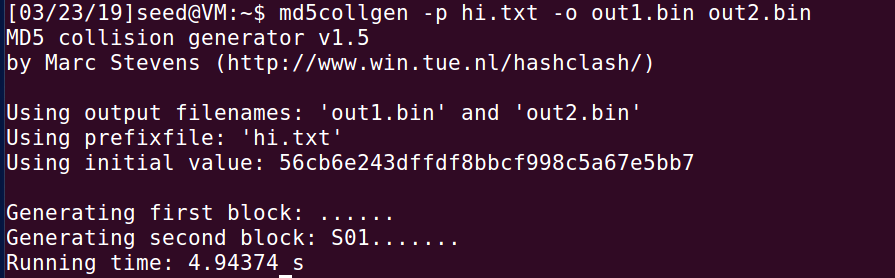
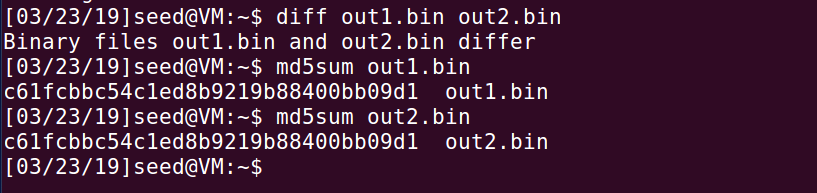
**MD5 Collision Attack Lab**

