Lab Title	Author	Date	Class	Section
MD5 Attack Collison Lab	Rushabh Prajapati	19/01/2022	CMPT 380 Computer Software Security	

Task 1

Generating Two Different Files with the Same MD5 Hash

Observation:

After running the **md5collgen,** with the prefix.txt, we can see that the md5 collgen executable generated two output files out1.bin out2.bin, with the same initial value.

The whole process took about 36.3359 s, to generate the out{1..2}.bin files.

```
[01/12/22]seed@VM:~/.../Labsetup$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 8f8bc32955226577455d8c879f85b7d2

Generating first block: ........................
Generating second block: W.........
Running time: 36.3359 s
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 1 Running md5collgen

```
[01/12/22]seed@VM:~/.../Labsetup$ ls
md5collgen  out1.bin  out2.bin  prefix.txt
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 2 Generated 2 out{1..2}.bin files

```
[01/12/22]seed@VM:~/.../Labsetup$ diff out{1..2}.bin
Binary files out1.bin and out2.bin differ
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 3 Using diff to see if the 2 files are same or not

```
[01/12/22]seed@VM:~/.../Labsetup$ md5sum out{1..2}.bin
4b28a31900c98ddde057a86e83b52bb0  out1.bin
4b28a31900c98ddde057a86e83b52bb0  out2.bin
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 4 Checking md5sum of bot the out.bin files

From Figure 1 to Figure 4, we can conclude that the out{1..2}.bin files generated by the md5collgen tool, are different from one another, i.e both are different files, however, after running the md5sum (an inbuilt linux utility) we can see that the md5sum hash value of both the out1.bin and out2.bin are same. Thus, this program violated the collision-resistance property of md5, which states that no two files can have the same md5 hash, i.e hash(out1.bin) != hash(out2.bin), but in this case the collision-resistance property failed.

Answers:

