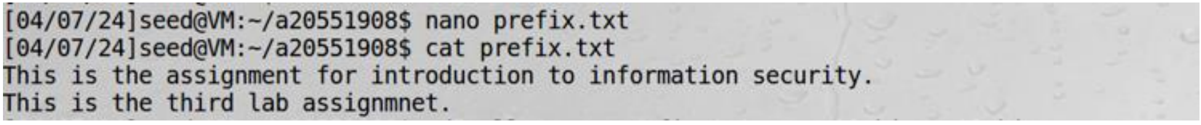
Introduction to Information Security - CS 458 - Spring 2024

CWID: A20551908

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

We created two distinct files for this task that have the same MD5 hash values. These two files have the same prefix at the beginning, making them identical. We have used the md5collgen program to do this.

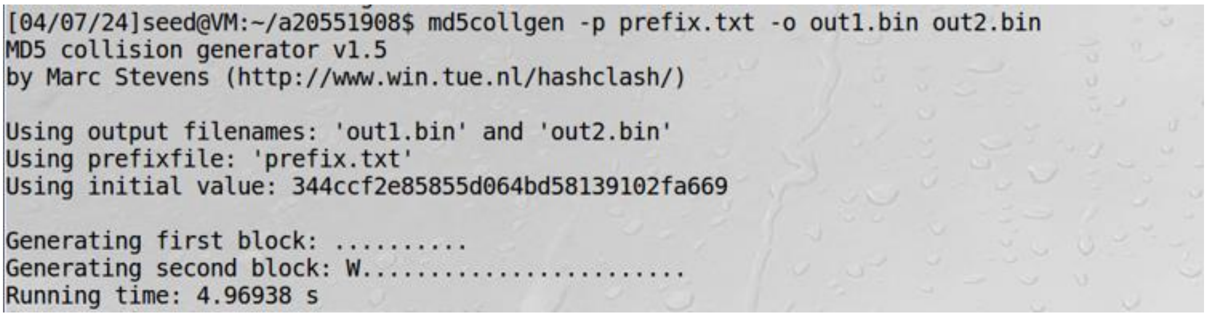
Creating a file named ‘prefix.txt’ and writing the content in to the file using nano command. Viewing the contents of the file using ‘cat’ command.



```
[04/07/24]seed@VM:~/a20551908$ nano prefix.txt
[04/07/24]seed@VM:~/a20551908$ cat prefix.txt
This is the assignment for introduction to information security.
This is the third lab assignmnet.
```

For a given prefix file 'prefix.txt', the following program generates two output files, out1.bin and out2.bin:

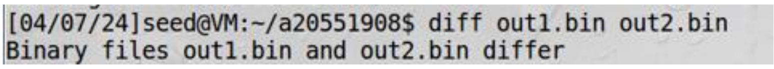
md5collgen -p prefix.txt -o out1.bin out2.bin



```
[04/07/24]seed@VM:~/a20551908$ 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: 344ccf2e85855d064bd58139102fa669

Generating first block: ..........
Generating second block: W......................
Running time: 4.96938 s
```

Using the command, we have verified if the output files are distinct or not.

diff out1.bin out2.bin

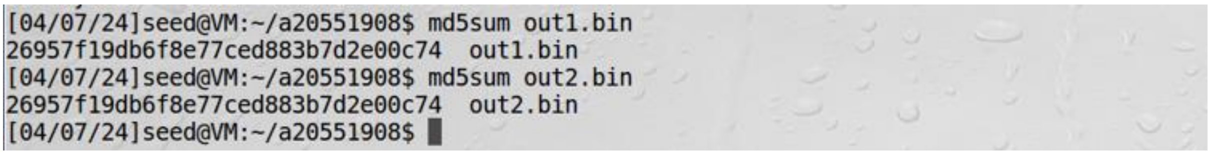


```
[04/07/24]seed@VM:~/a20551908$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
```

We have further verified each output file's MD5 hash using the md5sum command.