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

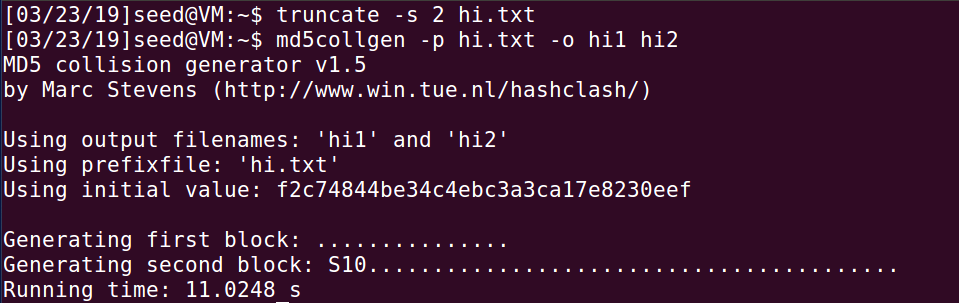


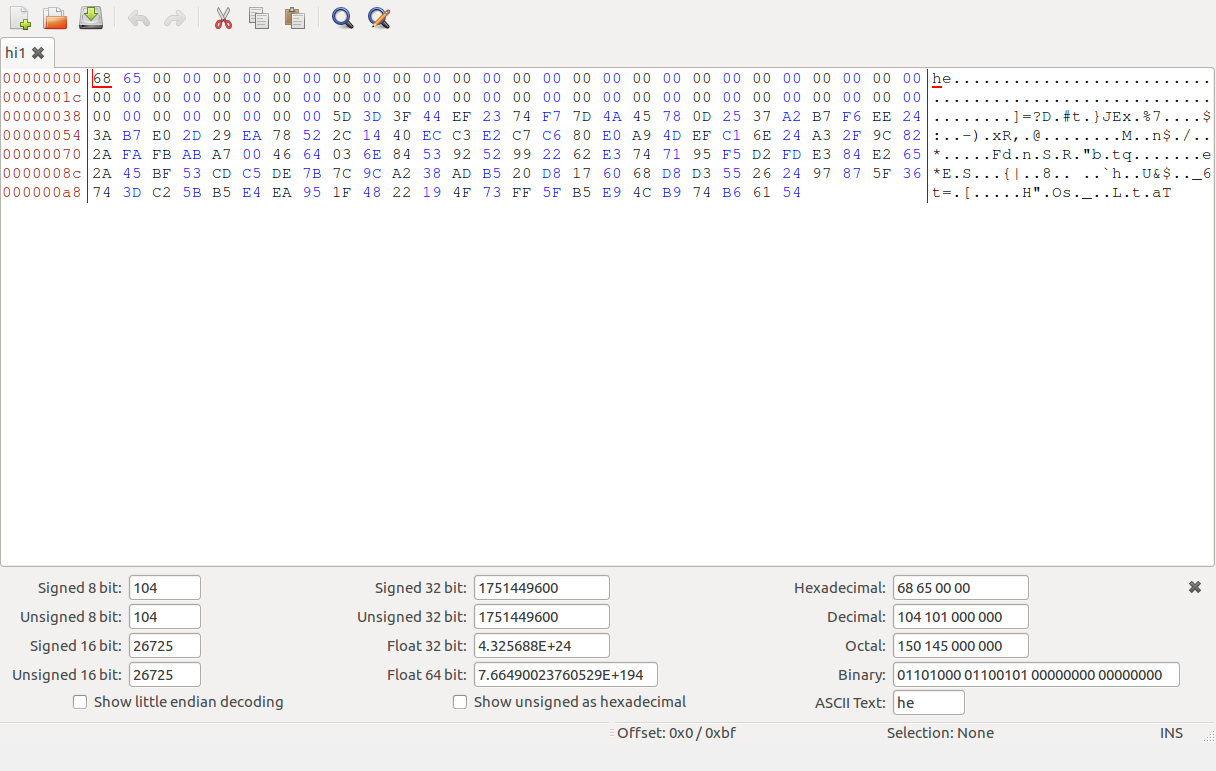




**Question 1. If the length of your prefix file is not multiple of 64, what is going to happen?**

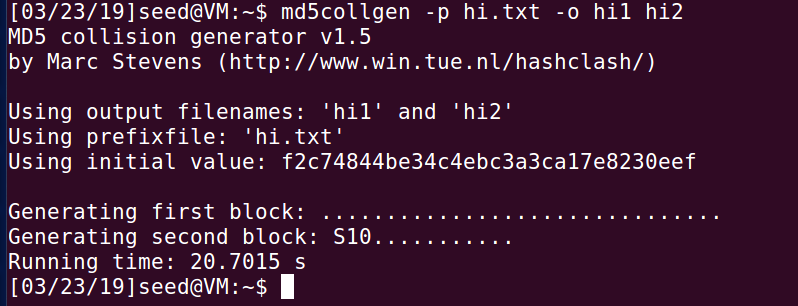
If the length of the prefix file is not 64, then the file will be padded with zeros. To see this, I created a file hi.txt and truncated it using truncate -s 2 hi.txt. After running md5collgen -p hi.txt -o hi1 hi2 and looking at the results using bless hi, we can see that it has been padded with zeros. This is because MD5 processes blocks of size 64.

