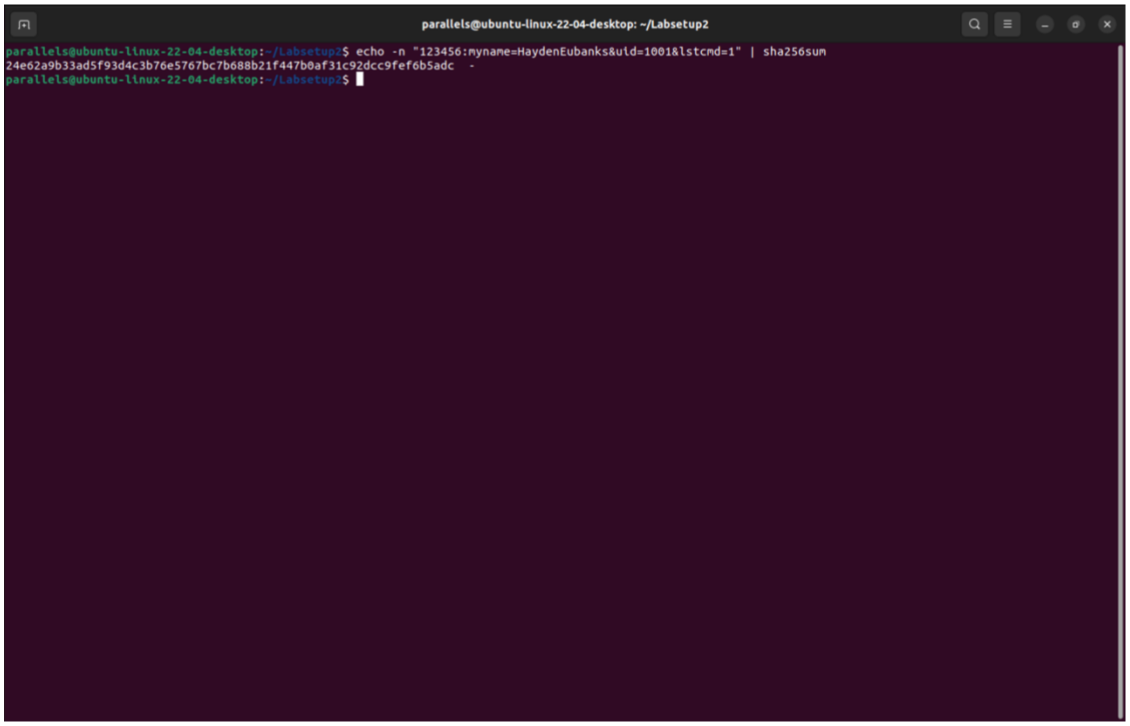
Screenshots: Task 1

Screenshot1: (Return to text)



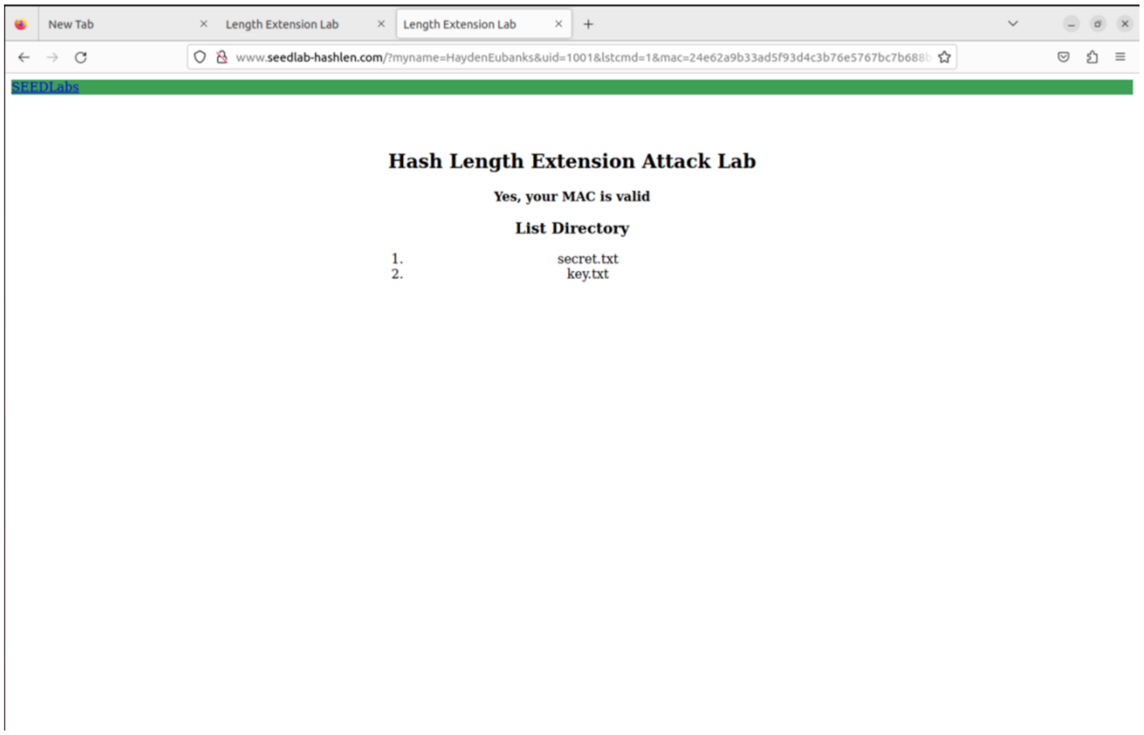
```
1001:123456
1002:983abe
1003:793zye
1004:88zjxc
1005:xciujk
```

Screenshot2: (Return to text)



```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ echo -n "123456:myname=HaydenEubanks&uid=1001&lstcmd=1" | sha256sum
24e62a9b33ad5f93d4c3b76e5767bc7b688b21f447b0af31c92dcc9fef6b5adc  -
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ 
```

Screenshot3: (Return to text)



```
www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1&mac=24e62a9b33ad5f93d4c3b76e5767bc7b688b
```

SEEDLabs

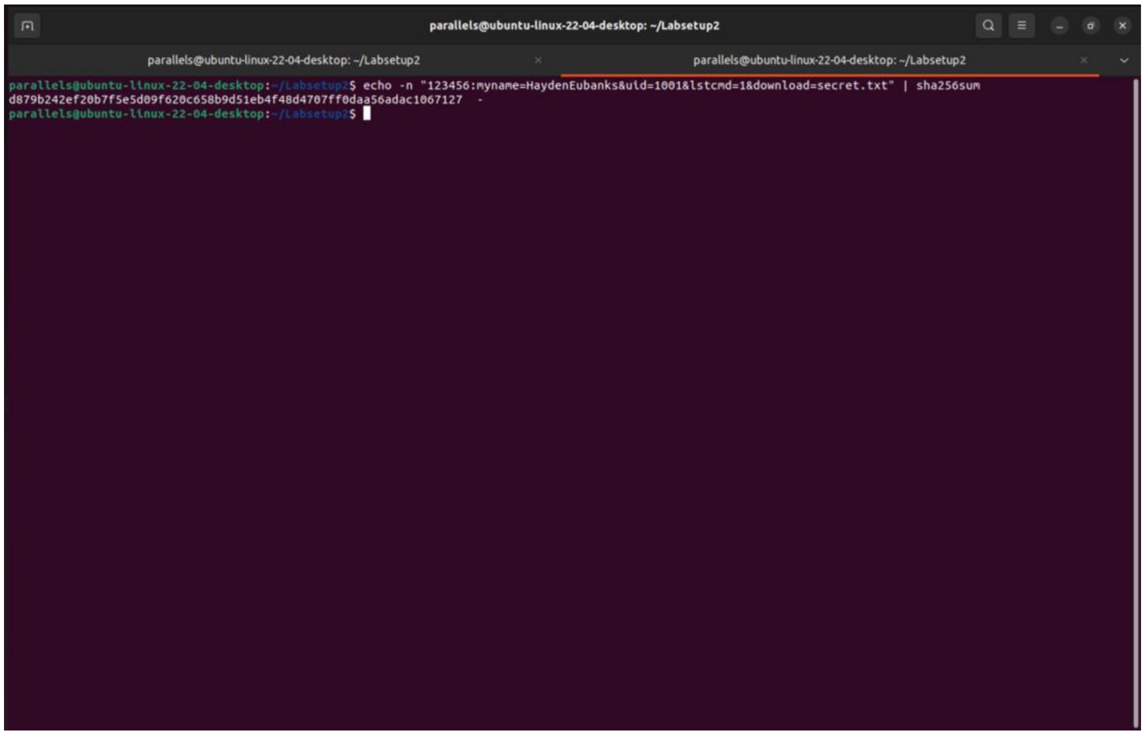
Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