md5sum out1.bin

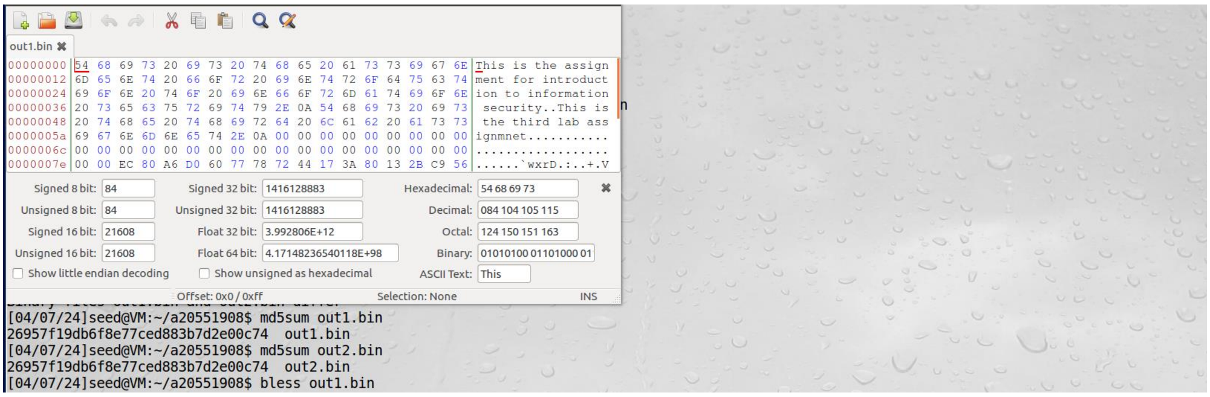
md5sum out2.bin



```
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin
26957f19db6f8e77ced883b7d2e00c74  out1.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out2.bin
26957f19db6f8e77ced883b7d2e00c74  out2.bin
[04/07/24]seed@VM:~/a20551908$ █
```

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

Zeros will be padded into our prefix file if its length is not a multiple of 64. This is a result of the file being processed by MD5 in 64-byte per block.



out1.bin

```
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E This is the assign
00000012 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 ment for introduct
00000024 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E ion to information
00000036 20 73 65 63 75 72 69 74 79 2E 0A 54 68 69 73 20 69 73  security..This is
00000048 20 74 68 65 20 74 68 69 72 64 20 6C 61 62 20 61 73 73  the third lab ass
0000005a 69 67 6E 6D 6E 65 74 2E 0A 00 00 00 00 00 00 00 00 00 ignmnet...........
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0000007e 00 00 EC 80 A6 D0 60 77 78 72 44 17 3A 80 13 2B C9 56 ......`wxrD.:..+.V
```

Signed 8 bit: 84 | Signed 32 bit: 1416128883 | Hexadecimal: 54 68 69 73

Unsigned 8 bit: 84 | Unsigned 32 bit: 1416128883 | Decimal: 084 104 105 115

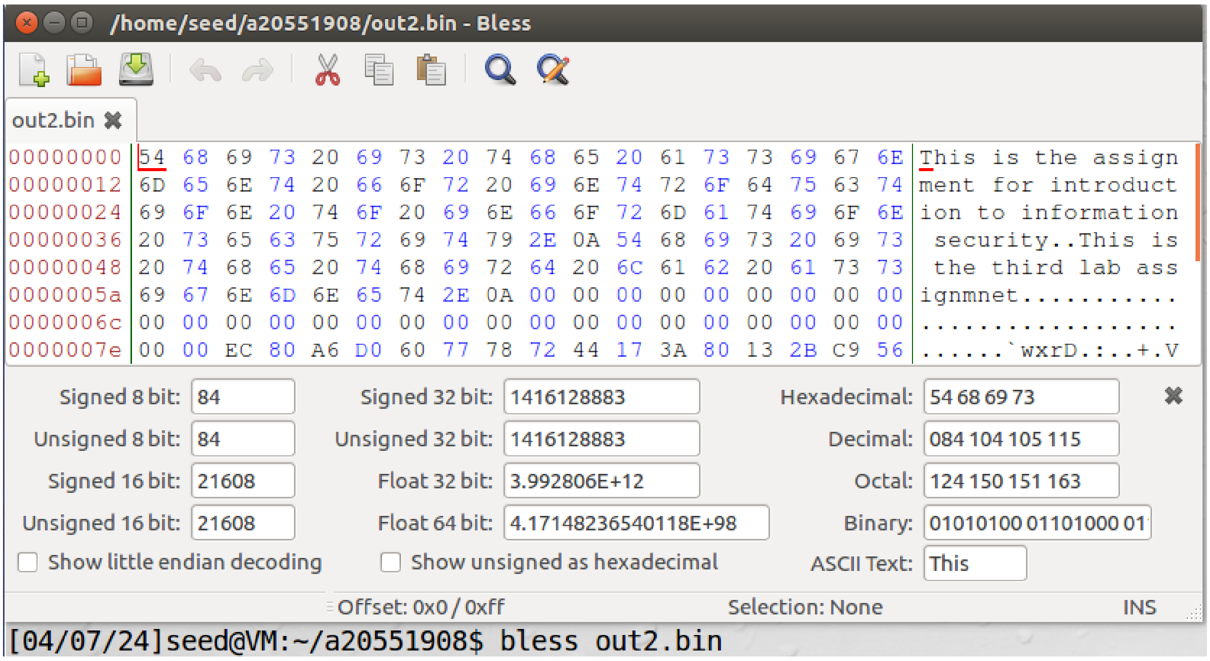
Signed 16 bit: 21608 | Float 32 bit: 3.992806E+12 | Octal: 124 150 151 163