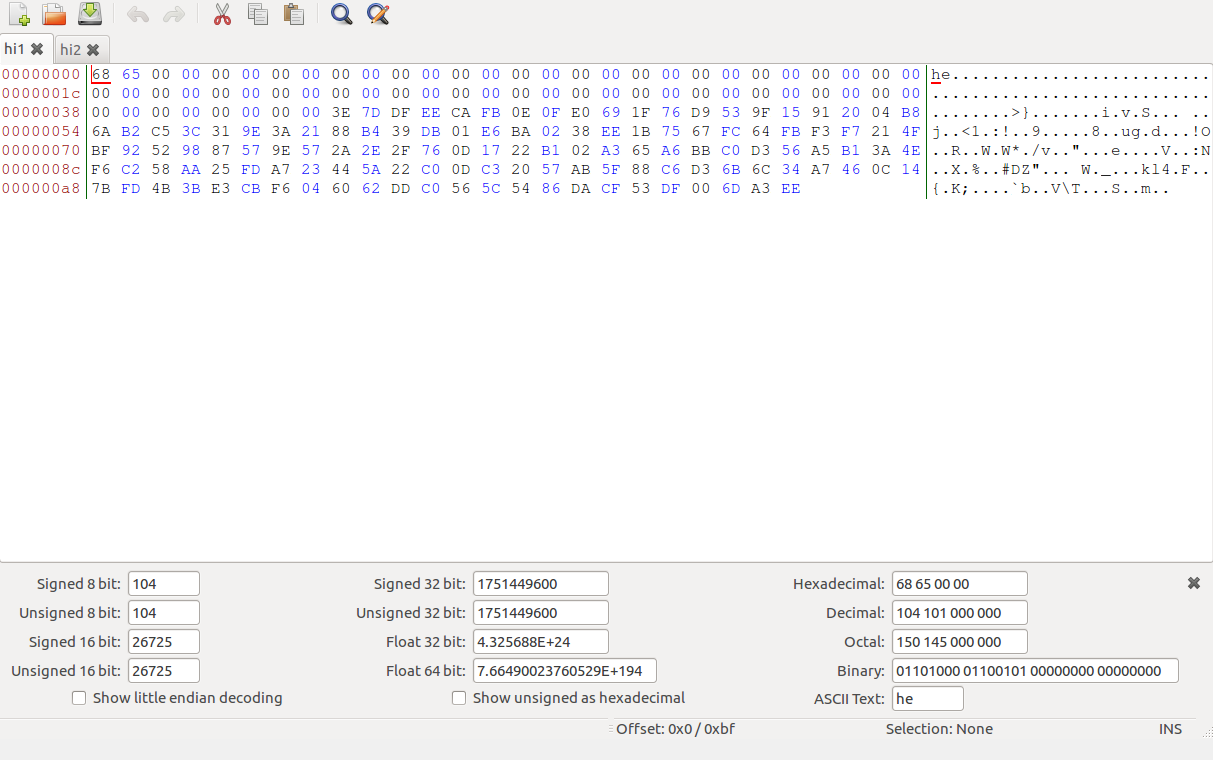




**Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.**

No zero padding is observed. To see this, create a file hi.txt and truncate -s 64 hi.txt and run md5collgen -p hi.txt -o hi1 hi2 and view the output using bless hi1 hi2.

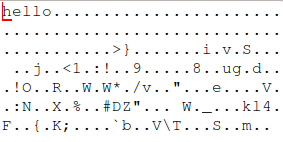




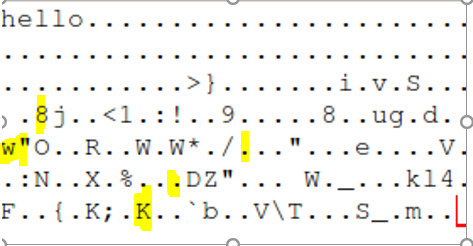
No zero padding is observed

**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.**

The two files generated are slightly different. In the previous case, the bytes differ only at 79, 84, 123,147 and 174. After multiple trials, it is clear that these differences are not constant



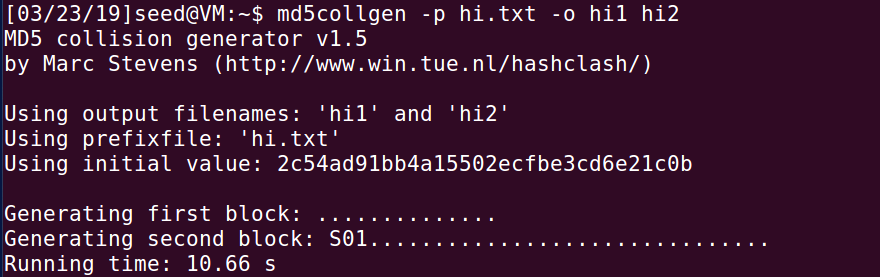
Hi1



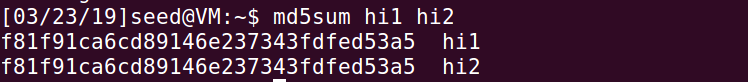
Hi2

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

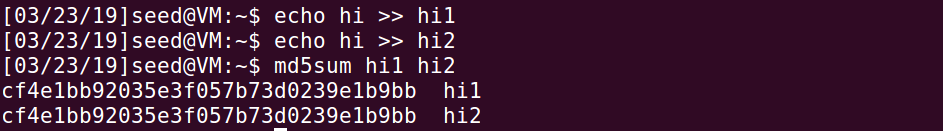
To test this, create a file hi.txt and run $md5collgen -p hi.txt -o hi1 hi2.



Verify that 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.



The above snapshot clearly shows that MD5 hashes remain identical.

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

I created two different versions of this program, such that the contents of the arrays are different.

