Hash Length Extension Attack Lab

1 Introduction

When a client and a server communicate over the internet, they are subject to MITM attacks. An attacker can intercept the request from the client. The attacker may choose to modify the data and send the modified request to the server. In such a scenario, the server needs to verify the integrity of the request received. The standard way to verify the integrity of the request is to attach a tag called MAC to the request. There are many ways to calculate MAC, and some of the methods are not secure.

MAC is calculated from a secret key and a message. A naive way to calculate MAC is to concatenate the key with the message and calculate the one way hash of the resulting string. This method seems to be fine, but it is subject to an attack called length extension attack, which allows attackers to modify the message while still being able to generate a valid MAC based on the modified message, without knowing the secret key.

The objective of this lab is to help students understand how the length extension attack works. Students will launch the attack against a server program; they will forge a valid command and get the server to execute the command.

Readings. Detailed coverage of the one way hash function can be found in the following:

- Chapter 22 of the SEED Book, *Computer & Internet Security: A Hands-on Approach*, 2nd Edition, by Wenliang Du. See details at `https://www.handsonsecurity.net`.

Lab environment. This lab has been tested on the SEED Ubuntu 20.04 VM. You can download a pre-built image from the SEED website, and run the SEED VM on your own computer. However, most of the SEED labs can be conducted on the cloud, and you can follow our instruction to create a SEED VM on the cloud.

2 Lab Environment

We have set up a web server for this lab. A client can send a list of commands to this server. Each request must attach a MAC computed based on a secret key and the list of commands. The server will only execute the commands in the request if the MAC is verified successfully. We will use the host VM as the client and use a container for the web server.

Container Setup and Commands. Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build  # Build the container images
$ docker-compose up     # Start the containers
$ docker-compose down   # Shut down the containers

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

About the web server. We use the domain `www.seedlab-hashlen.com` to host the server program. In our VM, we map this hostname to the web server container (`10.9.0.80`). This can be achieved by adding the following entry to the `/etc/hosts` file (please add it if the entry is not in the VM).

```
10.9.0.80  www.seedlab-hashlen.com
```

The server code is in the `Labsetup/image_flask/app` folder. It has two directories. The `www` directory contains the server code, and the `LabHome` directory contains a secret file and the key used for computing the MAC.

Sending requests. The server program accepts the following commands:

- The `lstcmd` command: the server will list all the files in the `LabHome` folder.
- The `download` command: the server will return the contents of the specified file from the `LabHome` directory.

A typical request sent by the client to the server is shown below. The server requires a `uid` argument to be passed. It uses `uid` to get the MAC key from `LabHome/key.txt`. The command in the example below is `lstcmd`, and its value is set to `1`. It requests the server to list all the files. The last argument is the MAC computed based on the secret key (shared by the client and the server) and the command arguments. Before executing the command, the server will verify the MAC to ensure the command's integrity.

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

Students should replace the value `JohnDoe` in the `myname` field with their actual names (no space is allowed). This parameter is to make sure that different students' results are different, so students cannot copy from one another. The server does not use this argument, but it checks whether the argument is present or not. Requests will be rejected if this field is not included. Instructors can use this argument to check whether students have done the work by themselves. No point will be given if students do not use their real names in this task.

The following shows another example. The request includes two commands: list all the files and download the file `secret.txt`. Similarly, a valid MAC needs to be attached, or the server will not execute these commands.

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&download=secret.txt
&mac=dc8788905dbcbceffcdd5578887717c12691b3cf1dac6b2f2bcfabc14a6a7f11
```

3 Tasks

3.1 Task 1: Send Request to List Files

In this task, we will send a benign request to the server so we can see how the server responds to the request. The request we want to send is as follows:

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<need-to-fill>
&lstcmd=1&mac=<need-to-calculate>
```

To send such a request, other than using our real names, we need to fill in the two missing arguments. Students need to pick a uid number from the `key.txt` in the `LabHome` directory. This file contains a list of colon-separated uid and key values. Students can use any uid and its associated key value. For example, students can use uid `1001` and its key `123456`.

The second missing argument is the MAC, which can be calculated by concatenating the key with the contents of the requests `R` (the argument part only), with a colon added in between. See the following example:

```
Key:R = 123456:myname=JohnDoe&uid=1001&lstcmd=1
```

The MAC will be calculated using the following command:

```
$ echo -n "123456:myname=JohnDoe&uid=1001&lstcmd=1" | sha256sum
7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3  -
```

We can then construct the complete request and send it to the server program using the browser:

```
http://www.seedlab-hashlen.com/?myname=JohnDoe&uid=1001&lstcmd=1
&mac=7d5f750f8b3203bd963d75217c980d139df5d0e50d19d6dfdb8a7de1f8520ce3
```

Task. Please send a download command to the server, and show that you can get the results back.