Unsigned 16 bit: 21608 | Float 64 bit: 4.17148236540118E+98 | Binary: 01010100 01101000 01

□ Show little endian decoding □ Show unsigned as hexadecimal ASCII Text: This

Offset: 0x0 / 0xff Selection: None INS

```
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin
26957f19db6f8e77ced883b7d2e00c74  out1.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out2.bin
26957f19db6f8e77ced883b7d2e00c74  out2.bin
[04/07/24]seed@VM:~/a20551908$ bless out1.bin
```



/home/seed/a20551908/out2.bin - Bless

out2.bin

```
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E This is the assign
00000012 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 ment for introduct
00000024 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E ion to information
00000036 20 73 65 63 75 72 69 74 79 2E 0A 54 68 69 73 20 69 73  security..This is
00000048 20 74 68 65 20 74 68 69 72 64 20 6C 61 62 20 61 73 73  the third lab ass
0000005a 69 67 6E 6D 6E 65 74 2E 0A 00 00 00 00 00 00 00 00 00 ignmnet...........
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0000007e 00 00 EC 80 A6 D0 60 77 78 72 44 17 3A 80 13 2B C9 56 ......`wxrD.:..+.V
```

Signed 8 bit: 84 | Signed 32 bit: 1416128883 | Hexadecimal: 54 68 69 73

Unsigned 8 bit: 84 | Unsigned 32 bit: 1416128883 | Decimal: 084 104 105 115

Signed 16 bit: 21608 | Float 32 bit: 3.992806E+12 | Octal: 124 150 151 163

Unsigned 16 bit: 21608 | Float 64 bit: 4.17148236540118E+98 | Binary: 01010100 01101000 01

□ Show little endian decoding □ Show unsigned as hexadecimal ASCII Text: This

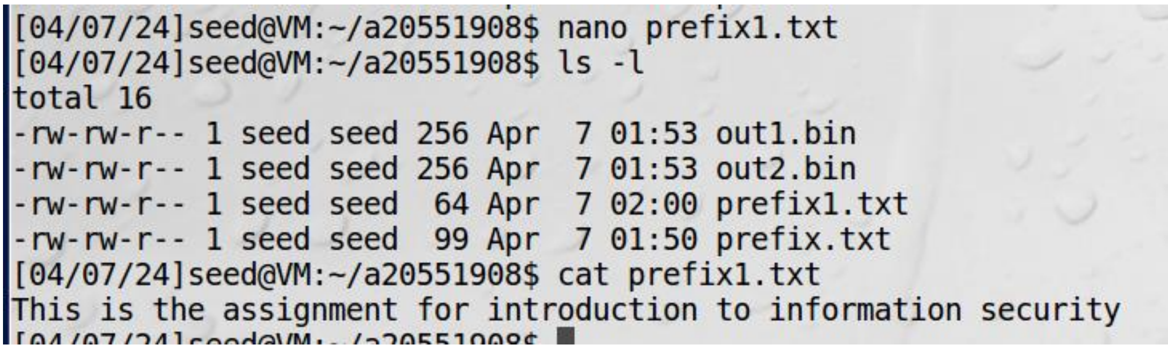
Offset: 0x0 / 0xff Selection: None INS

```
[04/07/24]seed@VM:~/a20551908$ bless out2.bin
```

The screenshots demonstrate that zeros were added to the file since its size was not a multiple of 64.