#include <stdio.h>

unsigned char xyz[200] = {

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

};

int main()

{

int i;

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

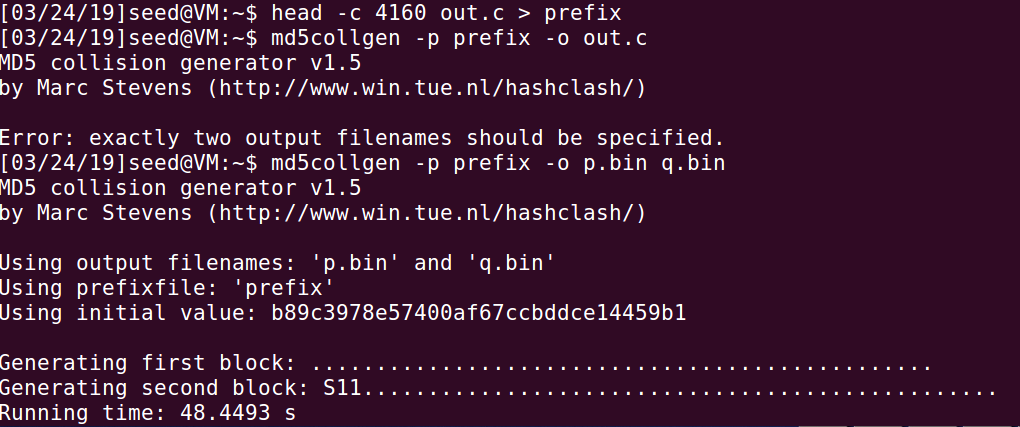
printf("%x", xyz[i]);

}

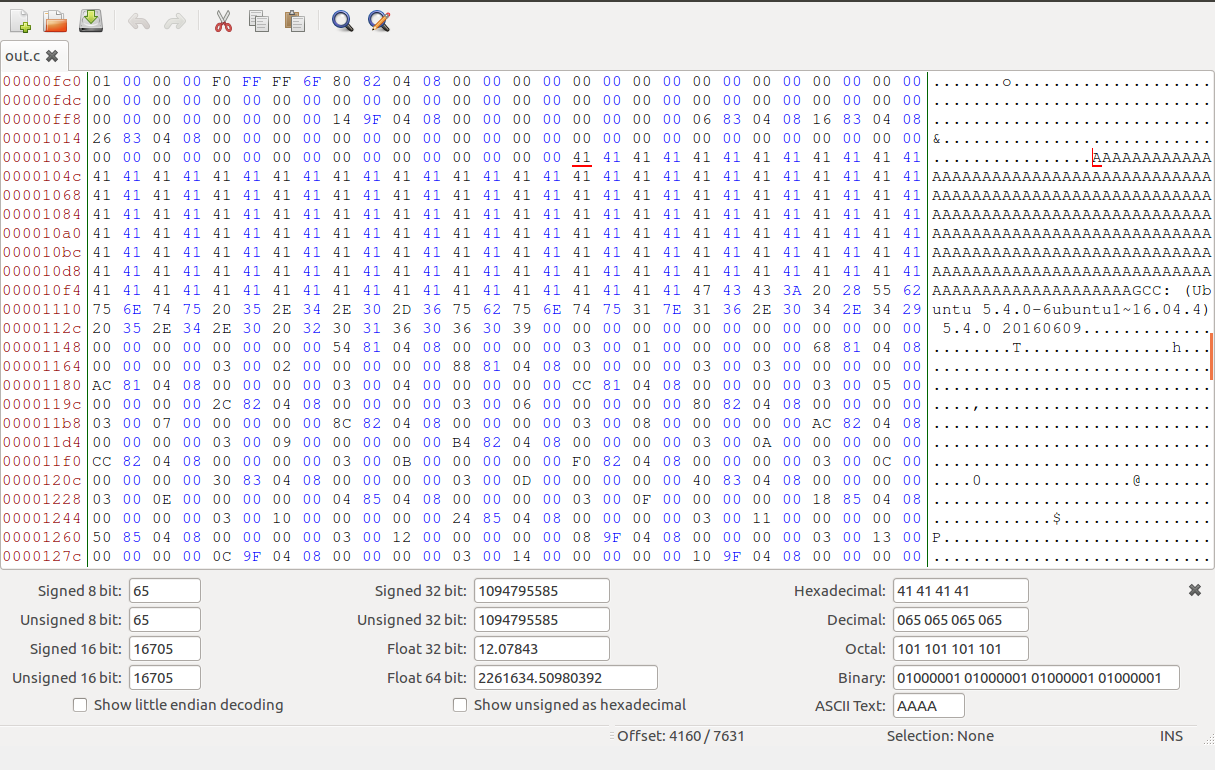
printf("\n");