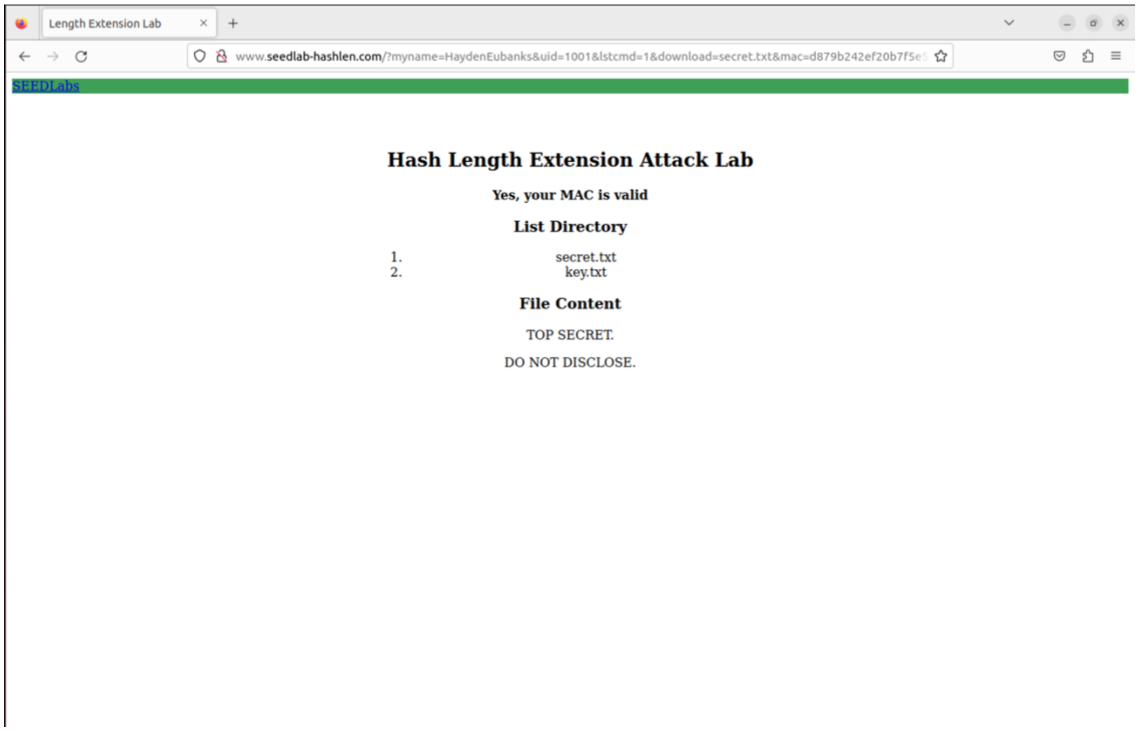
Screenshot4: (Return to text)



parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2

```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ echo -n "123456:myname=HaydenEubanks&uid=1001&lstcmd=1&download=secret.txt" | sha256sum
d879b242ef20b7f5e5d09f620c658b9d51eb4f48d4707ff0daa56adac1067127  -
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ 
```

Screenshot5: (Return to text)



www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1&download=secret.txt&mac=d879b242ef20b7f5e

SEEDLabs

Hash Length Extension Attack Lab

Yes, your MAC is valid

List Directory

1. secret.txt
2. key.txt

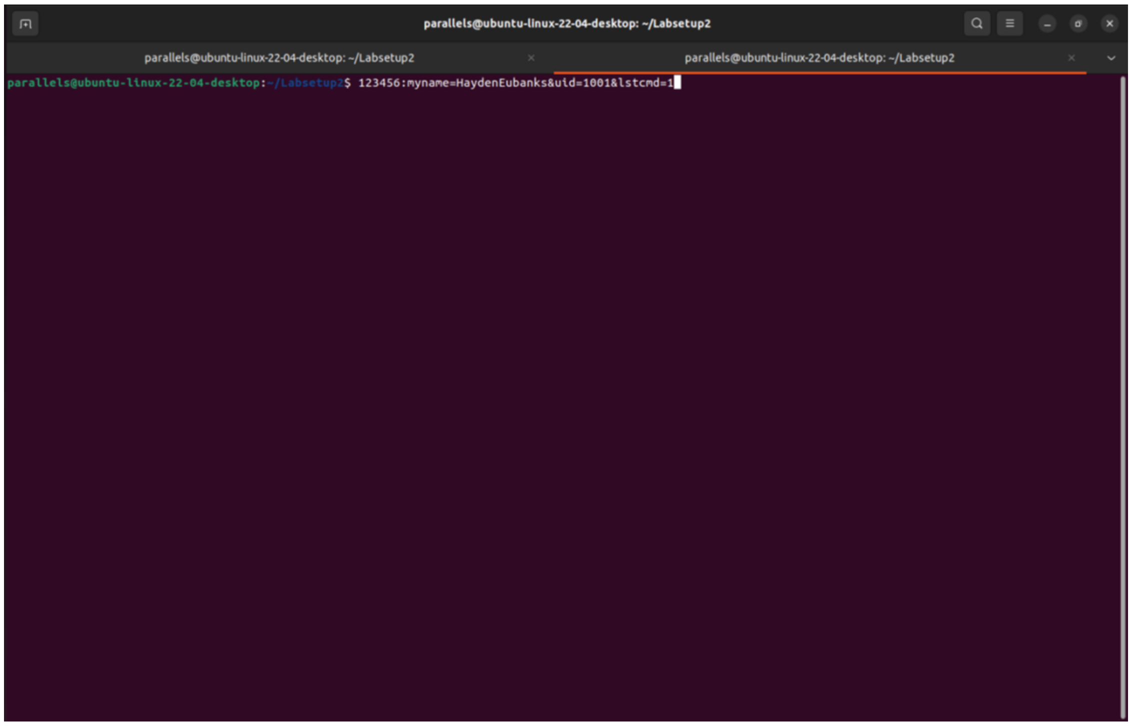
File Content

TOP SECRET.

DO NOT DISCLOSE.

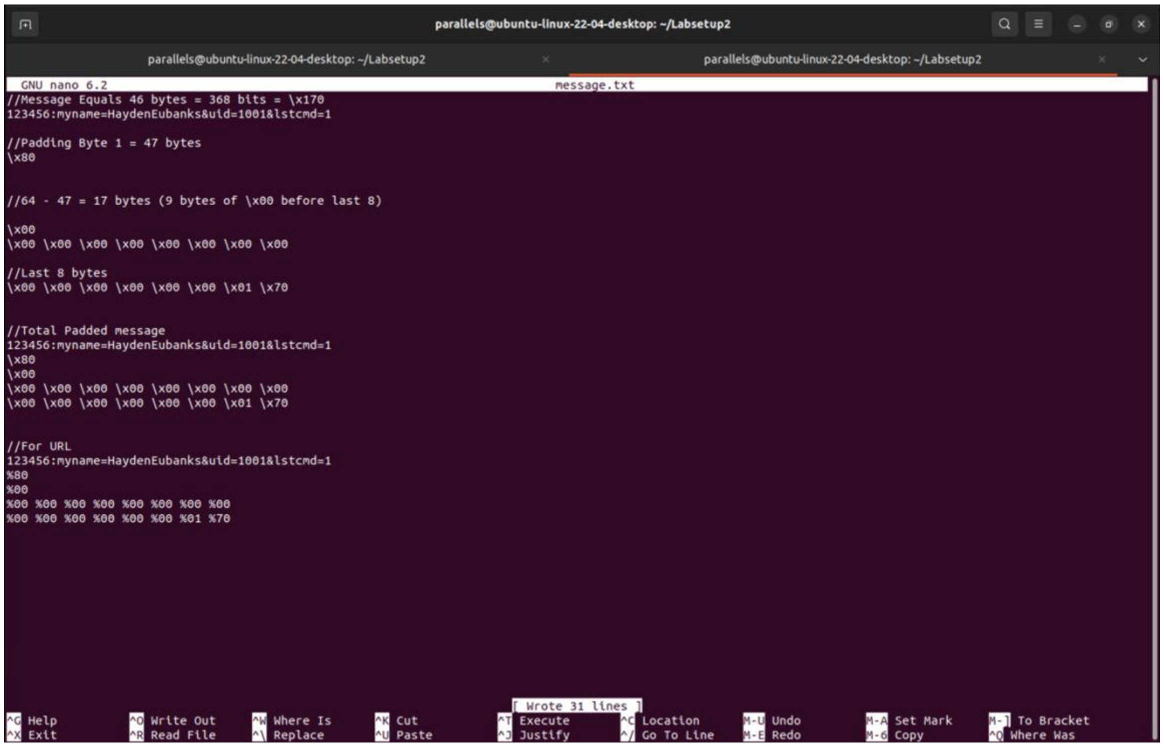
Screenshots: Task 2

Screenshot6: (Return to text)


parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2

```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ 123456:myname=HaydenEubanks&uid=1001&lstcmd=1
```