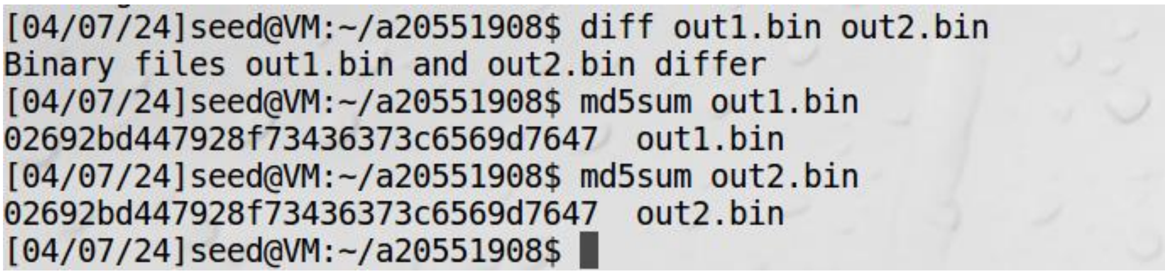
Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.

Creating a prefix file ‘prefix1.txt’ whose length is exactly 64



```
[04/07/24]seed@VM:~/a20551908$ nano prefix1.txt
[04/07/24]seed@VM:~/a20551908$ ls -l
total 16
-rw-rw-r-- 1 seed seed 256 Apr  7 01:53 out1.bin
-rw-rw-r-- 1 seed seed 256 Apr  7 01:53 out2.bin
-rw-rw-r-- 1 seed seed  64 Apr  7 02:00 prefix1.txt
-rw-rw-r-- 1 seed seed  99 Apr  7 01:50 prefix.txt
[04/07/24]seed@VM:~/a20551908$ cat prefix1.txt
This is the assignment for introduction to information security
[04/07/24]seed@VM:~/a20551908$
```

After running the ‘md5collgen -p prefix.txt -o out1.bin out2.bin’ command, the following commands are run.

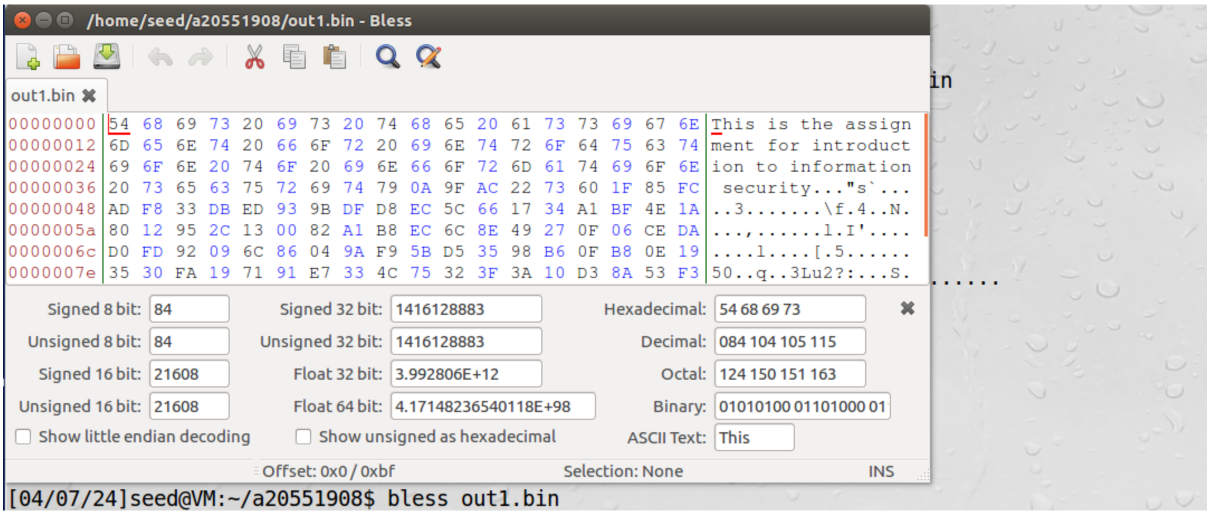


```
[04/07/24]seed@VM:~/a20551908$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin
02692bd447928f73436373c6569d7647  out1.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out2.bin
02692bd447928f73436373c6569d7647  out2.bin
[04/07/24]seed@VM:~/a20551908$ 
```