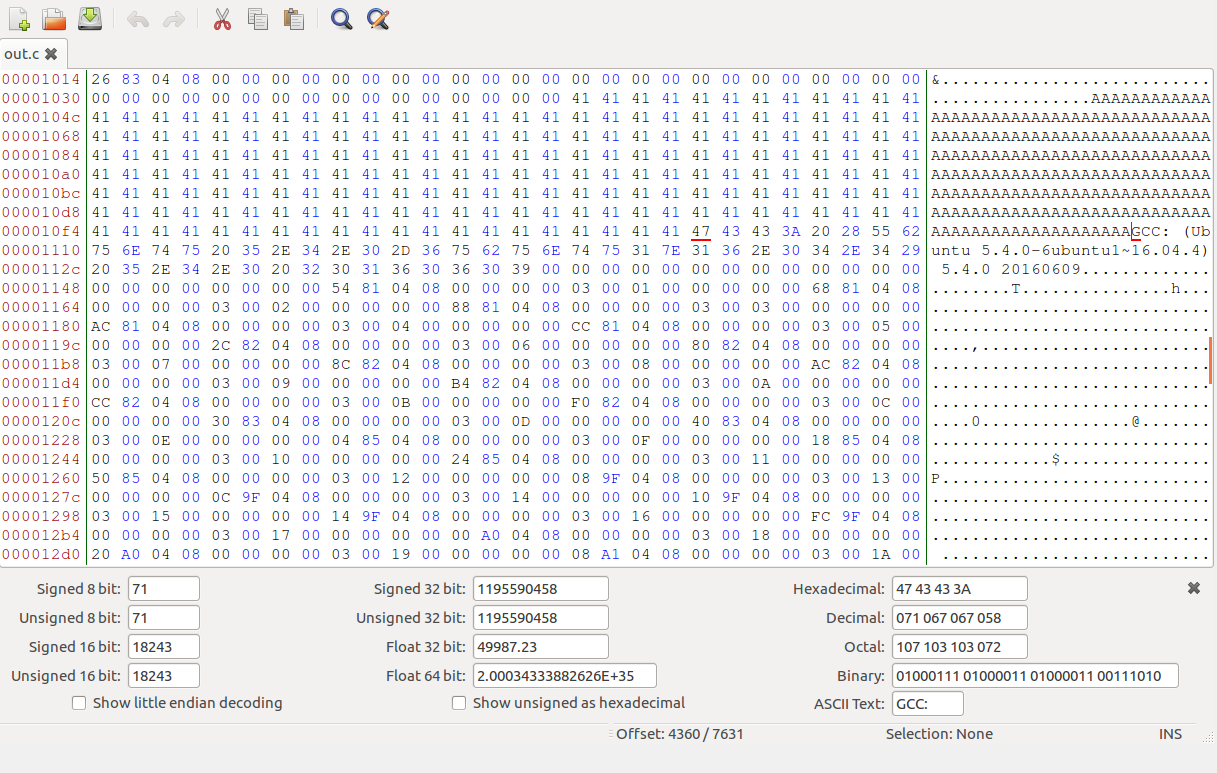
}

Running the code head -c 4160 a.out > prefix and md5collgen -p prefix -o out.c



Now we have two files with the same MD5 hash but different suffixes. Looking at out.c in bless.



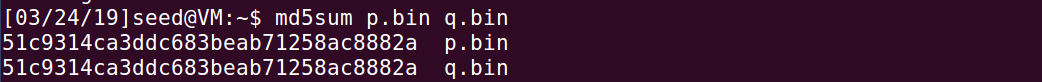


The common end to be appended is done using the command tail -c 4360 out.c > suffix.

