Applied Cryptography Lab-09 Manual

14 November 2022 13:05

Prerequisites

Labsetup files - https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_Hash_Length_Ext/

Step 1: Unzip Labsetup.zip file to your working directory. open a terminal in the folder Labsetup and run the following **commands:**

```
$ docker-compose build
$ docker-compose up
```

Step 2: Open /etc/hosts as root in a text editor of your choice and append the following line:

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

Task 1: Send Request to List Files

In this task, we 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_or_srn>&uid=<uid>&lstcmd=1&mac=<need-to-calculate>

Step 1: Finding the uid

Go to `Labsetup/image_flask/app/LabHome` and open `key.txt`
This file contains multiple uid:key pairs. Choose one and use that uid in the request.

Step 2: Calculating mac

Command:

```
$ echo -n "<key>:myname=<name_or_srn>&uid=<uid>&lstcmd=1" | sha256sum
```

Step 3: Sending the request

Fill in the uid and mac in the request and paste it in your browser. Note that cURL or wget will not work.

Your output should look something like this:

Repeat steps 1, 2 and 3 for the request http://www.seedlab-hashlen.com/?

myname=<name_or_srn>&uid=<uid>&lstcmd=1&&download=secret.txt&mac=<need-to-calculate>

Share your observations with screenshots

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 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 terms of bits is 22*8 = 176 = 0xB0.

We generate padding using the following script:

```
payload = bytearray("<key>:myname=<name>&uid=<uid>&lstcmd=1", "utf8")
length_field = (len(payload) * 8).to_bytes(8, "big")
padding = b"\x80" + b"\x00" * (64 - len(payload) - 1 - 8) + length_field
print("".join("\\x{:02x}".format(x) for x in padding))
# for url-encoding
print("".join("%{:02x}".format(x) for x in padding))
```

Note down the paddings generated
Give screenshots of your output with observation

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.

Step 1: Substitute in the required values, then compile and run the following code

```
#include <stdio.h>
#include <openssl/sha.h>
int main(int argc, const char *argv[])
{
    SHA256_CTX c;
    unsigned char buffer[SHA256_DIGEST_LENGTH];
    int i;
    SHA256_Init(&c);
    SHA256_Update(&c,
                  "123456:myname=<name>&uid=<uid>&lstcmd=1<padding_generated_in_task_
2>"
```

```
                  "&download=secret.txt",
                  64 + 20);
    SHA256_Final(buffer, &c);
    for (i = 0; i < 32; i++)
    {
        printf("%02x", buffer[i]);
    }
    printf("\n");
    return 0;
}
```

Commands

```
gcc calculate_mac.c -o calculate_mac -lcrypto
./calculate_mac
```

Note down the hash generated

Step 2: Visit the url

Format the attack URL as follows:
http://www.seedlab-hashlen.com/?myname=<name>&uid=<uid>&lstcmd=<url_encoded_padding>&download=secret.txt&mac=<mac_generated_in_step_1>

Step 3: Perform Hash Length Extension Attack without the knowledge of the key

1. Choose an alternate uid:key pair (not the one you've been using so far)
2. Generate a legitimate request to list files using this uid:key pair (task 1). Note the URL down.
   We now know the uid, command and the hash. We do not know the key, which is required to generate new mac in case of a new command. The attack involves attempting to run a different command (viewing the contents of secret.txt) without knowing the key

   Step 1: Run the following code after changing the parameters marked in <>

   Code

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <openssl/sha.h>
#include <string.h>
int main(int argc, const char *argv[])
{
    int i;
    unsigned char buffer[SHA256_DIGEST_LENGTH];
    SHA256_CTX c;
    char hex[] = "< mac_in_new_request >";
    char subbuffer[9];
    SHA256_Init(&c);
    for (i = 0; i < 64; i++)
        SHA256_Update(&c, "*", 1);
    // MAC of the original message M (padded)
    for (i = 0; i < 8; i++)
    {
        strncpy(subbuffer, hex + i * 8, 8);
        subbuffer[8] = '\0';
```

```
            c.h[i] = htole32(strtol(subbuffer, NULL, 16));
        }
        // 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;
    }
```

Note down the new mac.

Step 2: Use the following script to generate a new padding

```
payload = bytearray("******:myname=<name>&uid=<new_uid>&lstcmd=1",'utf8')
length_field = (len(payload)*8).to_bytes(8,'big')
padding = b'\x80' + b'\x00'*(64-len(payload)-1-8) + length_field
print(''.join('%{:02x}'.format(x) for x in padding))
```

Step 3: Create a new request using the padding and hash generated in the previous steps

http://www.seedlab-hashlen.com/?myname=<name>&uid=<new_uid>
lstcmd=<padding_generated>&download=secret.txt&mac=<new_mac>

Step 4: Visit the above generated URL and provide a screenshot of your observations

Task 4: Mitigation Using HMAC

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

Script

```
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)
```