To observe if 0’s are padded or not we view the ‘out1.bin’ and ‘out2.bin’ using the following command

bless out1.bin

bless out2.bin



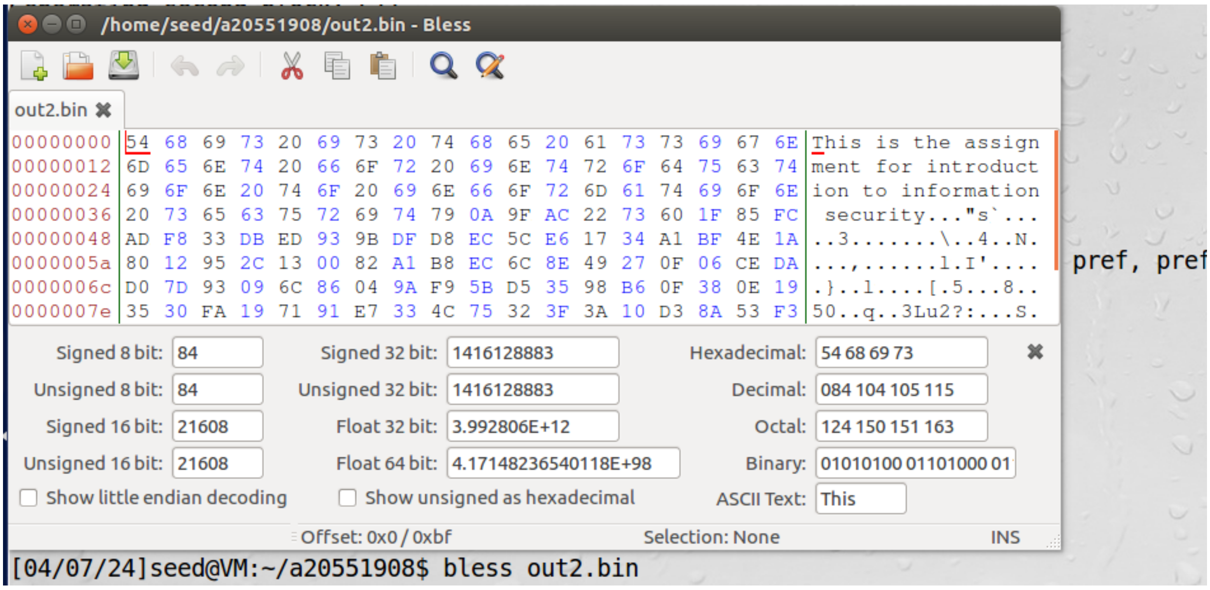
```
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E This is the assign
00000012 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 ment for introduct
00000024 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E ion to information
00000036 20 73 65 63 75 72 69 74 79 0A 9F AC 22 73 60 1F 85 FC  security..."s`...
00000048 AD F8 33 DB ED 93 9B DF D8 EC 5C 66 17 34 A1 BF 4E 1A ..3.......\f.4..N.
0000005a 80 12 95 2C 13 00 82 A1 B8 EC 6C 8E 49 27 0F 06 CE DA ...,......l.I'....
0000006c D0 FD 92 09 6C 86 04 9A F9 5B D5 35 98 B6 0F B8 0E 19 ....l....[.5......
0000007e 35 30 FA 19 71 91 E7 33 4C 75 32 3F 3A 10 D3 8A 53 F3 50..q..3Lu2?:...S.
```

Signed 8 bit: 84
Unsigned 8 bit: 84
Signed 16 bit: 21608
Unsigned 16 bit: 21608
Signed 32 bit: 1416128883
Unsigned 32 bit: 1416128883
Float 32 bit: 3.992806E+12
Float 64 bit: 4.17148236540118E+98
Hexadecimal: 54 68 69 73
Decimal: 084 104 105 115
Octal: 124 150 151 163
Binary: 01010100 01101000 01
ASCII Text: This

☐ Show little endian decoding ☐ Show unsigned as hexadecimal

Offset: 0x0 / 0xbf Selection: None INS

```
[04/07/24]seed@VM:~/a20551908$ bless out1.bin
```