- Question1: If the length of your prefix file is not multiple of 64, what is going to happen?
  - The output files out1.bin and out2.bin will be padded with zeroes.
  - For example, the current **prefix.txt** file has the **length of 12 bytes**, and we used this prefix.txt file to generate the out{1..2}.bin files,

```
[01/12/22]seed@VM:~/.../Labsetup$ cat prefix.txt
Testing MD5
[01/12/22]seed@VM:~/.../Labsetup$ stat -c %s "prefix.txt"
12
[01/12/22]seed@VM:~/.../Labsetup$ stat -c %s "out1.bin"
192
[01/12/22]seed@VM:~/.../Labsetup$ stat -c %s "out2.bin"
192
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 5 Content of prefix.txt and file sizes of each prefix.txt, out{1..2}.bin

As, we can see here the file size of prefix.txt is 12 bytes, not a multiple of 64. A property of the md5 hashing algorithm is to generate fixed sized blocks of 64 bytes, no matter what's the length of the data/file, in this case as the file size is less that 64 bytes, it will be padded with zeroes after the 12 bytes of original data.

We can check this by using the **bless** utility on the out{1.2}.bin files.

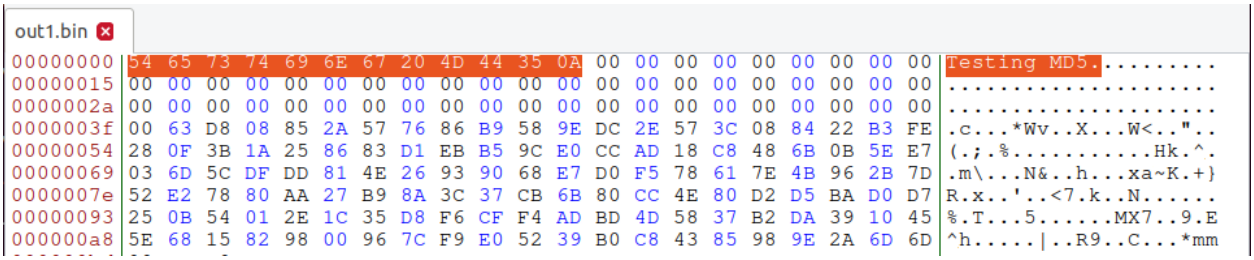

Figure 6 bless out1.bin, as you can see after 12 bytes all the out bytes of the file is padded with zeroes, inorder to generate a fixed 64 byte size block

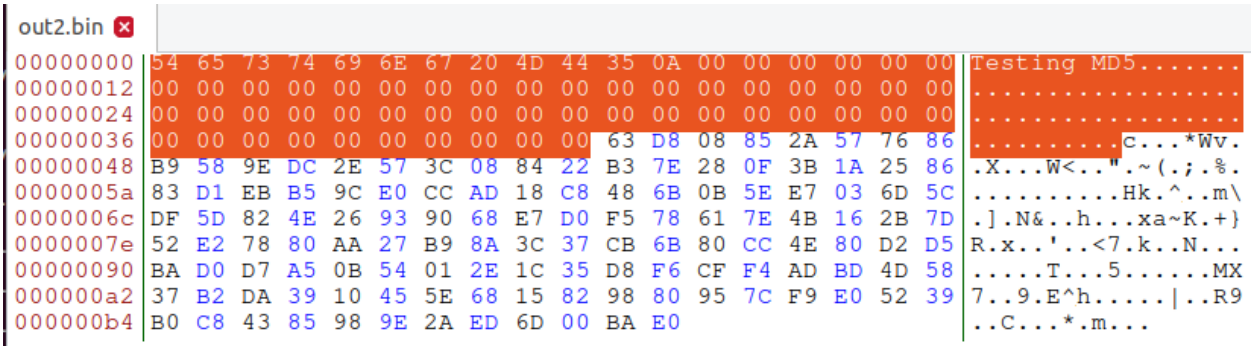

Figure 7 Similarly, bless out2.bin, as you can see after 12 bytes all the out bytes of the file is padded with zeroes, inorder to generate a fixed 64 byte size block

For a file size of 132 bytes, the padding can be more visible,

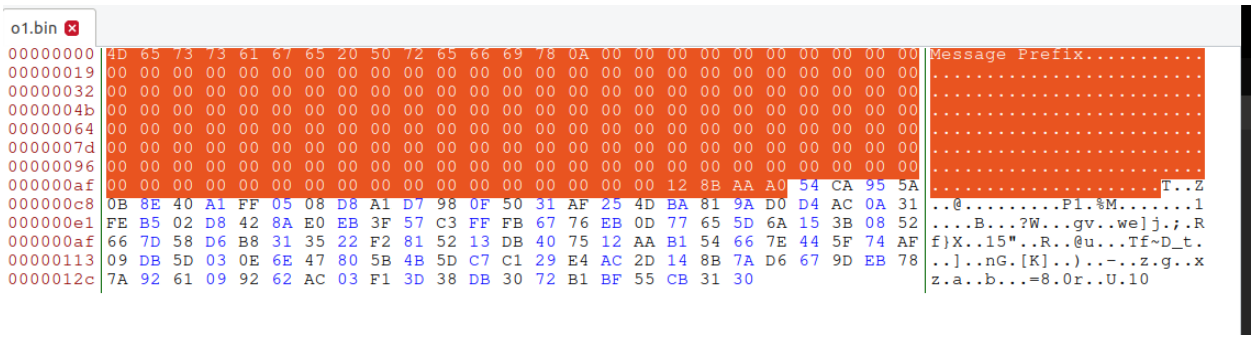

Figure 8 A o1.bin file generated from a pre_132.txt file of 132 bytes

- Question2: Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.
  - No padding of zeroes is observed inside the files out1.bin and out2.bin
  - In order, to create a file size of 64 bytes, I used the inbuilt linux utility, **truncate** which can be used to shrink or extend the size of a file to a specified size (man truncate).