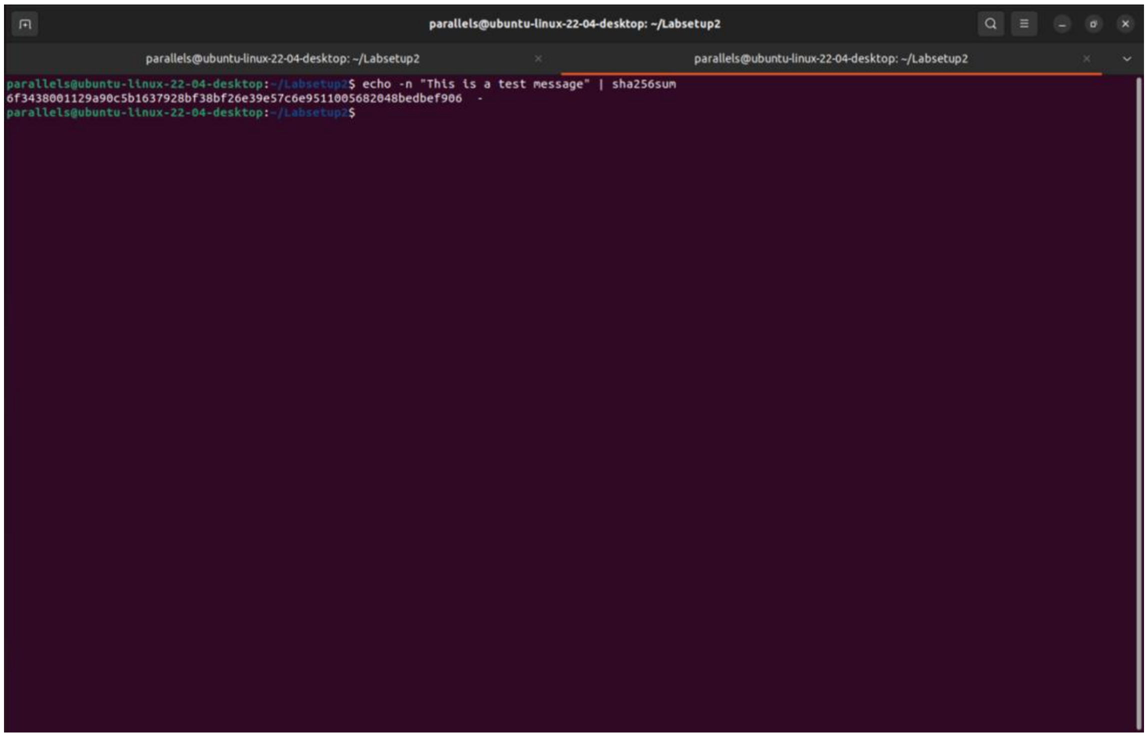
Screenshot7: (Return to text)



```
//Message Equals 46 bytes = 368 bits = \x170
123456:myname=HaydenEubanks&uid=1001&lstcmd=1

//Padding Byte 1 = 47 bytes
\x80


//64 - 47 = 17 bytes (9 bytes of \x00 before last 8)

\x00
\x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00

//Last 8 bytes
\x00 \x00 \x00 \x00 \x00 \x00 \x01 \x70


//Total Padded message
123456:myname=HaydenEubanks&uid=1001&lstcmd=1
\x80
\x00
\x00 \x00 \x00 \x00 \x00 \x00 \x00 \x00
\x00 \x00 \x00 \x00 \x00 \x00 \x01 \x70


//For URL
123456:myname=HaydenEubanks&uid=1001&lstcmd=1
%80
%00
%00 %00 %00 %00 %00 %00 %00 %00
%00 %00 %00 %00 %00 %00 %01 %70
```

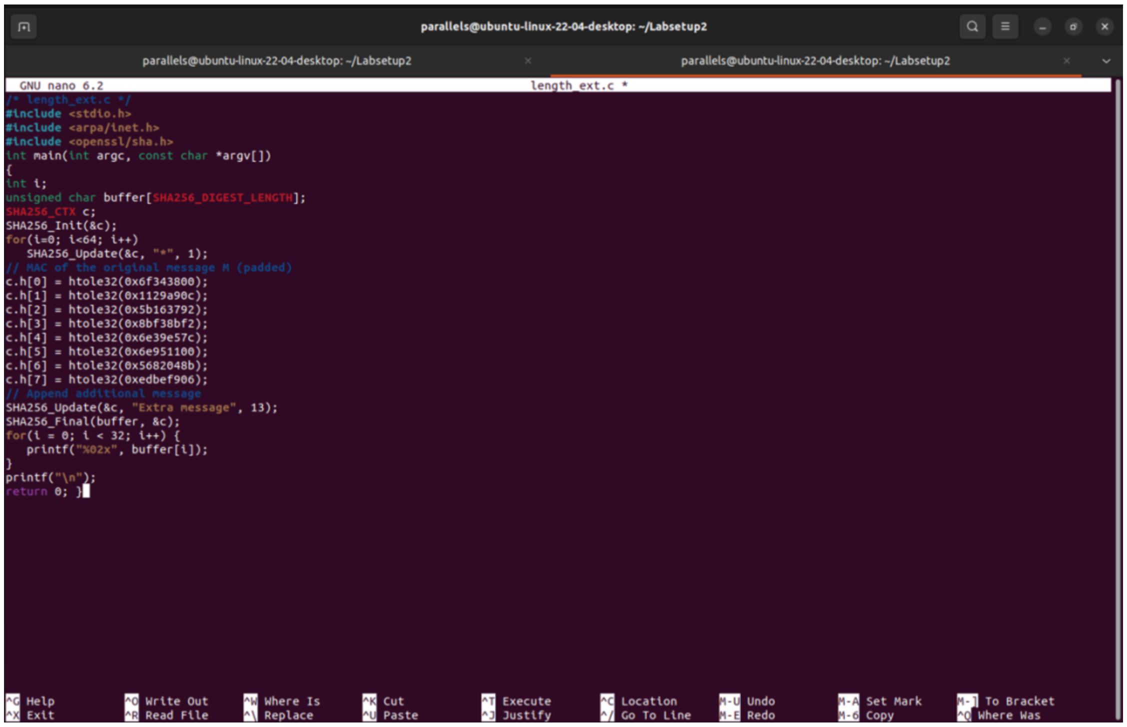
Screenshots: Task 3

Screenshot8: (Return to text)


parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2

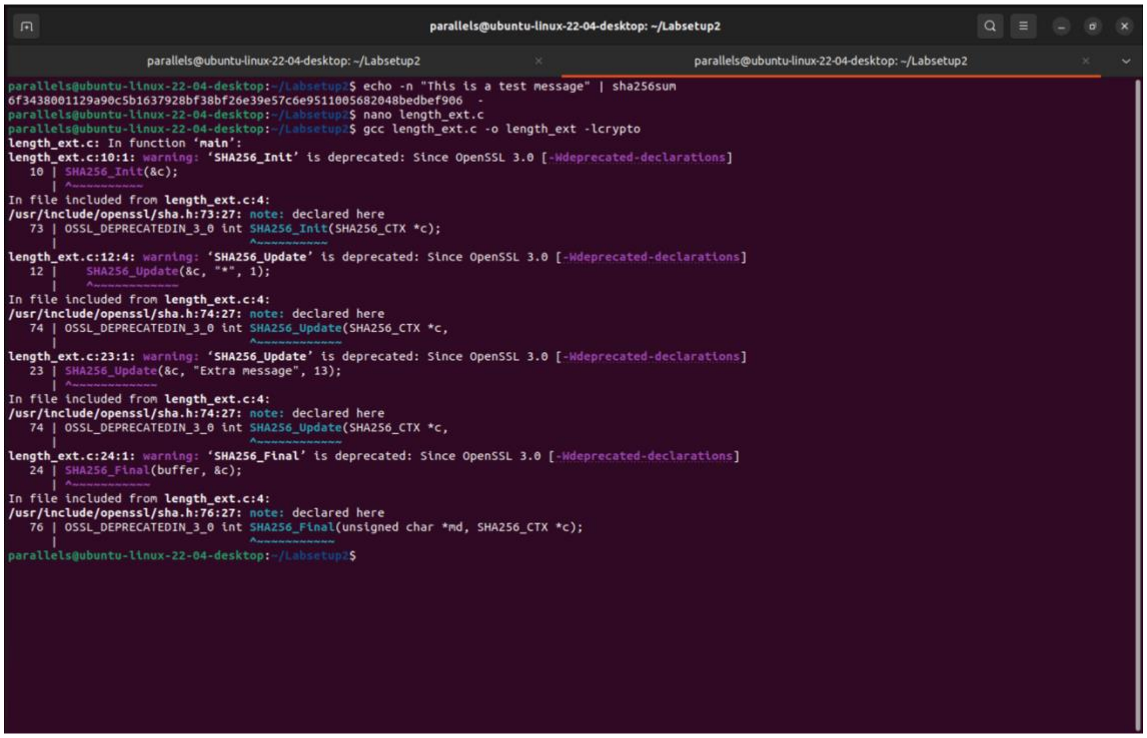
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906  -
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot9: (Return to text)