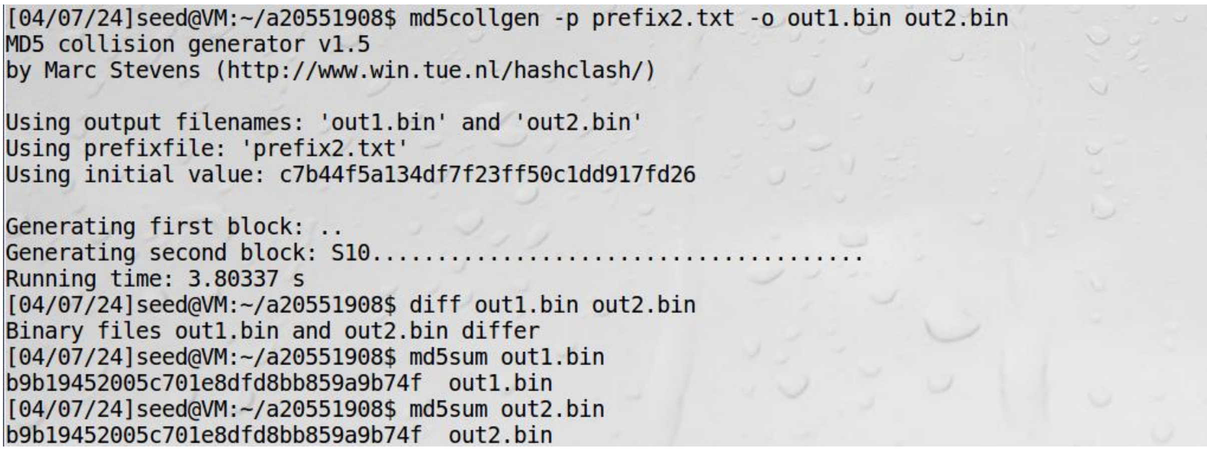
```
/home/seed/a20551908/out2.bin - Bless
out2.bin
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E This is the assign
00000012 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 ment for introduct
00000024 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E ion to information
00000036 20 73 65 63 75 72 69 74 79 0A 9F AC 22 73 60 1F 85 FC  security..."s`...
00000048 AD F8 33 DB ED 93 9B DF D8 EC 5C E6 17 34 A1 BF 4E 1A ..3.......\..4..N.
0000005a 80 12 95 2C 13 00 82 A1 B8 EC 6C 8E 49 27 0F 06 CE DA ...,......l.I'....
0000006c D0 7D 93 09 6C 86 04 9A F9 5B D5 35 98 B6 0F 38 0E 19 .}..l....[.5...8..
0000007e 35 30 FA 19 71 91 E7 33 4C 75 32 3F 3A 10 D3 8A 53 F3 50..q..3Lu2?:...S.

  Signed 8 bit: 84       Signed 32 bit: 1416128883              Hexadecimal: 54 68 69 73
Unsigned 8 bit: 84     Unsigned 32 bit: 1416128883                  Decimal: 084 104 105 115
 Signed 16 bit: 21608      Float 32 bit: 3.992806E+12                  Octal: 124 150 151 163
Unsigned 16 bit: 21608     Float 64 bit: 4.17148236540118E+98          Binary: 01010100 01101000 01
☐ Show little endian decoding   ☐ Show unsigned as hexadecimal     ASCII Text: This
                  Offset: 0x0 / 0xbf              Selection: None              INS
[04/07/24]seed@VM:~/a20551908$ bless out2.bin
```

`pref, pref`

We observe that no zero's are padded as the file is of exactly 64 bytes.

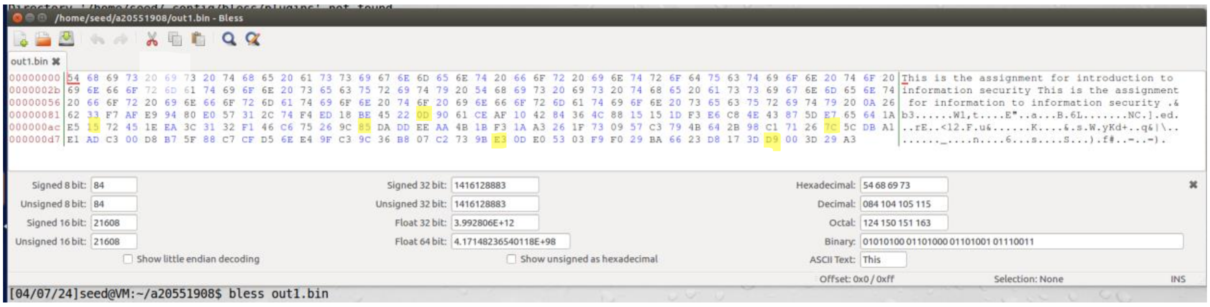
Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different.

Created a new file named 'prefix2.txt' of 128 bytes. The following commands are run over the newly created file.