```
[01/12/22]seed@VM:~/.../Labsetup$ truncate -s 64 prefix.txt
[01/12/22]seed@VM:~/.../Labsetup$ stat prefix.txt
  File: prefix.txt
  Size: 64              Blocks: 8          IO Block: 4096   regular file
Device: 805h/2053d      Inode: 1460461     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    seed)   Gid: ( 1000/    seed)
Access: 2022-01-12 01:53:31.687530365 -0500
Modify: 2022-01-12 12:49:26.108829856 -0500
Change: 2022-01-12 12:49:26.108829856 -0500
 Birth: -
[01/12/22]seed@VM:~/.../Labsetup$ stat -c %s prefix.txt
64
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 9 Changed the file size of prefix.txt to 64 bytes

Observation:

After generating 2 new out.bin files with the new truncated prefix.txt file,

```
[01/12/22]seed@VM:~/.../Labsetup$ md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: 8f8bc32955226577455d8c879f85b7d2

Generating first block: ................................
Generating second block: S11................................
Running time: 58.0154 s
[01/12/22]seed@VM:~/.../Labsetup$
```

We can observe that for 64 bytes prefix.txt file the time taken to generate the blocks was increased from 36.3359s to 58.0154s, because our original prefix.txt file was only 12 bytes. Thus, the higher the length of the prefix file, the more time it will take to generate the blocks.

Let's run **bless** on, the out{1..2}.bin files, we can find that, no padding of zeroes is observed if the file size is exactly 64 bytes, I,e the **.bin files**, when examined through bless does not have padding of zeroes, prefix+suffix.

```
[01/12/22]seed@VM:~/.../Labsetup$ ls -al prefix.txt out{1..2}.bin
-rw-rw-r-- 1 seed seed 192 Jan 12 12:52 out1.bin
-rw-rw-r-- 1 seed seed 192 Jan 12 12:52 out2.bin
-rw-rw-r-- 1 seed seed  64 Jan 12 12:49 prefix.txt
[01/12/22]seed@VM:~/.../Labsetup$
```

Figure 10 64 bytes prefix

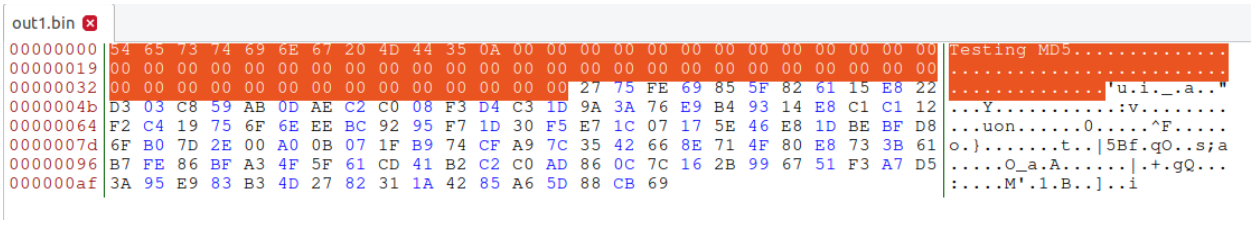

Figure 11 out1.bin -> prefix(64 bytes) +suffix(data)(128 bytes)

Similarly, for our2.bin,

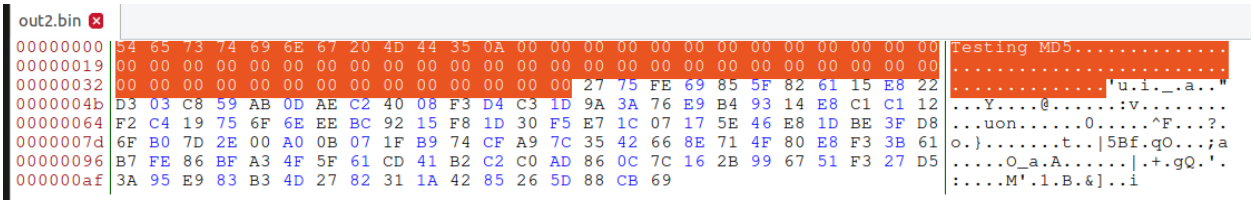

Figure 12 out2.bin -> prefix(64 bytes) +suffix(data)(128 bytes)

- Question3 Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

```
[01/12/22]seed@VM:~/.../Labsetup$ python3 bytes.py out1_128 out2_128
                          out1_128 ----- out2_128
Byte Difference at 0x53 : 0xc0 ----- 0x40
Byte Difference at 0x6d : 0x95 ----- 0x15
Byte Difference at 0x6e : 0xf7 ----- 0xf8
Byte Difference at 0x7b : 0xbf ----- 0x3f
[01/12/22]seed@VM:~/.../Labsetup$
```

A python program that read 2 files byte by byte and returns the different bytes from the out1.bin and out2.bin, we can see that the bytes differ at position 0x53, 0x6d, 0x6e,0x7b, in decimal at byte 83, 109,110 and 123 respectively.