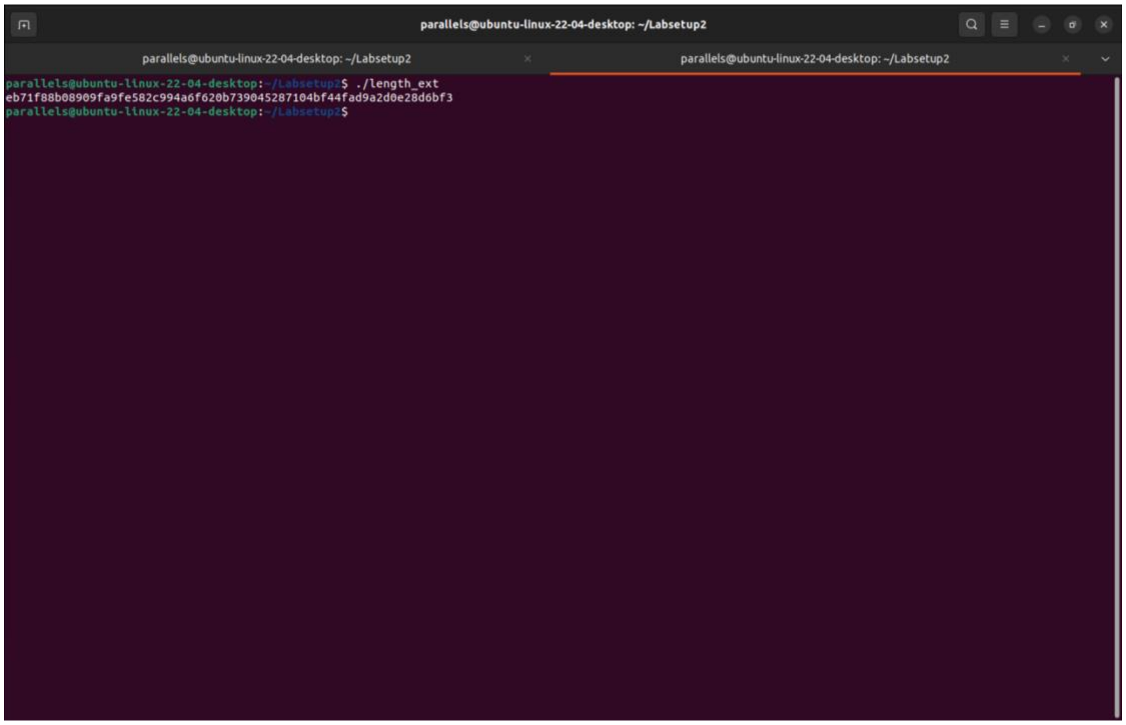
```
/* length_ext.c */
#include <stdio.h>
#include <arpa/inet.h>
#include <openssl/sha.h>
int main(int argc, const char *argv[])
{
int i;
unsigned char buffer[SHA256_DIGEST_LENGTH];
SHA256_CTX c;
SHA256_Init(&c);
for(i=0; i<64; i++)
   SHA256_Update(&c, "*", 1);
// MAC of the original message M (padded)
c.h[0] = htole32(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; }
```

Screenshot10: (Return to text)



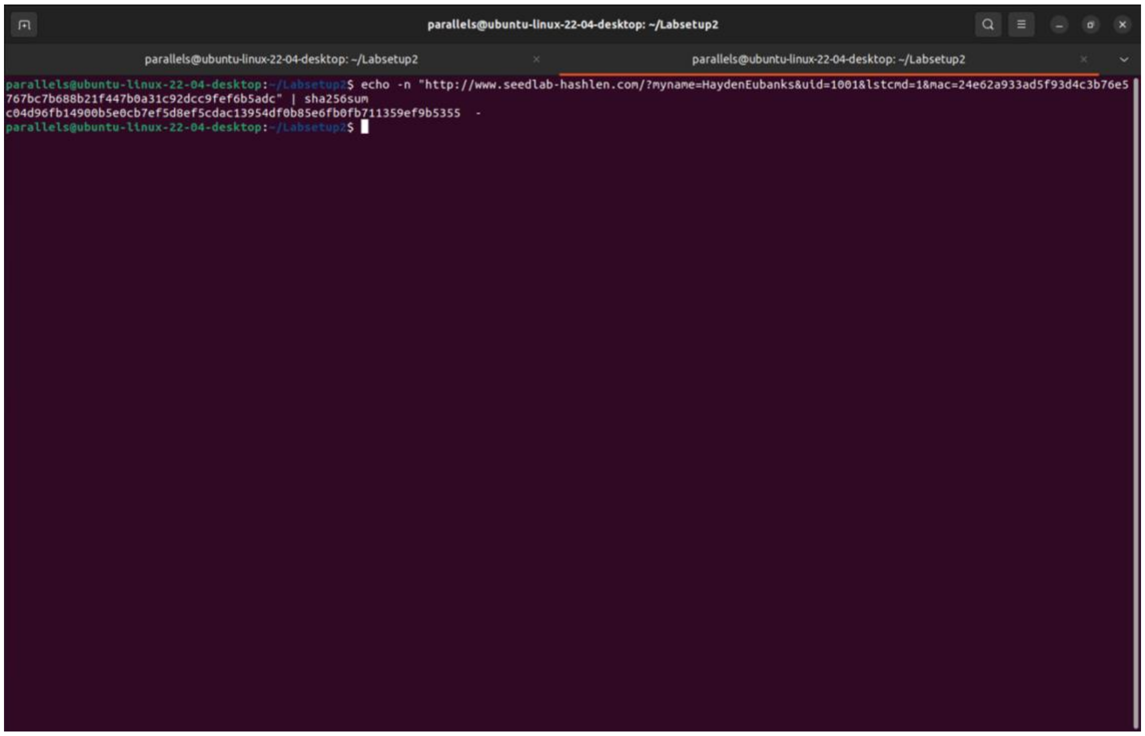
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906  -
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ nano length_ext.c
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ gcc length_ext.c -o length_ext -lcrypto
length_ext.c: In function ‘main’:
length_ext.c:10:1: warning: ‘SHA256_Init’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   10 | SHA256_Init(&c);
      | ^~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:73:27: note: declared here
   73 | OSSL_DEPRECATEDIN_3_0 int SHA256_Init(SHA256_CTX *c);
      |                           ^~~~~~~~~~~
length_ext.c:12:4: warning: ‘SHA256_Update’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   12 |    SHA256_Update(&c, "*", 1);
      |    ^~~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:74:27: note: declared here
   74 | OSSL_DEPRECATEDIN_3_0 int SHA256_Update(SHA256_CTX *c,
      |                           ^~~~~~~~~~~~~
length_ext.c:23:1: warning: ‘SHA256_Update’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   23 | SHA256_Update(&c, "Extra message", 13);
      | ^~~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:74:27: note: declared here
   74 | OSSL_DEPRECATEDIN_3_0 int SHA256_Update(SHA256_CTX *c,
      |                           ^~~~~~~~~~~~~
length_ext.c:24:1: warning: ‘SHA256_Final’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   24 | SHA256_Final(buffer, &c);
      | ^~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:76:27: note: declared here
   76 | OSSL_DEPRECATEDIN_3_0 int SHA256_Final(unsigned char *md, SHA256_CTX *c);
      |                           ^~~~~~~~~~~~
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot11: (Return to text)



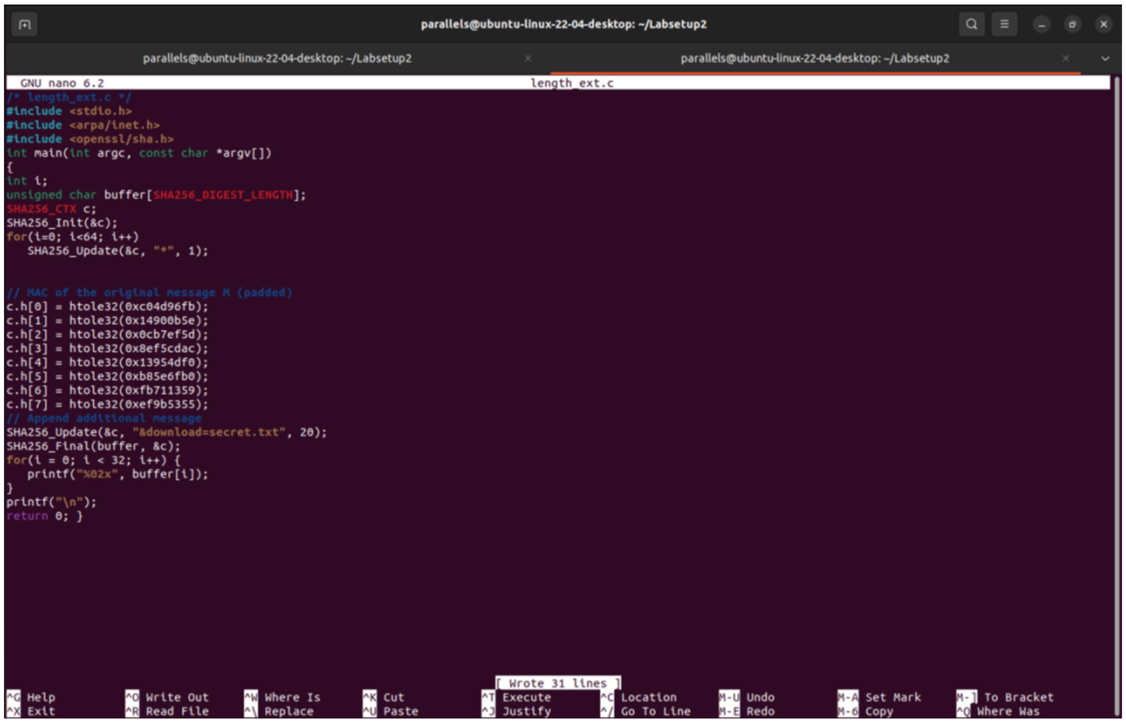
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ ./length_ext
eb71f88b08909fa9fe582c994a6f620b739045287104bf44fad9a2d0e28d6bf3
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot12: (Return to text)