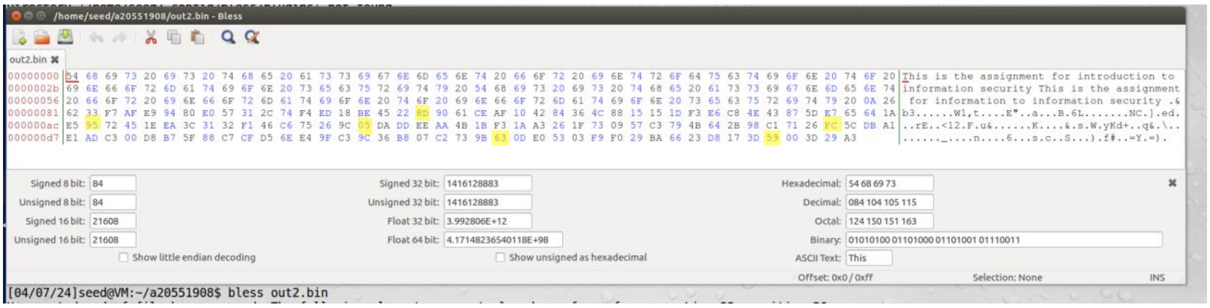


```
[04/07/24]seed@VM:~/a20551908$ md5collgen -p prefix2.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix2.txt'
Using initial value: c7b44f5a134df7f23ff50c1dd917fd26

Generating first block: ..
Generating second block: S10..................................
Running time: 3.80337 s
[04/07/24]seed@VM:~/a20551908$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin
b9b19452005c701e8dfd8bb859a9b74f  out1.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out2.bin
b9b19452005c701e8dfd8bb859a9b74f  out2.bin
```

In order to observe the difference in bytes we use 'bless'.



```
Directory '/home/seed/.config/bless/plugins' not found
/home/seed/a20551908/out1.bin - Bless
out1.bin
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 69 6F 6E 20 74 6F 20 This is the assignment for introduction to 
0000002b 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 73 65 63 75 72 69 74 79 20 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E 6D 65 6E 74 information security This is the assignment
00000056 20 66 6F 72 20 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 73 65 63 75 72 69 74 79 20 0A 26  for information to information security .&
00000081 62 33 F7 AF E9 94 80 E0 57 31 2C 74 F4 ED 18 BE 45 22 0D 90 61 CE AF 10 42 84 36 4C 88 15 15 1D F3 E6 C8 4E 43 87 5D E7 65 64 1A b3......W1,t....E"..a...B.6L.......NC.].ed.
000000ac E5 15 72 45 1E EA 3C 31 32 F1 46 C6 75 26 9C 85 DA DD EE AA 4B 1B F3 1A A3 26 1F 73 09 57 C3 79 4B 64 2B 98 C1 71 26 7C 5C DB A1 ..rE..<12.F.u&......K....&.s.W.yKd+..q&|\..
000000d7 E1 AD C3 00 D8 B7 5F 88 C7 CF D5 6E E4 9F C3 9C 36 B8 07 C2 73 9B E3 0D E0 53 03 F9 F0 29 BA 66 23 D8 17 3D D9 00 3D 29 A3       ......_....n....6...s....S...).f#..-..-).

  Signed 8 bit: 84        Signed 32 bit: 1416128883            Hexadecimal: 54 68 69 73
Unsigned 8 bit: 84      Unsigned 32 bit: 1416128883                Decimal: 084 104 105 115
 Signed 16 bit: 21608       Float 32 bit: 3.992806E+12                Octal: 124 150 151 163
Unsigned 16 bit: 21608      Float 64 bit: 4.17148236540118E+98        Binary: 01010100 01101000 01101001 01110011
       ☐ Show little endian decoding     ☐ Show unsigned as hexadecimal     ASCII Text: This
                                                  Offset: 0x0 / 0xff         Selection: None         INS
[04/07/24]seed@VM:~/a20551908$ bless out1.bin
```



/home/seed/a20551908/out2.bin - Bless

out2.bin

```
00000000 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E 6D 65 6E 74 20 66 6F 72 20 69 6E 74 72 6F 64 75 63 74 69 6F 6E 20 74 6F 20 This is the assignment for introduction to 
0000002b 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 73 65 63 75 72 69 74 79 20 54 68 69 73 20 69 73 20 74 68 65 20 61 73 73 69 67 6E 6D 65 6E 74 information security This is the assignment
00000056 20 66 6F 72 20 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 74 6F 20 69 6E 66 6F 72 6D 61 74 69 6F 6E 20 73 65 63 75 72 69 74 79 20 0A 26  for information to information security .&
00000081 62 33 F7 AF E9 94 80 E0 57 31 2C 74 F4 ED 18 BE 45 22 8D 90 61 CE AF 10 42 84 36 4C 88 15 15 1D F3 E6 C8 4E 43 87 5D E7 65 64 1A b3......W1,t....E"..a...B.6L.......NC.].ed.
000000ac E5 95 72 45 1E EA 3C 31 32 F1 46 C6 75 26 9C 05 DA DD EE AA 4B 1B F3 1A A3 26 1F 73 09 57 C3 79 4B 64 2B 98 C1 71 26 FC 5C DB A1 ..rE..<12.F.u&......K....&.s.W.yKd+..q&.\..
000000d7 E1 AD C3 00 D8 B7 5F 88 C7 CF D5 6E E4 9F C3 9C 36 B8 07 C2 73 9B 63 0D E0 53 03 F9 F0 29 BA 66 23 D8 17 3D 59 00 3D 29 A3       ......_....n....6...s.c..S...).f#..=Y.=).
```