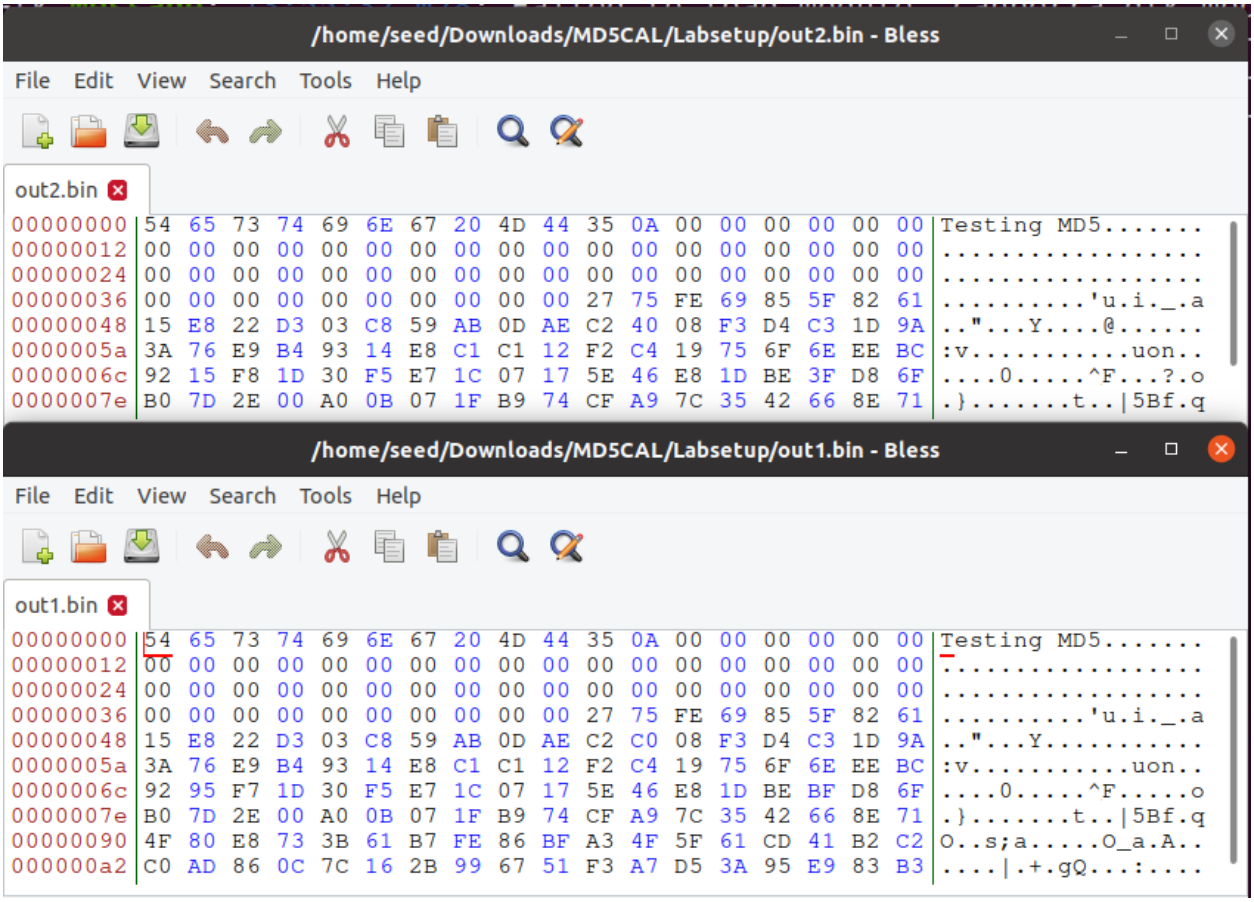

Figure 13 out2.bin and out1.bin

Task 2

Understanding MD5's Property

Observation:

```
[01/12/22]seed@VM:~/.../Task2$ ls
bytes.py  md5collgen  out1_128  out1.bin  out2_128  out2.bin  prefix.txt
[01/12/22]seed@VM:~/.../Task2$ echo "Message Suffix" > suffix.txt
[01/12/22]seed@VM:~/.../Task2$ cat out1.bin suffix.txt > out1_long.bin
[01/12/22]seed@VM:~/.../Task2$ cat out2.bin suffix.txt > out2_long.bin
[01/12/22]seed@VM:~/.../Task2$ diff out1_
out1_128       out1_long.bin
[01/12/22]seed@VM:~/.../Task2$ diff out1_long.bin out2_long.bin
Binary files out1_long.bin and out2_long.bin differ
[01/12/22]seed@VM:~/.../Task2$
```

Figure 14 Create a suffix.txt file and concatenate it with out1.bin and out2.bin

```
[01/12/22]seed@VM:~/.../Task2$ md5sum out1_long.bin
50dbd8eb1d21855b297af50f252c1a11  out1_long.bin
[01/12/22]seed@VM:~/.../Task2$ md5sum out2_long.bin
50dbd8eb1d21855b297af50f252c1a11  out2_long.bin
[01/12/22]seed@VM:~/.../Task2$
```

Figure 15 out1_long.bin and out2_long.bin with the same md5sum after concatenating the same suffix.txt value

This property of MD5 is called the **Length Extension Property**, which states that if 2 input files have the same hash value, hash(file1) = hash(file2), but the suffix part is a bit random, if we concatenate the same suffix to the bin files, that will result in 2 different files still with the same hash value. This happens because of the construction function, Merkel-Damgard , used in MD5, as both the files have the same hash value, even though we add the same suffix value, but as all this depends on the previous iteration of the hash value which is same for both the file, we end up with the a hash value which is same for both the new files with the extended suffix. Thus, making MD5 vulnerable to **Length Extension Attack**.

Task 3

Generating Two executable Files with the Same MD5 Hash

Observation:

We first compile the C program, with 200 A's, then we use **bless** to analyse the **print_array** executable file and find the offset of A's.

Using bless we can see that to find a multiple of 64, we start with the offset 12352. So, we'll cut from here.