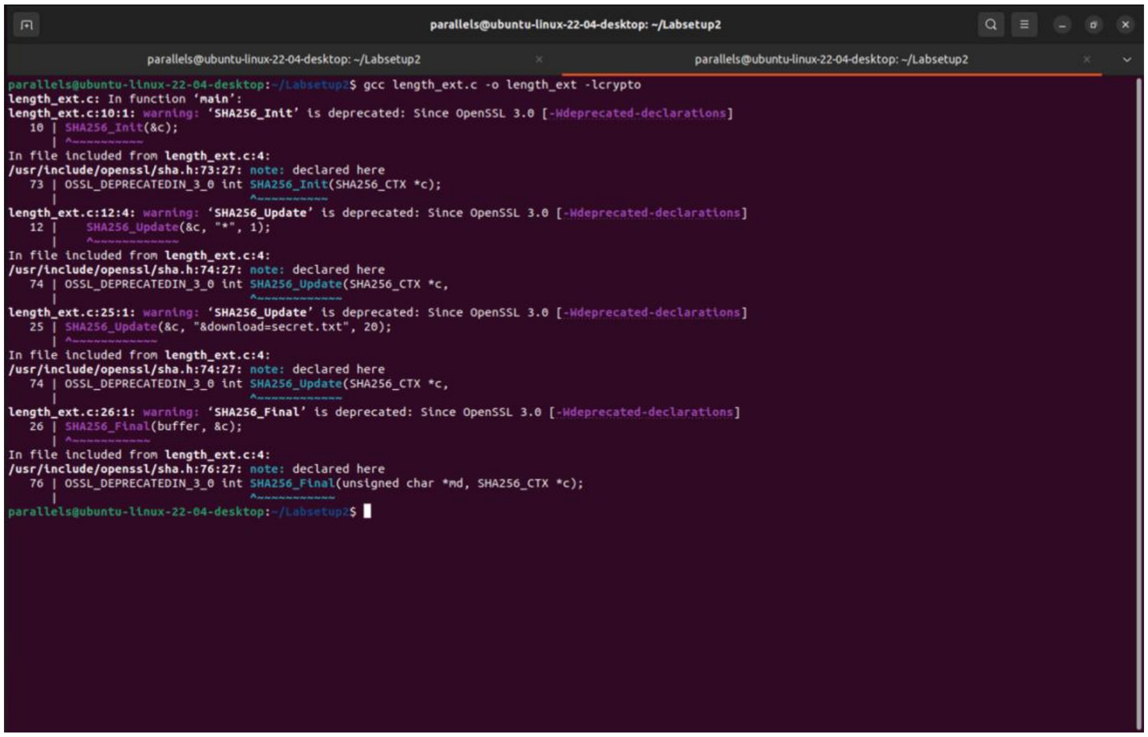
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ echo -n "http://www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1&mac=24e62a933ad5f93d4c3b76e5
767bc7b688b21f447b0a31c92dcc9fef6b5adc" | sha256sum
c04d96fb14900b5e0cb7ef5d8ef5cdac13954df0b85e6fb0fb711359ef9b5355  -
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ 
```

Screenshot13: (Return to text)



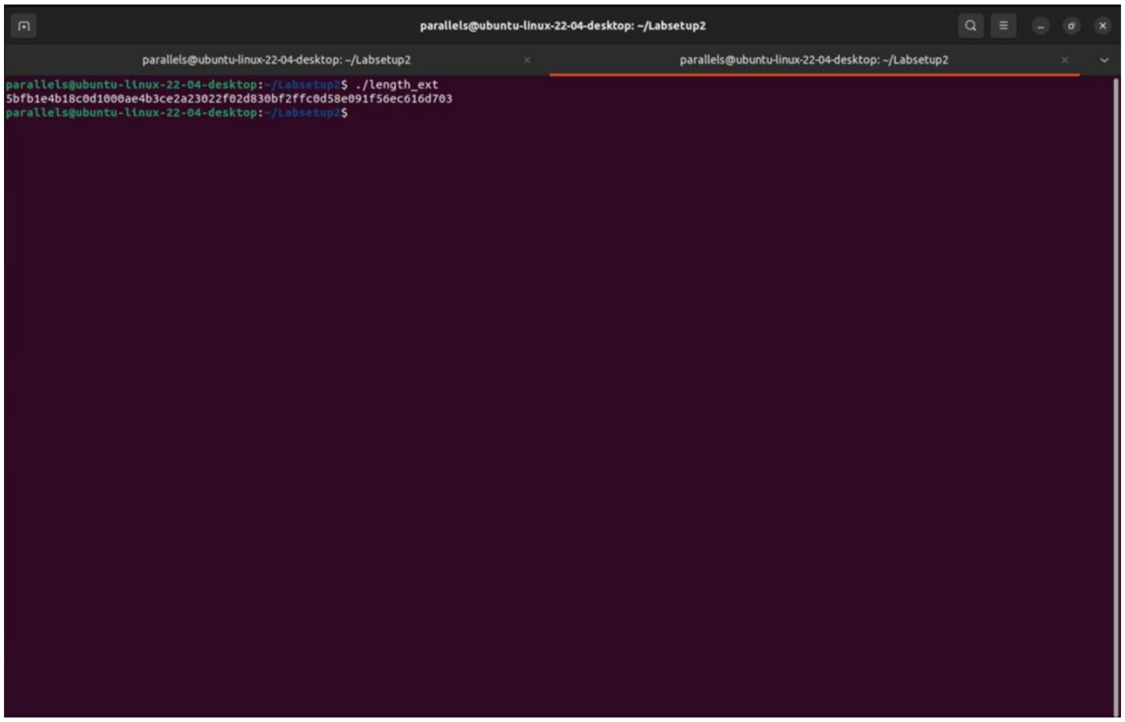
```
  GNU nano 6.2                                length_ext.c
/* length_ext.c */
#include <stdio.h>
#include <arpa/inet.h>
#include <openssl/sha.h>
int main(int argc, const char *argv[])
{
int i;
unsigned char buffer[SHA256_DIGEST_LENGTH];
SHA256_CTX c;
SHA256_Init(&c);
for(i=0; i<64; i++)
   SHA256_Update(&c, "*", 1);


// MAC of the original message M (padded)
c.h[0] = htole32(0xc04d96fb);
c.h[1] = htole32(0x14900b5e);
c.h[2] = htole32(0x0cb7ef5d);
c.h[3] = htole32(0x8ef5cdac);
c.h[4] = htole32(0x13954df0);
c.h[5] = htole32(0xb85e6fb0);
c.h[6] = htole32(0xfb711359);
c.h[7] = htole32(0xef9b5355);
// 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; }

