Hash Length Extension Attack Lab

Telle Miller

Liberty University – CSIS 463

10/13/2021

**Introduction**

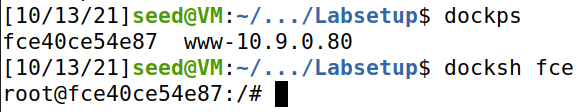
The goal of this lab was to familiarize myself with message authentication codes (MAC) and their purpose in the client-server communication schema. When there is a message transmission between the client and the server, the packet being sent could be intercepted and then forwarded on to the server once it has been manipulated. This type of malicious activity is known as a “man in the middle” (MITM) attack. To avoid this type of attack, the server must verify the integrity of the request received using a MAC. This lab showed how to work with MACs and demonstrated how they can be insecure when targeted by a length extension attack.

**Setup**

As with the previous labs, I completed this lab using the resources provided by SEED labs. This included their pre-built Ubuntu 20.04 VM and the setup files outlined in the lab PDF. This lab also required setting up a webserver to send communications to, which is a first for the labs so far. This webserver was constructed inside of a container and built using Docker. The lab instructions made this setup simple.

Text

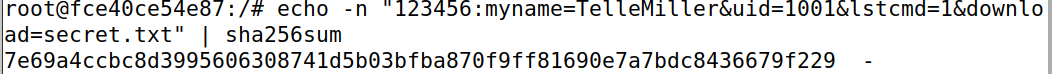
Description automatically generated



**Tasks**

Task 1: Send Request to List Files

This first task was a good way to acquainted with the server setup and how to send commands to it using a MAC. Using the UID provided by the lab files and the *echo* command, I leveraged the sha256sum hash to generate a MAC. From this point, all that was left was to enter the URL into a browser and see what the server returned. The images below demonstrate how to generate the MAC and what the server returned when the URL was accessed.



Graphical user interface, text, application

Description automatically generated

Task 2: Create Padding

This task was to demonstrate how the SHA-256 hash algorithm block size affected the message. The hash algorithm has a block size of 64 bytes, so all messages need to be a multiple of this number. Any messages that are not a multiple will have padding added to the end. The message to pad is shown below:



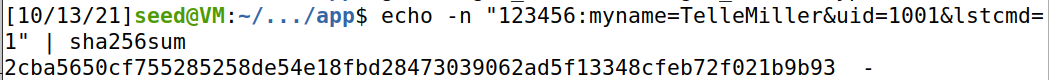
This message is 43 bytes long, meaning that the SHA-256 algorithm requires 21 bytes of padding. The original message is 344 bits long (43 bytes \* 8), which is **\x01 \x58** in hexadecimal. All this information is used to create the padding needed for the original message. The format of the padding is shown below:

Text, letter

Description automatically generated

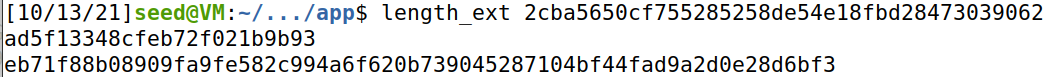
Task 3: The Length Extension Attack

The goal of this task is to perform a length extension attack by generating a new MAC for a previously valid message. Recreating this MAC is done using the *length\_ext.c* file provided by SEED labs. The first step was to create a genuine message with a genuine MAC.



Now that I was able to construct the original message and MAC, I compiled the *length\_ext.c* file and used it to add the malicious command and generate a new MAC. This is achieved by passing the genuine MAC to the program as a command line argument.





I then used this altered MAC to construct a URL that showed the contents of the *secret.txt* file, despite that command not being in the originally hashed message.

Graphical user interface, text, application

Description automatically generated

Task 4: Attack Mitigation using HMAC

The final task in this lab was to demonstrate how to protect against the kinds of attacks used in task 3. The initial step was to change how the MAC was verified in the servers *lab.py* file. The code on line 75 through 77 in the image below was added to use the *hmac* library to verify the integrity of the MAC.

Text

Description automatically generated

After restarting the server container, I was still able to access the *lstfile* command with a valid MAC verified by the *hmac* change in the lab.py file.

Text, letter

Description automatically generated

The benefit of using HMAC over MAC is that it is not prone to length extension attacks. The difference here is that HMAC computes a hash twice, using two keys generated when the message is hashed. On the first hash, the algorithm uses the message and the first key. On the second hash, the final output is created using the inner hash result and the second key. This means that there is no way to propagate a length extension attack against HMAC.

**References**

Delfs, Hans, and Helmut Knebl. *Introduction to Cryptography: Principles and Applications.* Springer, 2007.

Lawson, N. (2009, October 29). *Stop using unsafe keyed hashes, use HMAC.* rdist. Retrieved October 13, 2021, from <https://rdist.root.org/2009/10/29/stop-using-unsafe-keyed-hashes-use-hmac/>.

*"In Our Time: Cryptography".* BBC Radio 4. Retrieved 11 October 2021.

Boström, M. (2015, October 29). *size\_t Does Matter. London*; DevOps Exchange 2015. Retrieved October 13, 2021 from <https://bostik.iki.fi/dc4420/size_t-does-matter--hash-length-extensions.pdf>.

Du, W. (2019). *Computer & internet security: A hands-on approach* (2nd ed.). Wenliang Du.