We run them together using the following commands:

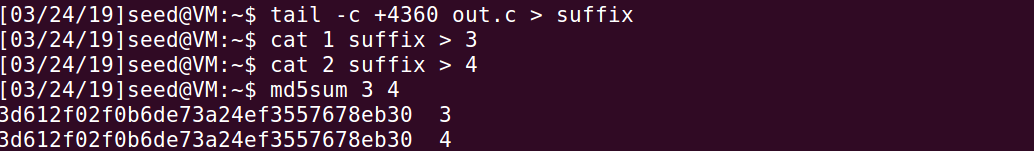
Cat 1 suffix >> 3

Cat 2 suffix >> 4









Conclusion: Thus, the hash values of the executables are same.

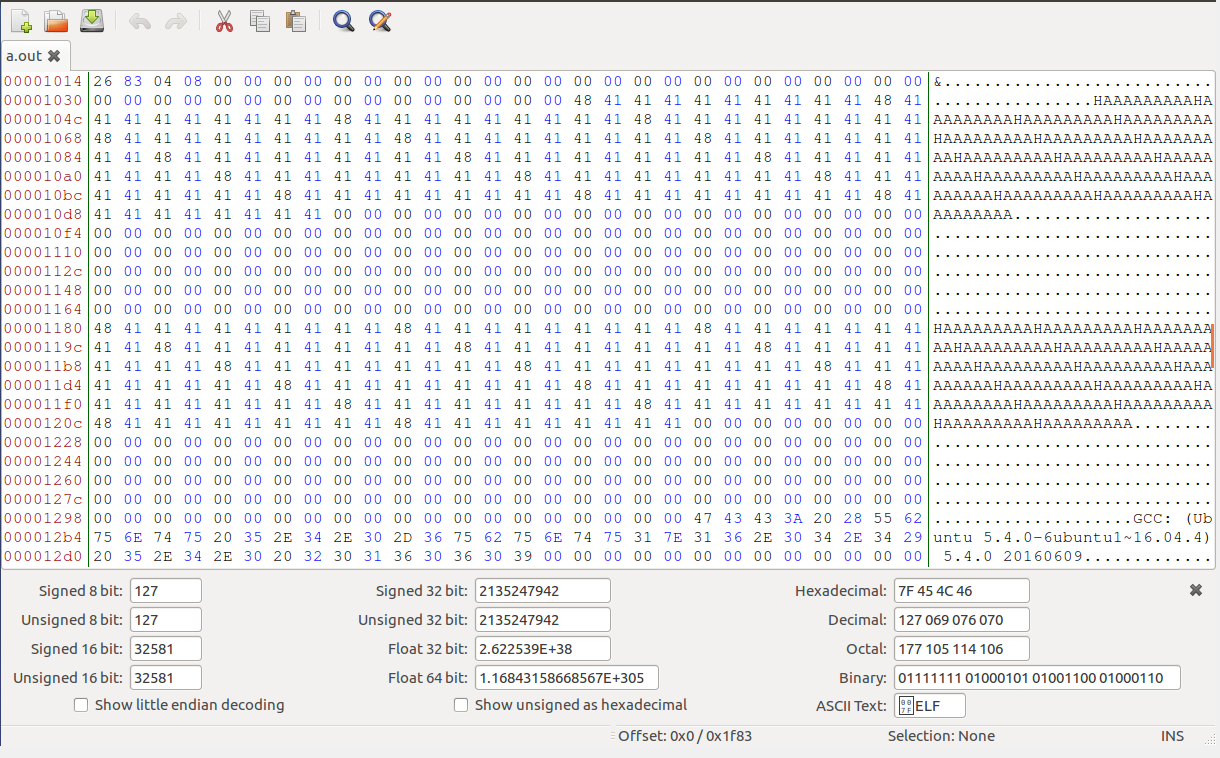
**2.4 Task 4: Making the Two Programs Behave Differently**

This task requires us to create two separate programs which have the same MD5 sum but, behave differently. So, I’ll simply make the program print different statements and practically speaking, the second program can execute some malicious code.