                                         [ Wrote 31 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^Q Where Was
```

Screenshot14: (Return to text)



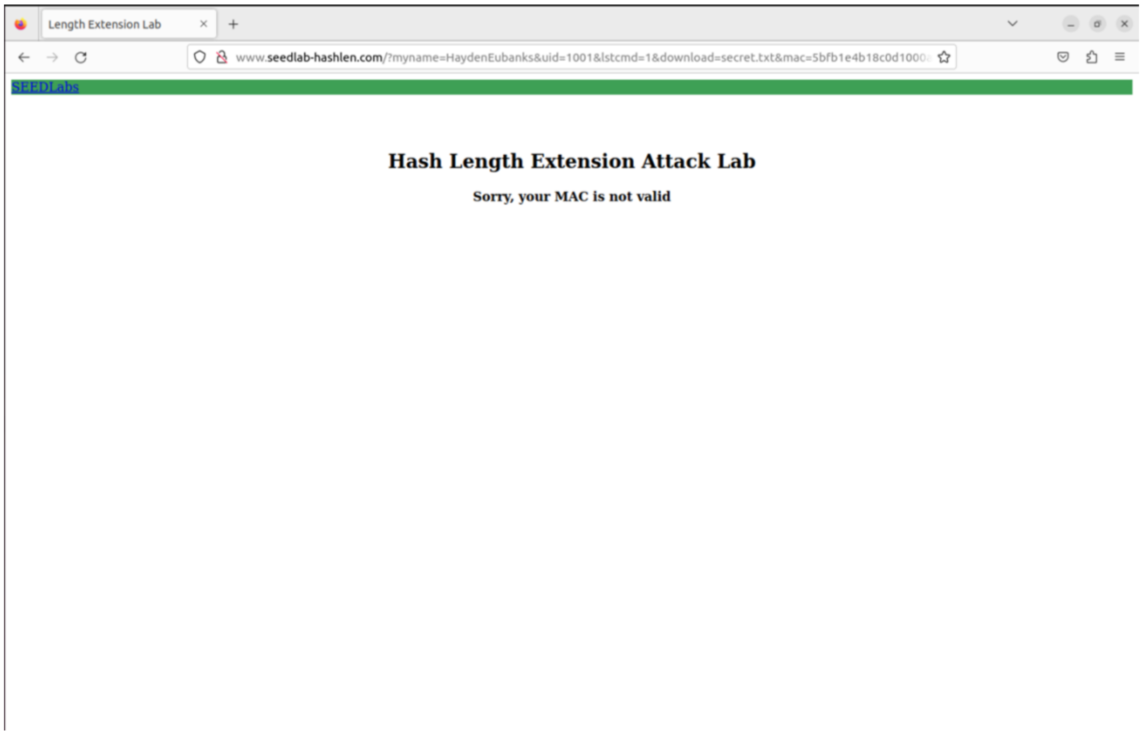
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ gcc length_ext.c -o length_ext -lcrypto
length_ext.c: In function ‘main’:
length_ext.c:10:1: warning: ‘SHA256_Init’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   10 | SHA256_Init(&c);
      | ^~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:73:27: note: declared here
   73 | OSSL_DEPRECATEDIN_3_0 int SHA256_Init(SHA256_CTX *c);
      |                           ^~~~~~~~~~~
length_ext.c:12:4: warning: ‘SHA256_Update’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   12 |    SHA256_Update(&c, "*", 1);
      |    ^~~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:74:27: note: declared here
   74 | OSSL_DEPRECATEDIN_3_0 int SHA256_Update(SHA256_CTX *c,
      |                           ^~~~~~~~~~~~~
length_ext.c:25:1: warning: ‘SHA256_Update’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   25 | SHA256_Update(&c, "&download=secret.txt", 20);
      | ^~~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:74:27: note: declared here
   74 | OSSL_DEPRECATEDIN_3_0 int SHA256_Update(SHA256_CTX *c,
      |                           ^~~~~~~~~~~~~
length_ext.c:26:1: warning: ‘SHA256_Final’ is deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
   26 | SHA256_Final(buffer, &c);
      | ^~~~~~~~~~~~
In file included from length_ext.c:4:
/usr/include/openssl/sha.h:76:27: note: declared here
   76 | OSSL_DEPRECATEDIN_3_0 int SHA256_Final(unsigned char *md, SHA256_CTX *c);
      |                           ^~~~~~~~~~~~
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ 
```

Screenshot15: (Return to text)



```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ ./length_ext
5bfb1e4b18c0d1000ae4b3ce2a23022f02d830bf2ffc0d58e091f56ec616d703
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot16: (Return to text)



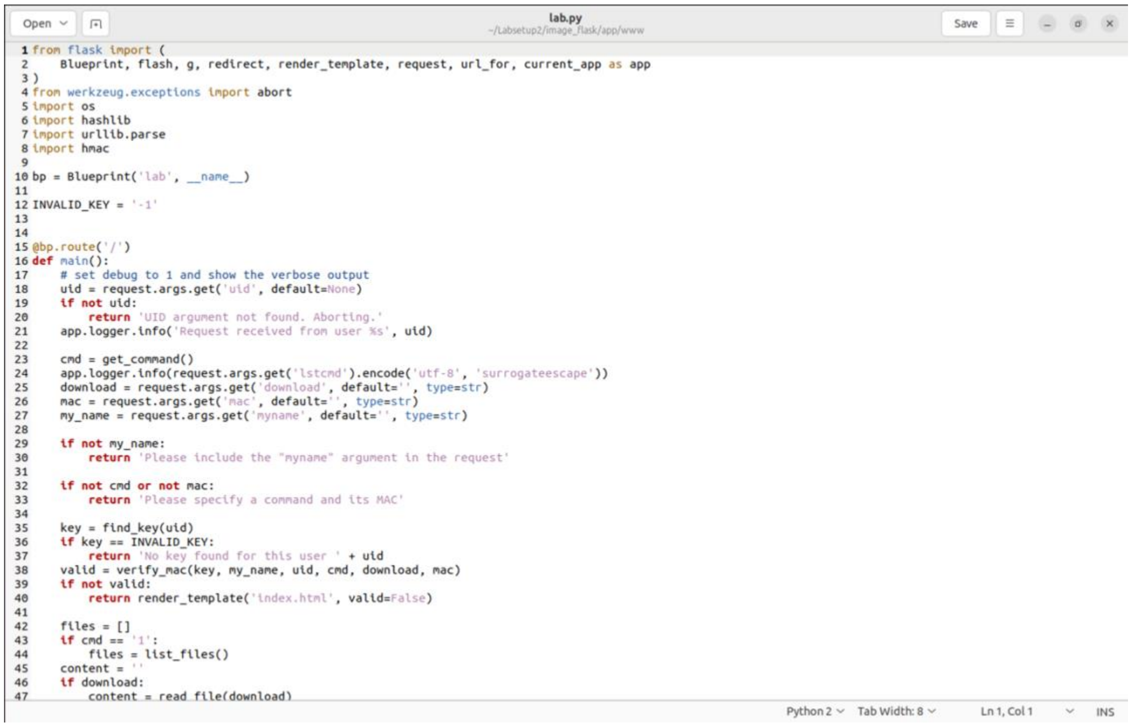
www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1&download=secret.txt&mac=5bfb1e4b18c0d1000

Hash Length Extension Attack Lab

Sorry, your MAC is not valid

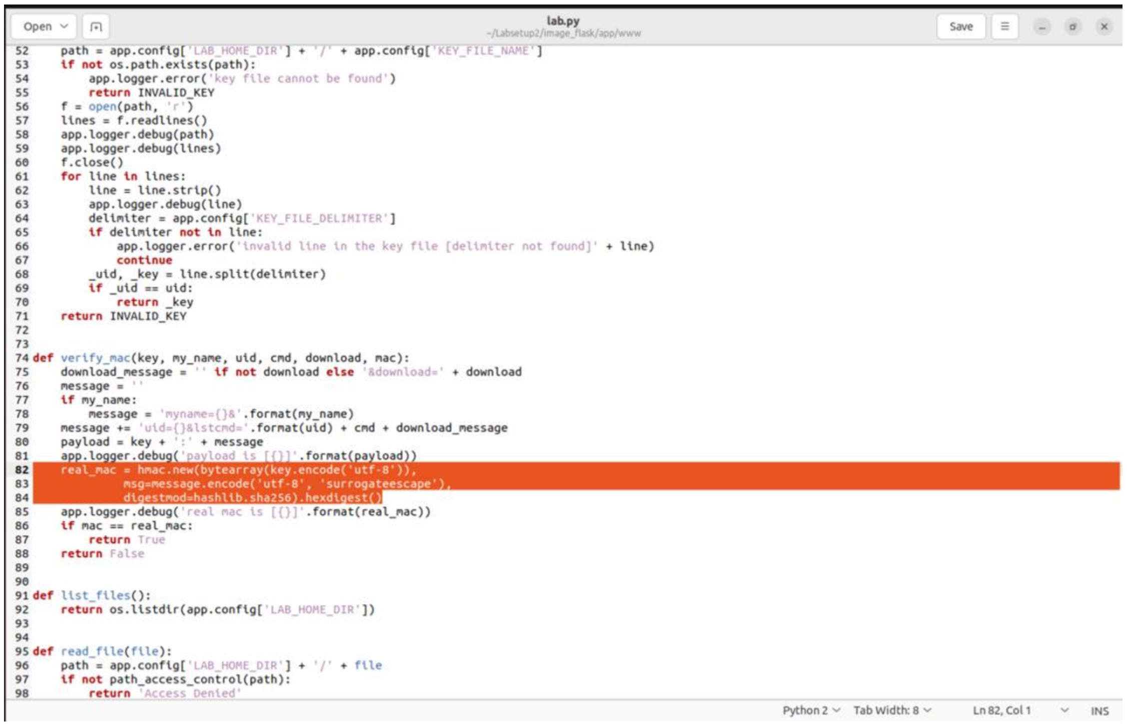
Screenshots: Task 4

Screenshot17: (Return to text)



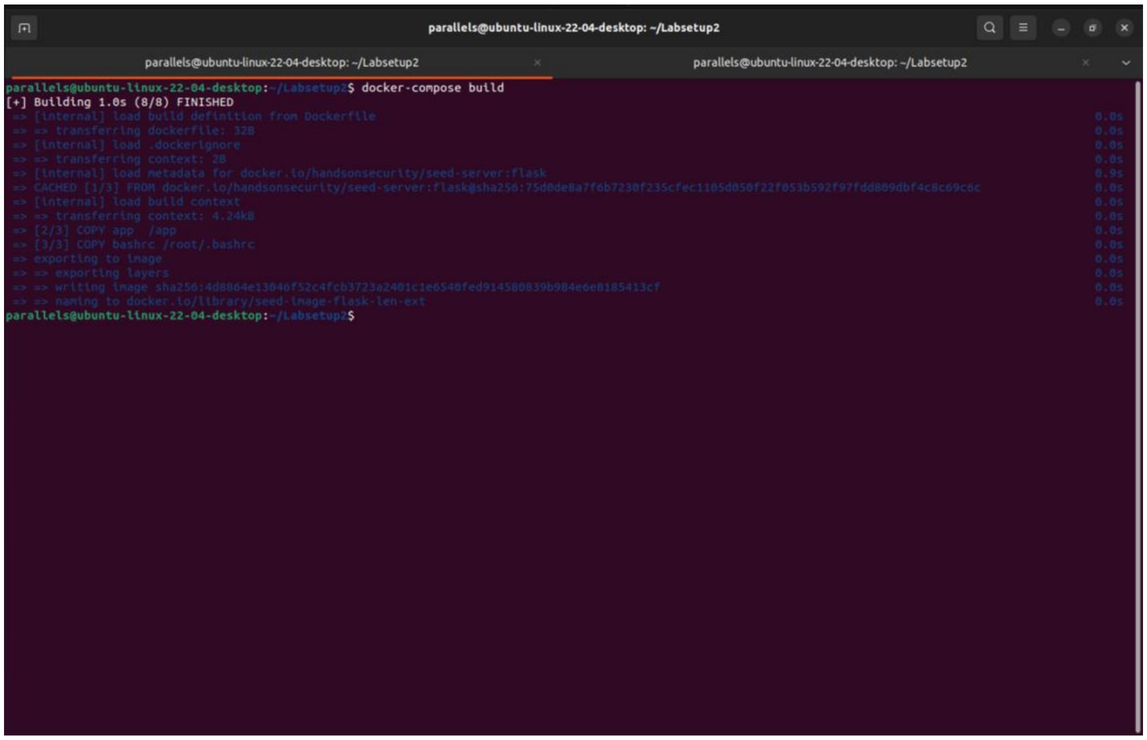
```
from flask import (
    Blueprint, flash, g, redirect, render_template, request, url_for, current_app as app
)
from werkzeug.exceptions import abort
import os
import hashlib
import urllib.parse
import hmac