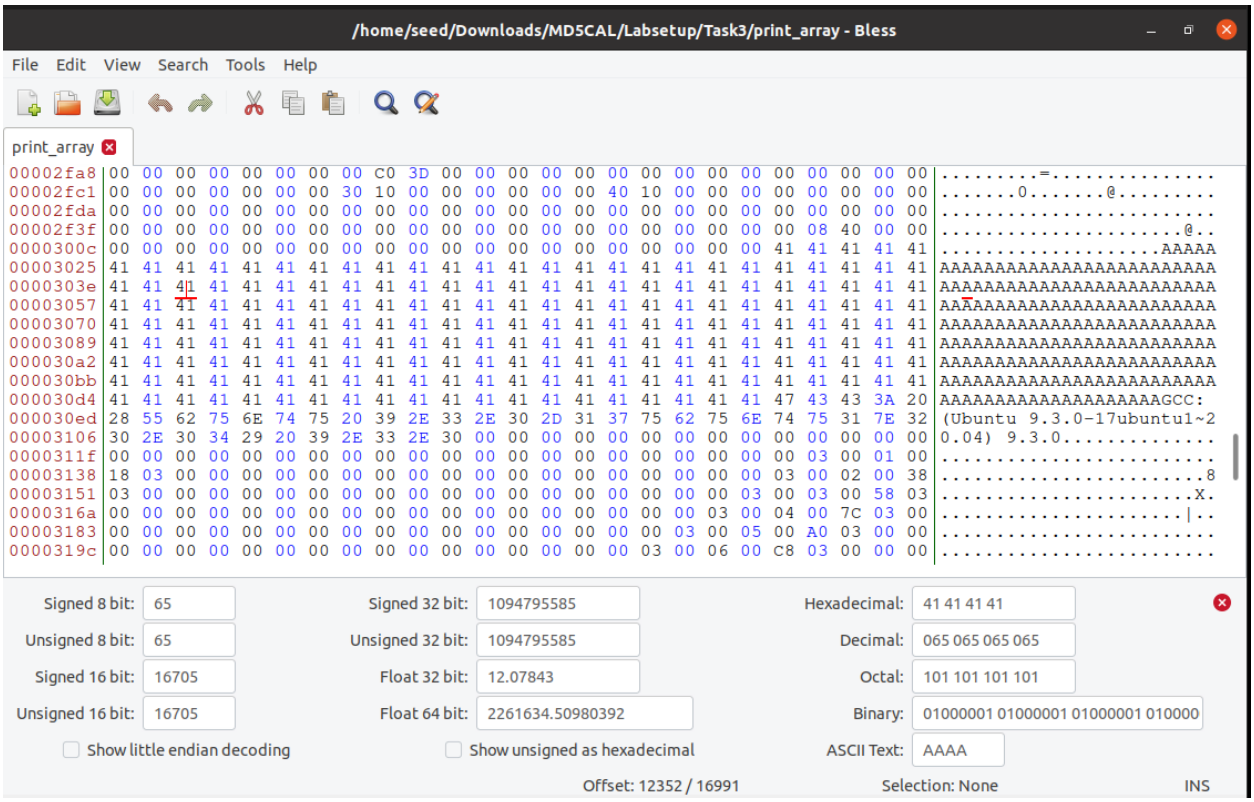

Figure 16 Offset 12352 multiple of 64

Using the head utility, we cut the **prefix** from offset 12352, i.e. the first 12352 bytes of the binary executable **print_array**, then we use the tail utility to cut the **suffix** from the binary executable print_array. The suffix is the region ,12352+128 = 12480 bytes to the end of the binary executable.

The in between 128 byte region will be reserved for the content that we will replace, in place of the A's.

```
[01/12/22]seed@VM:~/.../Task3$ head -c 12352 print_array > prefix
[01/12/22]seed@VM:~/.../Task3$ head -c +12480 print_array > suffix
[01/12/22]seed@VM:~/.../Task3$ ls
prefix  print_array  print_array.c  suffix
[01/12/22]seed@VM:~/.../Task3$ tail -c +12480 print_array > suffix
[01/12/22]seed@VM:~/.../Task3$ ls
prefix  print_array  print_array.c  suffix
[01/12/22]seed@VM:~/.../Task3$ cp ../md5collgen .
[01/12/22]seed@VM:~/.../Task3$ md5collgen -p prefix -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix'
Using initial value: b10420a625760352e425335b24b79722

Generating first block: ............................
Generating second block: S10.......................
Running time: 76.2861 s
[01/12/22]seed@VM:~/.../Task3$
```

Figure 17 Using head and tail to create a prefix and suffix file for the executables

```
[01/12/22]seed@VM:~/.../Task3$ md5sum out{1..2}.bin
3b9d9edf5c569126994055adecd27891  out1.bin
3b9d9edf5c569126994055adecd27891  out2.bin
[01/12/22]seed@VM:~/.../Task3$ tail -c 128 out1.bin > P
[01/12/22]seed@VM:~/.../Task3$ tail -c 128 out2.bin > Q
[01/12/22]seed@VM:~/.../Task3$ md5sum P Q
4fd5f647047a4436a1a6e105a8c9b264  P
fe43447d68fc97edb28fd40b2d4c7516  Q
[01/12/22]seed@VM:~/.../Task3$
```

Figure 18 Checking md5sum of ou1.bin and out2.bin created from md5collgen with file prefix and the extracting the last 128 bytes of data from out1.bin and out2.bin for file P and Q which are going to be used to replace the content of the array.

