Applied Cryptography Lab-08 Manual

31 October 2022 08:57

Prerequisites

Labsetup files - https://seedsecuritylabs.org/Labs_20.04/Crypto/Crypto_MD5_Collision/

Task 1: Generating Two Different Files with the Same MD5 Hash

In this task, we will generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. We can achieve this using the md5collgen program, which allows us to provide a prefix file with any arbitrary content.

First, create a file "prefix.txt" with exactly 64 characters. Then run md5collgen on it.

Command

```
$ python3 -c "print('A'*64,end='')" > prefix.txt
$ md5collgen -p prefix.txt -o out1.bin out2.bin
```

Once the outputs are generated, we check to see if they're different using the diff command, then check their hashes using the md5sum command

Commands

```
$ diff out1.bin out2.bin
$ md5sum out1.bin
$ md5sum out2.bin
```

Question

If the length of the prefix is not a multiple of 64, what happens?

Give your observation with screenshots

Task 2: Understanding MD5's Property

In this task, we will try to understand some of the properties of the MD5 algorithm. For this, we'll use the two files, out1.bin and out2.bin generated in the previous task.

Commands

```
$ tail -c 128 out1.bin > P
$ tail -c 128 out2.bin > Q
```

We now check if P and Q have the same MD5 hash values.

Commands

```
$ md5sum P
$ md5sum Q
```

Next, we create a new suffix and concatenate files with the same MD5 hash with the suffix to generate 2 new files

Commands

```
$ python3 -c "print('114514'*10,end='')" > suffix
$ cat out1.bin suffix > f1
$ cat out2.bin suffix > f2
```

Checking the MD5 hashes of the newly generated files

Commands

```
$ md5sum f1
$ md5sum f2
```

Give your observations with screenshots

Task 3: Generating Two Executable Files with the Same MD5 Hash

In this task, you are given the following C program. Your job is to create two different versions of this program, such that the contents of their xyz arrays are different, but the hash values of the executables are the same.

Code

```
#include<stdio.h>
unsigned char xyz[200] = {
   "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
   "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
   "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
   "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"};
int main() {
    inti;
    for(i = 0; i < 200; i++)
    {
        printf("%x", xyz[i]);
    }
    printf("\n");
}
```

Compile the code, and open the executable in a hex editor

Commands

```
$ gcc task3.c -o task3
$ bless task3
```

In the hex dump, look for the offset of where the array is stored. It should be the start and end offset of a bunch of AAAAs. Ensure you convert them to decimal and note them down for future reference.

Step 1

Truncate the prefix and suffix

Commands

```
$ head -c <start_offset_in_decimal> task3 > prefix
$ tail -c +<end_offset_in_decimal> task3 > suffix
```

Step 2

Generate two files with the prefix and append the suffix to the m to make normal programs. Make them executable.

Commands

```
$ md5collgen -p prefix -o P Q
$ cat P suffix > arr1.out
$ cat Q suffix > arr2.out
$ sudo chmod +x arr1.out arr2.out
```

Step 3

Verify the success of the task

Commands

```
$ ./arr1.out > f1
$ ./arr2.out > f2
# Compare the md5 sums
$ md5sum arr1.out
$ md5sum arr2.out
# Compare the output of the programs
$ diff f1 f2
```

Give your screenshots with observations

Task 4: Making the Two Programs Behave Differently

In the previous task, we have successfully created two programs that have the same MD5 hash, but their behaviours are different. However, their differences are only in the data they print out; they still execute the same sequence of instructions.

In this task, we would like to achieve something more significant and more meaningful.

Assume that you have created a software which does good things. You send the software to a trusted authority to get certified. The authority conducts a comprehensive testing of your software, and concludes that your software is indeed doing good things. The authority will present you with a certificate, stating that your program is good. To prevent you from changing your program after getting the certificate, the MD5 hash value of your program is also included in the certificate; the certificate is signed by the authority, so you cannot change anything on

the certificate or your program without rendering the signature invalid. You would like to get your malicious software certified by the authority, but you have zero chance to achieve that goal if you simply send your malicious software to the authority. However, you have noticed that the authority uses MD5 to generate the hash value. You got an idea. You plan to prepare two different programs. One program will always execute benign instructions and do good things, while the other program will execute malicious instructions and cause damages. You find a way to get these two programs to share the same MD5 hash value.

You then send the benign version to the authority for certification. Since this version does good things, it will pass the certification, and you will get a certificate that contains the hash value of your benign program. Because your malicious program has the same hash value, this certificate is also valid for your malicious program. Therefore, you have successfully obtained a valid certificate for your malicious program. If other people trusted the certificate issued by the authority, they will download your malicious program.

The objective of this task is to launch the attack described above.

Code

```
#include <stdio.h>
#define LEN 300

unsigned char X[LEN] = {
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"};

unsigned char Y[LEN] = {
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"};

int main() {
    for (int i = 0; i < LEN; i++) {
        if (X[i] != Y[i]) {
            printf("i = %d, X[i] = %.2x, Y[i] = %.2x\n", i, X[i], Y[i]);
            printf("Malicious\n");
            return 0;
        }
    }
    printf("Benign\n");
    return 0;
}
```

Step 1

Compile task4.c

Commands

```
$ gcc task4.c -o task4
```

Step 2

Find the start and end offsets of the array X in the executable task4 using bless. Note the offset down in decimals. Then split the executable into prefix and suffix.

Commands

```
$ head -c <start_offset_decimal> task4 > prefix
$ tail -c +<end_offset_decimal> task4 > suffix
```

Step 3

Use md5collgen to generate multiple files from prefix

Commands

```
$ md5collgen -p prefix -o s1 s2
$ tail -c 128 s1 > P
$ tail -c 128 s2 > Q
```

Find the start and end offsets of array Y in suffix using bless. Then, split the suffix into two files based on that offset.

Commands

```
$ head -c <y_start_offset_decimal> suffix > suffix_pre
$ tail -c +<y_end_offset_decimal> suffix > suffix_post
```

Step 4

Construct the final executable and execute it

Commands

```
$ cat s1 suffix_pre P suffix_post > benign
$ cat s2 suffix_pre P suffix_post > evil
$ chmod u+x benign evil
```

Give your screenshots with observations