**MD5 Collision Attack Lab - A Cryptographic Security Seed Lab**

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

**Question 1.** If the length of your prefix file is not multiple of 64, what is going to happen?

It will be padded with zeros.

To test this out, I created a file **hi.txt** and truncated it using **truncate -s YOUR\_DESIRED\_SIZE hi.txt.** After running **md5collgen -p hi.txt -o hi1 hi2** looking at the results using **bless hi1**, we can see that it has been padded with zeros. This is because MD5 processes blocks of size 64 bytes.



**Question 2.**

Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens?

No zero padding is observed.

To see this in action, create a file and use **truncate -s 64 YOUR\_FILE**. Run **md5collgen -p hi -o hi1 hi2** and view the output (**bless hi1 hi2`**).



**Question 3.**

Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different?

No, not all bytes are different. In previous case, the bytes only differ at positions 20, 60, 84 and 110. After multiple trials, it is noted that these differences are not constant.

 

**hi1** **hi2**

**2.2-Task 2: Understanding MD5’s Property**

One interesting point is that you can append the same file to the outputs generated by md5collgen and while it changes the MD5 hash, both the new hashes will be identical. This is because many hash functions like SHA-1, SHA-2 and MD5 are subject to length extension. The hash function has an internal state and processes the message in fixed blocks by applying a compression function to the current state and the current block.

Since the MD5 hashes of both the files were the same, it is safe to assume that the internal state after the algorithm has been run was the same. Let us call this internal state s. So, after appending a file there will be a stage which involves running the compression algorithm on s and the current block, which would be identical for both modified files. This would result in the same MD5 hash for both files.

To test this create a file **hi.txt** and run **md5collgen -p hi.txt -o hi1 hi2**.



Verify that the MD5 hashes are the same using **md5sum hi1 hi2**.



Now let’s append a random string to the end of both files and check the MD5 hashes of both files again.

*echo hi >> hi1*

*echo hi >> hi2*

*md5sum hi1 hi2*



We can see that the MD5 hashes remain identical.

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

Save the following code in a file **program.c**.

#include <stdio.h>

unsigned char a[200] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A'

};

int main()

{

int i;

a[199]='D';

a[198]='N';

a[197]='E';

for(i = 0; i < 200; i++)

{

printf("%x", a[i]);

}

printf("\n");

}

Compile the above with **gcc program.c -o program.out** . The location where the array values are written can easily be identified. 0x1040 works out to be 4160 in decimal. This is divisible by 64 but for now we’ll select a prefix length of 4224 bytes (0x1080). (This is not necessary, but I wanted to have the split in the middle of the array.)

Running **head -c 4224 a.out** > prefix and **md5collgen -p prefix -o agen bgen** ,



Now we have two files with the same MD5 hash but different suffixes. Looking at both agen and bgen in bless ( **bless agen bgen** ).

 

A part of agen A portion of bgen

Now we will get the common end to be appended using **tail -c 4353 a.out > commonend**.

Put them together by running,

*cat commonend >> agen*

*cat commonend >> bgen*

Add executable permission to both files and run them. Note that the output differs.



**2.4-Task 4: Making the Two Programs Behave Differently**

For this, we have to create two programs which have the same MD5 sum, but behave differently. In this example, we will make the programs simply print different statements, however, practically speaking, the second program can execute some malicious code.

This is the program we will be using. Copy the contents and save it as **program.c.**

#include <stdio.h>

unsigned char a[300] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A'};

unsigned char b[300] = { 'H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A','H','A','A','A','A','A','A','A','A','A'};

int main()

{

int i;

int isSame=1;

for(i = 0; i < 200; i++)

{

if(a[i]!=b[i])

isSame=0;

}

if(isSame)

printf("This would run the safe code and display the intended behaviour");

else

printf("This is where malicious code would be run");

printf("\n");

}

Now we compile the program using **gcc program.c**. The output binary is stored by default as a.out. Looking at the contents of a.out using **bless a.out**.



We note that the first array starts at position **0x1040 = 4160**. As before, he prefix can extend until there, but I will continue to use the position **0x1080** so that the generated data will start in the middle of the array.

Since **0x1080** will refer to the first 4224 bytes of the data, we will run **head -c 4224 a.out > prefix** to store the first 4224 bytes in prefix. Then we will run **md5collgen -p prefix -o pgen qgen** which will generate two binaries with the same MD5 sum but they will differ.

Opening both in bless using **bless pgen qgen**.

We can see the 128 characters generated by md5collgen. The MD5 sums of **pgen** and **qgen** will be identical (Verify this: **md5sum pgen qgen**)

We can then select and create two files **p** and **q** having the generated values found in **pgen** and **qgen** respectively. To get the middle section note where the generated data begins in the first array. In the second instance of “HAAAAAAAAAAHAAAA…” (where the second area is located) we place a marker at where the data should be replaced and take it out. (You can do this by creating a new file **head -c +4353 a.out > end** and selecting until the marker.) The resulting **middle** should look like this.

To create the common **end** file, we would need the characters from the **end** file created earlier after accounting for a 128+192 (size of **middle**) offset. Do this by **running tail -c +321 end > commonend** .

Now to put all the files together. Run

*cp pgen benigncode*

*cp qgen maliciouscode*

*cat middle >> benigncode*

*cat middle >> maliciouscode*

*cat p >> benigncode*

*cat p >> maliciouscode*

*cat commonend >> benigncode*

*cat commonend >> maliciouscode*

Verify using **md5sum benigncode maliciouscode** and check if the hashes are the same. Also, make sure the size of **a.out**, **benigncode** and **maliciouscode** are the same.

Finally, run

*chmod +x benigncode maliciouscode*

*./benigncode*

*./maliciouscode*