bp = Blueprint('lab', __name__)

INVALID_KEY = '-1'


@bp.route('/')
def main():
    # set debug to 1 and show the verbose output
    uid = request.args.get('uid', default=None)
    if not uid:
        return 'UID argument not found. Aborting.'
    app.logger.info('Request received from user %s', uid)

    cmd = get_command()
    app.logger.info(request.args.get('lstcmd').encode('utf-8', 'surrogateescape'))
    download = request.args.get('download', default='', type=str)
    mac = request.args.get('mac', default='', type=str)
    my_name = request.args.get('myname', default='', type=str)

    if not my_name:
        return 'Please include the "myname" argument in the request'

    if not cmd or not mac:
        return 'Please specify a command and its MAC'

    key = find_key(uid)
    if key == INVALID_KEY:
        return 'No key found for this user ' + uid
    valid = verify_mac(key, my_name, uid, cmd, download, mac)
    if not valid:
        return render_template('index.html', valid=False)

    files = []
    if cmd == '1':
        files = list_files()
    content = ''
    if download:
        content = read_file(download)
```

Screenshot18: (Return to text)



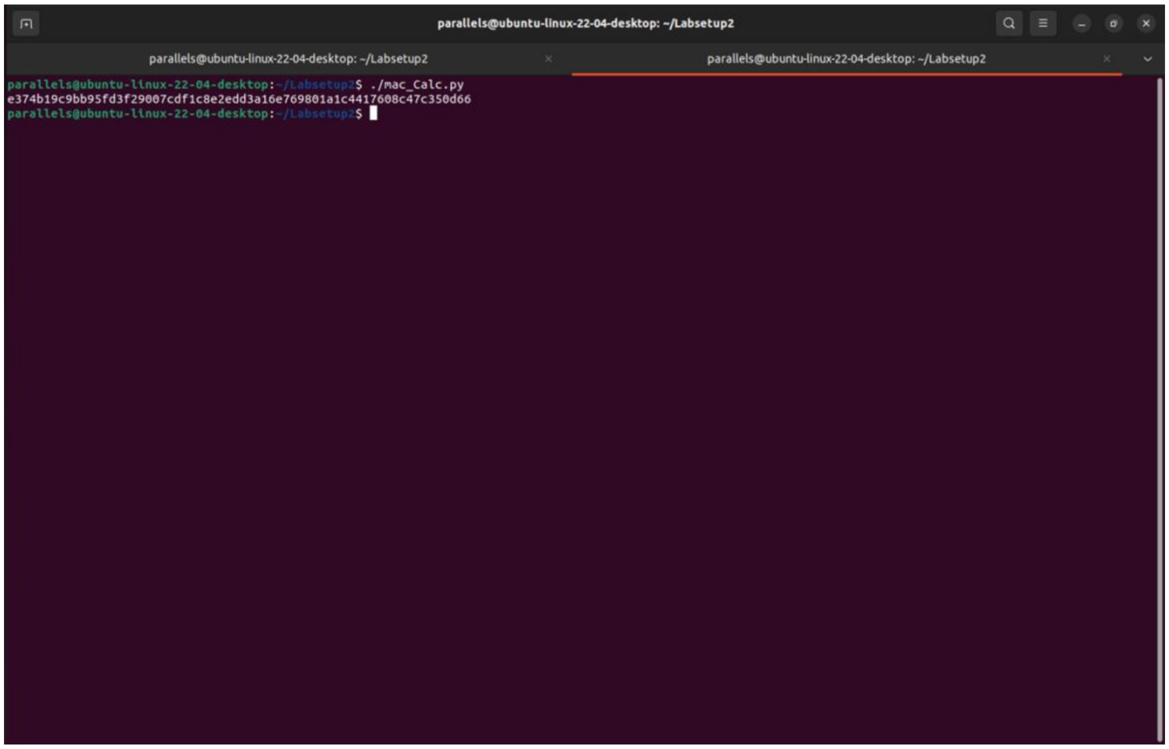
```
    path = app.config['LAB_HOME_DIR'] + '/' + app.config['KEY_FILE_NAME']
    if not os.path.exists(path):
        app.logger.error('key file cannot be found')
        return INVALID_KEY
    f = open(path, 'r')
    lines = f.readlines()
    app.logger.debug(path)
    app.logger.debug(lines)
    f.close()
    for line in lines:
        line = line.strip()
        app.logger.debug(line)
        delimiter = app.config['KEY_FILE_DELIMITER']
        if delimiter not in line:
            app.logger.error('invalid line in the key file [delimiter not found]' + line)
            continue
        _uid, _key = line.split(delimiter)
        if _uid == uid:
            return _key
    return INVALID_KEY