Once we got the prefix and the suffix, now we need to generate 2 **.bin** files with the same MD5 hash value, which we can get by using the **md5collgen** utility. Now, to create 2 binary executables with the same hash value but different content, we will extract the **last 128 bytes** of the out1.bin and out2.bin, calling them as **P and Q** as these are the bytes that differ in those binary out1.bin and out2.bin files. And we will replace these 128 bytes in the **128-byte-region**, in the **print_array** executable with **P and Q,** instead of the A's.

Thus we are ready to create 2 executable files, based on the property of the MD5 algorithm of if we have add the same prefix or suffix to the files with the same hash

value, even after stitching the prefix and the suffix their hash value will remain the same. Below in Figure 19, we can see that the using the **cat** utility in linux we **concatenate file P and Q** with the same **prefix and suffix** and generate 2 file **a1.out and a2.out**, using **chmod** we make these files executable, and when we check the **md5 hash value** a1.out and a2.out has the same hash value but both these binary file differ. Thus, we were able to create 2 different versions of the same C program, such that the content of their xyz array were different, but the hash value of the executables are the same.

After running the MD5 collision tool , we get outl.bin and out2 . b in, which have the same MD5 hash value. We then take out the last 128 bytes from these two files , and save them to files P and Q, respectively

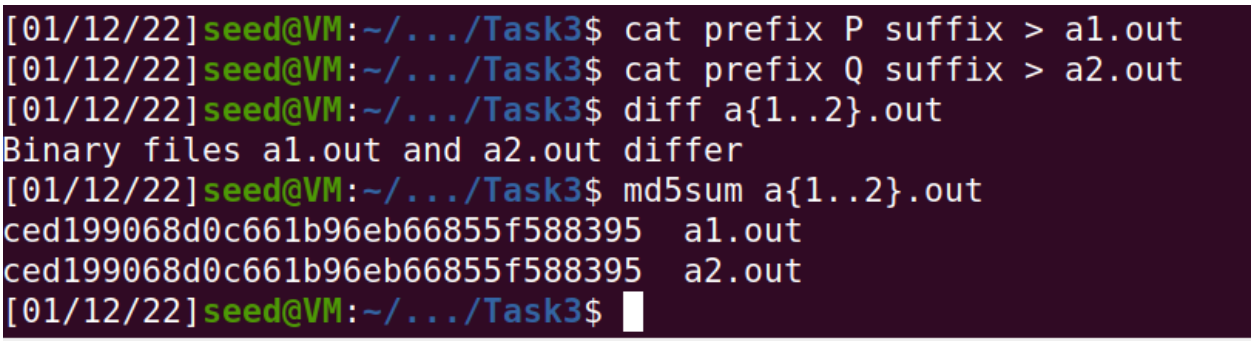

Figure 19 Using the cat utility we created 2 binary files a1.out and a2.out with the same hash value but different content

Task 4

Making the Two Programs Behave Differently

Observation:

```
#include<stdio.h>
#define LENGTH 200

unsigned char X[LENGTH] = {
        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,
        };

unsigned char Y[LENGTH] = {
        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,
test_file.c
```

Figure 20 Main Program with 2 arrays, test_file.c

```
int main() {
        int i = 0;
        for(i=0; i<LENGTH;i++) {
                if (X[i] != Y[i]){ break;}
        }

        if (i == LENGTH) {
                printf("%s\n","Executing benign Code");
        }
        else {
                printf("%s\n", "Executing malicious Code");
        }
        return 0;
}
test_file.c
```

Figure 21 Diffrentiate between benign and malicious code

After compiling the above the code, we get a **test_file** executable. Using the same technique, we used in the task above we would first split the file **into prefix and suffix**.

Since the first multiple of 64 bytes into the array is found at 12352, that will be our prefix, then we leave 128 bits for the file P or Q, then so that makes 12352 + 128 = 12480. So, for our suffix we have the value +12481 to the end of the file.

Once we do that, we, can use the **md5collgen** tool to generate 2 **out1 and out2** files. We generate the hash collision using prefix and save the last 128 bytes of the output files to P and Q, respectively.