3.2 Task 2: Create Padding

To conduct the hash length extension attack, we need to understand how padding is calculated for one-way hash. The block size of SHA-256 is 64 bytes, so a message M will be padded to the multiple of 64 bytes during the hash calculation. According to RFC 6234, paddings for SHA256 consist of one byte of \x80, followed by a many 0's, followed by a 64-bit (8 bytes) length field (the length is the number of **bits** in the M).

Assume that the original message is `M = "This is a test message"`. The length of M is 22 bytes, so the padding is `64 - 22 = 42` bytes, including 8 bytes of the length field. The length of M in term of bits is `22 * 8 = 176 = 0xB0`. SHA256 will be performed in the following padded message:

```
"This is a test message"
"\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\x00\x00\x00"
"\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\xB0"
```

It should be noted that the length field uses the Big-Endian byte order, i.e., if the length of the message is `0x012345`, the length field in the padding should be:

```
"\x00\x00\x00\x00\x00\x01\x23\x45"
```

Task. Students need to construct the padding for the following message (the actual value of the `<key>` and `<uid>` should be obtained from the `LabHome/key.txt` file.

```
<key>:myname=<name>&uid=<uid>&lstcmd=1
```

It should be noted that in the URL, all the hexadecimal numbers in the padding need to be encoded by changing `\x` to `%`. For example, `\x80` in the padding should be replaced with `%80` in the URL above. On the server side, encoded data in the URL will be changed back to the binary numbers. See the following example:

```
"\x80\x00\x00\x99" should be encoded as "%80%00%00%99"
```

3.3 Task 3: The Length Extension Attack

In this task, we will generate a valid MAC for a URL without knowing the MAC key. Assume that we know the MAC of a valid request R, and we also know the size of the MAC key. Our job is to forge a new request based on R, while still being able to compute the valid MAC.

Given the original message `M="This is a test message"` and its MAC value, we will show how to add a message `"Extra message"` to the end of the padded M, and then compute its MAC, without knowing the secret MAC key.

```
$ echo -n "This is a test message" | sha256sum
6f3438001129a90c5b1637928bf38bf26e39e57c6e9511005682048bedbef906
```

The program below can be used to compute the MAC for the new message:

```
/* 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;
}
```

Students can compile the program as follows:

```
$ gcc length_ext.c -o length_ext -lcrypto
```

Task. Students should first generate a valid MAC for the following request (where `<uid>` and the MAC key should be obtained from the `LabHome/key.txt` file):

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1&mac=<mac>
```

Based on the `<mac>` value calculated above, please construct a new request that includes the `download` command. You are not allowed to use the secret key this time. The URL looks like below.

```
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>
&lstcmd=1<padding>&download=secret.txt&mac=<new-mac>
```

Please send the constructed request to the server, and show that you can successfully get the content of the `secret.txt` file.

3.4 Task 4: Attack Mitigation using HMAC

In the tasks so far, we have observed the damage caused when a developer computes a MAC in an insecure way by concatenating the key and the message. In this task, we will fix the mistake made by the developer. The standard way to calculate MACs is to use HMAC. Students should modify the server program's `verify_mac()` function and use Python's `hmac` module to calculate the MAC. The function resides in `lab.py`. Given a key and message (both of type string), the HMAC can be computed as shown below (if you copy and paste the code from this PDF file, the ′ characters might not be copied correctly on some platforms).

```
real_mac = hmac.new(bytearray(key.encode('utf-8')),
             msg=message.encode('utf-8', 'surrogateescape'),
             digestmod=hashlib.sha256).hexdigest()
```

After making the changes, stop all the containers, rebuild them, and start all the containers again. The change will then take effect. Students should repeat Task 1 to send a request to list files while using HMAC for the MAC calculation. Assuming that the chosen key is 123456, the HMAC can be computed in the following program.

```
#!/bin/env python3

import hmac
import hashlib

key='123456'
message='lstcmd=1'
mac = hmac.new(bytearray(key.encode('utf-8')),
               msg=message.encode('utf-8', 'surrogateescape'),
               digestmod=hashlib.sha256).hexdigest()
print(mac)
```

Students should describe why a malicious request using length extension and extra commands will fail MAC verification when the client and server use HMAC.

4 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.