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()
    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):
    path = app.config['LAB_HOME_DIR'] + '/' + file
    if not path_access_control(path):
        return 'Access Denied'
```

Screenshot19: (Return to text)



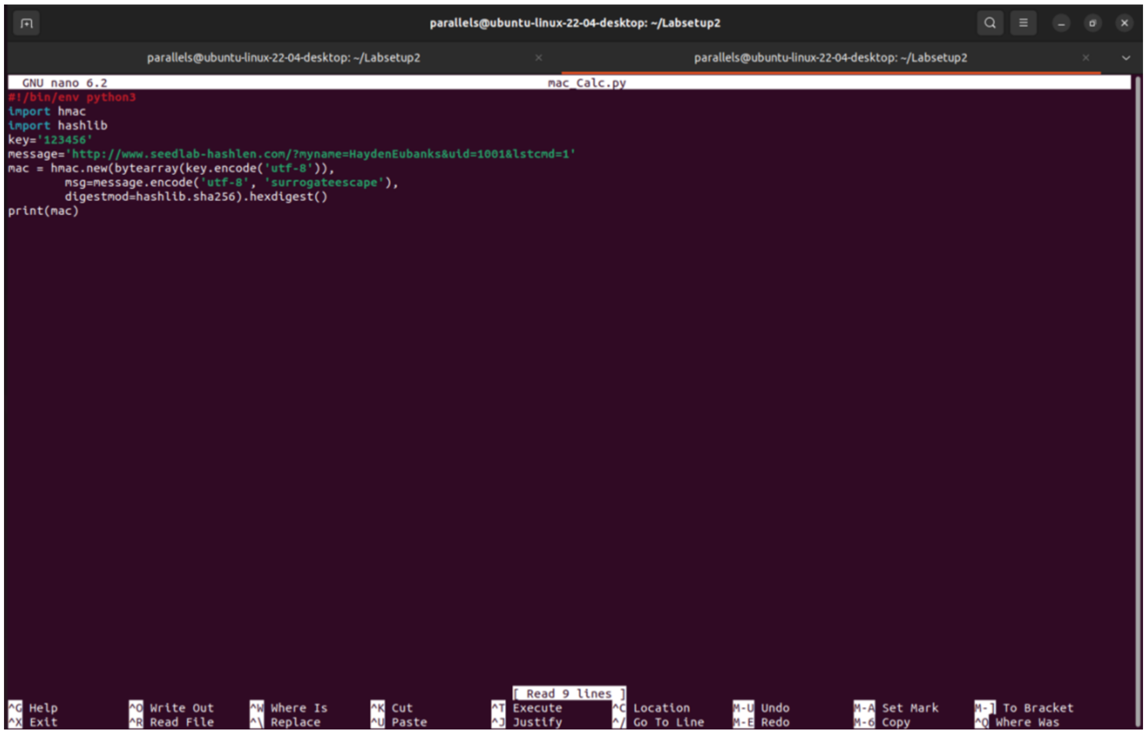
```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ docker-compose build
[+] Building 1.0s (8/8) FINISHED
 => [internal] load build definition from Dockerfile                                                                                   0.0s
 => => transferring dockerfile: 32B                                                                                                    0.0s
 => [internal] load .dockerignore                                                                                                      0.0s
 => => transferring context: 2B                                                                                                       0.0s
 => [internal] load metadata for docker.io/handsonsecurity/seed-server:flask                                                           0.9s
 => CACHED [1/3] FROM docker.io/handsonsecurity/seed-server:flask@sha256:75d0de8a7f6b7230f235cfec1105d050f22f053b592f97fdd809dbf4c8c69c6c  0.0s
 => [internal] load build context                                                                                                      0.0s
 => => transferring context: 4.24kB                                                                                                    0.0s
 => [2/3] COPY app  /app                                                                                                               0.0s
 => [3/3] COPY bashrc /root/.bashrc                                                                                                    0.0s
 => exporting to image                                                                                                                 0.0s
 => => exporting layers                                                                                                                0.0s
 => => writing image sha256:4d8864e13046f52c4fcb3723a2401c1e6540fed914580839b984e6e8185413cf                                           0.0s
 => => naming to docker.io/library/seed-image-flask-len-ext                                                                            0.0s
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot20: (Return to text)



```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ ./mac_Calc.py
e374b19c9bb95fd3f29007cdf1c8e2edd3a16e769801a1c4417608c47c350d66
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

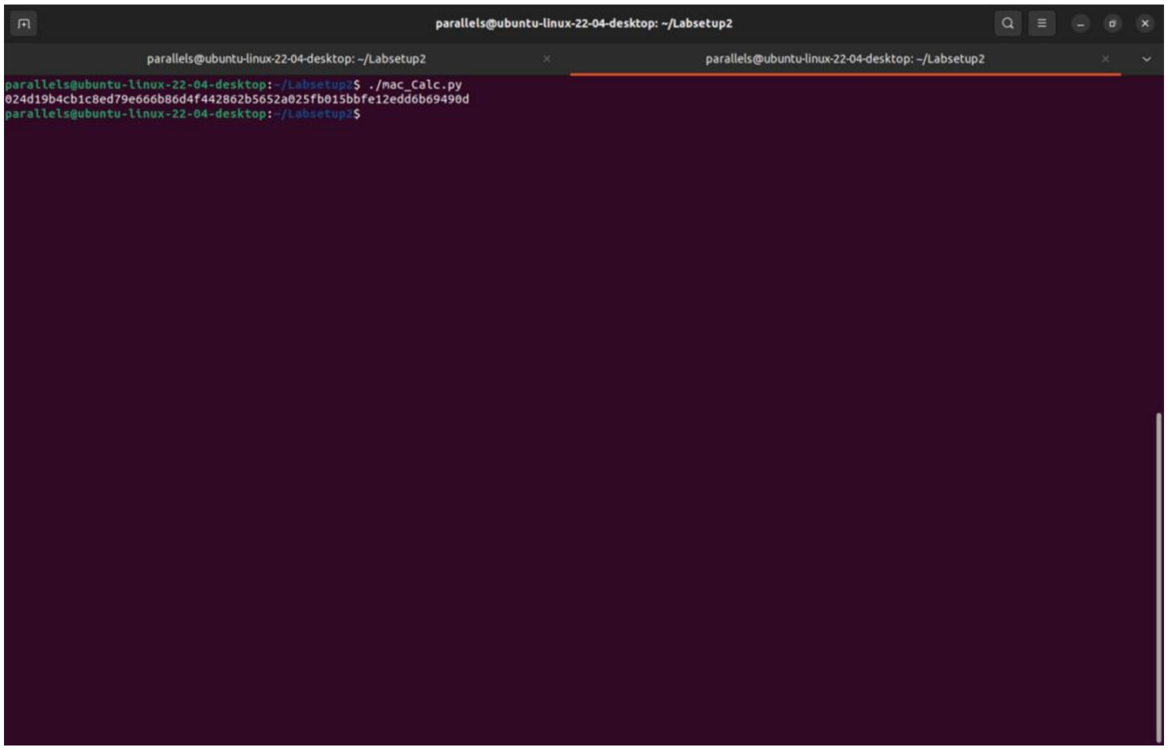
Screenshot21: (Return to text)



parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2

```
  GNU nano 6.2                                    mac_Calc.py
#!/bin/env python3
import hmac
import hashlib
key='123456'
message='http://www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),
        msg=message.encode('utf-8', 'surrogateescape'),
        digestmod=hashlib.sha256).hexdigest()
print(mac)

                                    [ Read 9 lines ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo      M-A Set Mark  M-] To Bracket
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo      M-6 Copy      ^Q Where Was
```

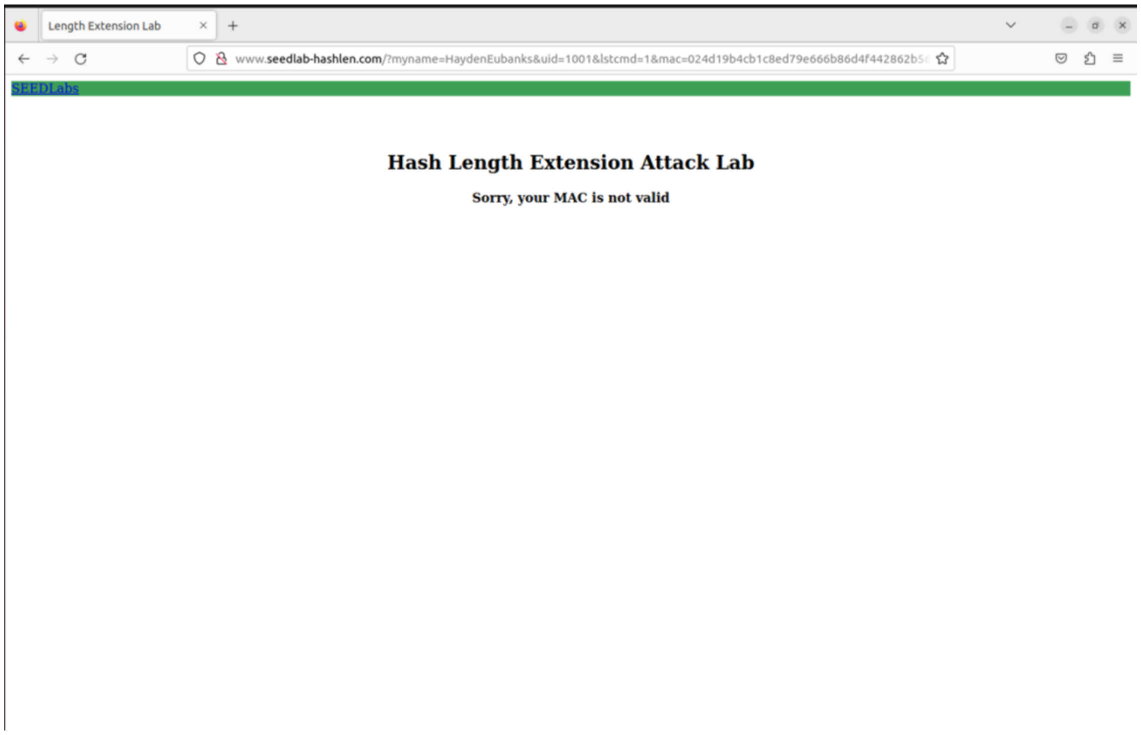
Screenshot22: (Return to text)



parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2
parallels@ubuntu-linux-22-04-desktop: ~/Labsetup2

```
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$ ./mac_Calc.py
024d19b4cb1c8ed79e666b86d4f442862b5652a025fb015bbfe12edd6b69490d
parallels@ubuntu-linux-22-04-desktop:~/Labsetup2$
```

Screenshot23: (Return to text)


Length Extension Lab
www.seedlab-hashlen.com/?myname=HaydenEubanks&uid=1001&lstcmd=1&mac=024d19b4cb1c8ed79e666b86d4f442862b5
SEEDLabs

Hash Length Extension Attack Lab

Sorry, your MAC is not valid