Because we need to modify the suffix using P, we further break suffix into two pieces: a piece before P is placed and another one after P. ln our program, the first piece of the suffix is the first 96 bytes, so the second piece is 96 + 128 = 224 to the end of the file. So, from +225 to the end of the will be the part **end** from **suffix.** So, the suffix is divided into **middle + P + end.**

```
[01/19/22]seed@VM:~/.../testmd5$ cat Task4_md5
head -c 12352 test_file > prefix
tail -c +12481 test_file > suffix
md5collgen -p prefix -o out1 out2
tail -c 128 out1 > P
tail -c 128 out2 > Q
head -c 96 suffix > middle
tail -c +225 suffix > end
cat prefix P middle P end > a1.out
cat prefix Q middle P end > a2.out
[01/19/22]seed@VM:~/.../testmd5$
```

Figure 22 Test Tasks fro generating prefix, suffix, P, Q, and further dividing the suffix into two more part

Now, the after generating all the required files, we'll put everything back together. Using

prefix, P, middle, P, end -> benign code -> a1.out

prefix, Q, middle, P, end -> malicious code -> a2.out

Now as you can see below when we run both 2 **programs a1.out goes into benign branch** and **a2.out goes into malicious branch.** We can also see that the md5sum of the program are the same, yet they behave differently.

```
Change: 2022-01-19 03:54:33.044265623 -0500
 Birth: -
[01/19/22]seed@VM:~/.../Task4$ head -c 12352 test_file > prefix
[01/19/22]seed@VM:~/.../Task4$ tail -c +12481 test_file > suffix
[01/19/22]seed@VM:~/.../Task4$ rm out*
[01/19/22]seed@VM:~/.../Task4$ md5collgen  -p prefix -o out1 out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1' and 'out2'
Using prefixfile: 'prefix'
Using initial value: fb7fbbf724e931b329025864e3e602d5

Generating first block: .......
Generating second block: S10.
Running time: 9.56115 s
[01/19/22]seed@VM:~/.../Task4$ tail -c 128 out1 > P
[01/19/22]seed@VM:~/.../Task4$ tail -c 128 out2 > Q
[01/19/22]seed@VM:~/.../Task4$ bless suffix
Gtk-Message: 04:36:05.692: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find a part of the path '/home/seed/.config/bless/plugins'.
Could not find file "/home/seed/.config/bless/export_patterns"
```

Figure 23 Generating prefix, suffix, out1, out2 and P and Q.

```
[01/19/22]seed@VM:~/.../Task4$ chmod u+x a1.out a2.out
[01/19/22]seed@VM:~/.../Task4$ ./a1.out
Executing malicious Code
[01/19/22]seed@VM:~/.../Task4$ ./a2.out
Executing malicious Code
[01/19/22]seed@VM:~/.../Task4$ rm a1.out a2.out
[01/19/22]seed@VM:~/.../Task4$ tail -c +225 suffix > end
[01/19/22]seed@VM:~/.../Task4$ cat prefix P middle P end > a1.out
[01/19/22]seed@VM:~/.../Task4$ cat prefix Q middle P end > a2.out
[01/19/22]seed@VM:~/.../Task4$ chmod u+x a1.out a2.out
[01/19/22]seed@VM:~/.../Task4$ ./a1.out
Executing benign Code
[01/19/22]seed@VM:~/.../Task4$ ./a2.out
Executing malicious Code
```

Wireshark

Figure 24 Executing the output files, a1.out being the benign code and a2.out being the malicious code

```
[01/19/22]seed@VM:~/.../Task4$ md5sum a1.out a2.out
a221d73cae86a25b8dc788cc5ac4da25  a1.out
a221d73cae86a25b8dc788cc5ac4da25  a2.out
[01/19/22]seed@VM:~/.../Task4$
```

Figure 25 md5sum and execution of a1.out and a2.out

As you can see that the md5sum of both the files a1.out and a2.out is the same. Thus, we were successful in creating a program with the same md5 hash value but behaving differently.