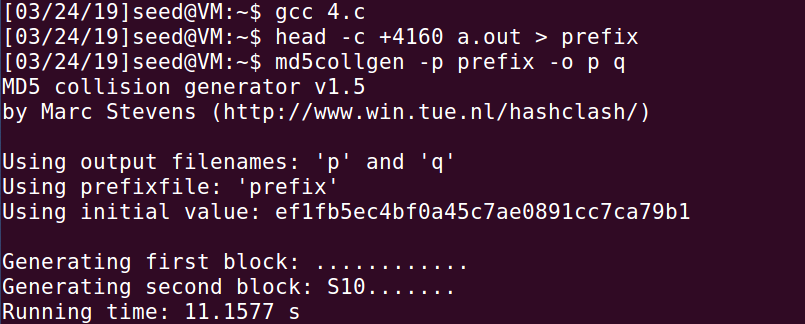
#include<stdio.h>

|  |  |
| --- | --- |
| # | 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("run benign code"); |
|  | else |
|  | printf("run malicious code"); |
|  | printf("\n"); |
|  | } |

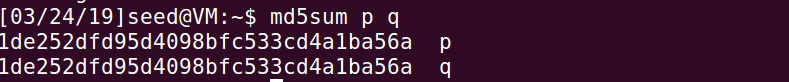
I compile the program as “gcc p24.c” and view the contents of the output binary file as “bless a.out”.



Since, we refer to the first 4224 bytes of data, we will run the code head -c 4160 a.out > prefix to store the first 4224 bytes in prefix. Then, we run mdcollgen -p prefix -o p q to generate two binaries with the same MD5 sum but they will differ.



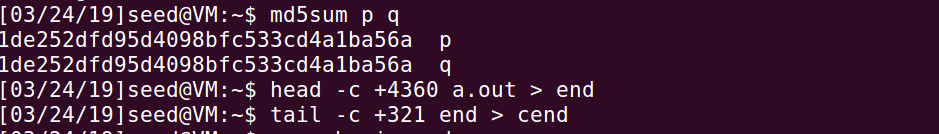
Opening both the files using bless p and q we can see the 128 characters generated by md5collgen. Let’s verify this using md5sum p q



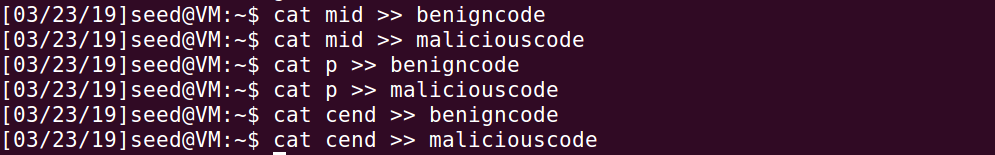
We can then select and create two files **p** and **q** having the generated values found in **p** and **q** respectively. To get the middle section note where the generated data begins in the first array. In the second instance of “HAAAAAAAAAAHAAAA…” (where the second area is located) we place a marker at where the data should be replaced and take it out*.*I did this by creating a new file head -c +4360 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 4480-4160 = 320 ( 4480 is a position where the first array along with the gap ends and 4160 is a position where the first array begins). I did this by running tail -c +321 end > cend (321 is a position where the second array begins).

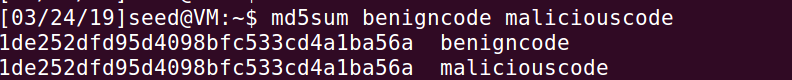
Now to put all the files together. Run



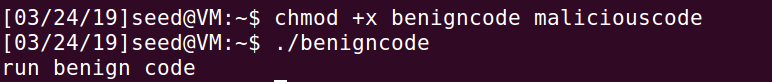


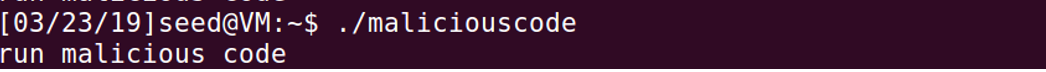


We verify the above using md5sum benigncode maliciouscode and check if the hashes are same.



Lastly, we run





Conclusion: If I change the contents of the two arrays, then the hash values are not same which is shown in the snapshots provided below:

Here, I changed three values of the array.

#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','H','H','A','H','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 flag = 1;

int i;

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

{

//printf("%c ", a[i]);

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

{

flag = 0;

break;

}

}

if(flag)

printf("good code!\n");

else

printf("run benigncode!\n");

return 0;

}