Signed 8 bit: 84
Unsigned 8 bit: 84
Signed 16 bit: 21608
Unsigned 16 bit: 21608
□ Show little endian decoding
Signed 32 bit: 1416128883
Unsigned 32 bit: 1416128883
Float 32 bit: 3.992806E+12
Float 64 bit: 4.17148236540118E+98
□ Show unsigned as hexadecimal
Hexadecimal: 54 68 69 73
Decimal: 084 104 105 115
Octal: 124 150 151 163
Binary: 01010100 01101000 01101001 01110011
ASCII Text: This
Offset: 0x0 / 0xff
Selection: None
INS

```
[04/07/24]seed@VM:~/a20551908$ bless out2.bin
```

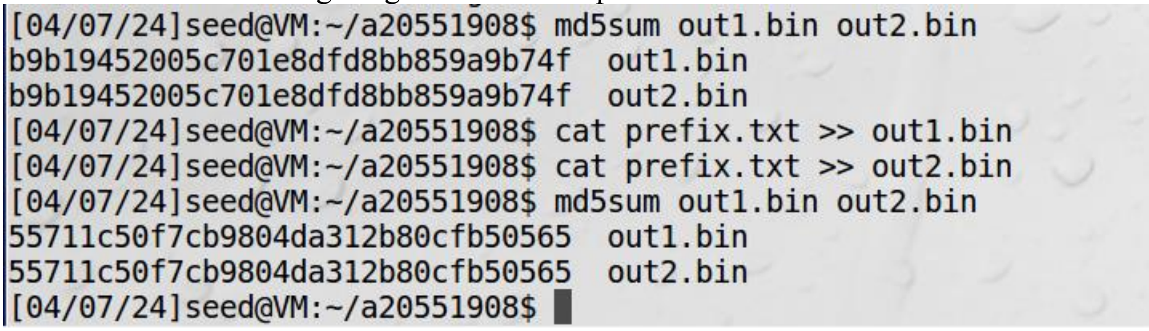
No, the data generated by md5collgen are not completely different, **only few bytes differ**. The bytes that differ are as follows:

out1.bin	out2.bin
0D	8D
15	95
85	05
7C	FC
E3	63
D9	59

2.2 Task 2: Understanding MD5's Property

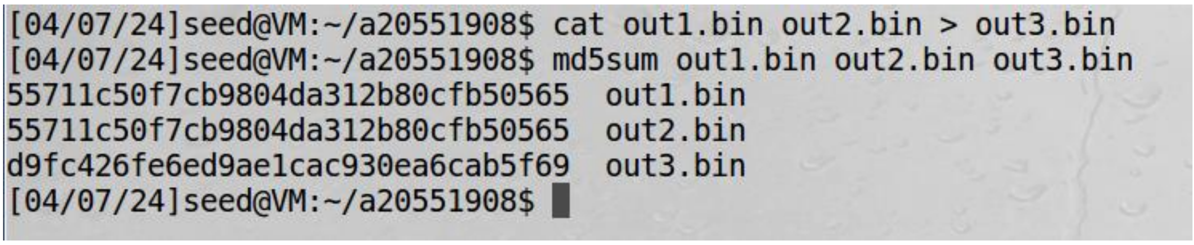
The file prefix.txt will is used to verify if the MD5 hashes match. The produced files out1.bin and out2.bin will then have a random text appended to the end, and their MD5 hashes will be checked once again.

We will run the md5collgen again on the file 'prefix.txt'



```
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin out2.bin
b9b19452005c701e8dfd8bb859a9b74f  out1.bin
b9b19452005c701e8dfd8bb859a9b74f  out2.bin
[04/07/24]seed@VM:~/a20551908$ cat prefix.txt >> out1.bin
[04/07/24]seed@VM:~/a20551908$ cat prefix.txt >> out2.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin out2.bin
55711c50f7cb9804da312b80cfb50565  out1.bin
55711c50f7cb9804da312b80cfb50565  out2.bin
[04/07/24]seed@VM:~/a20551908$ 
```

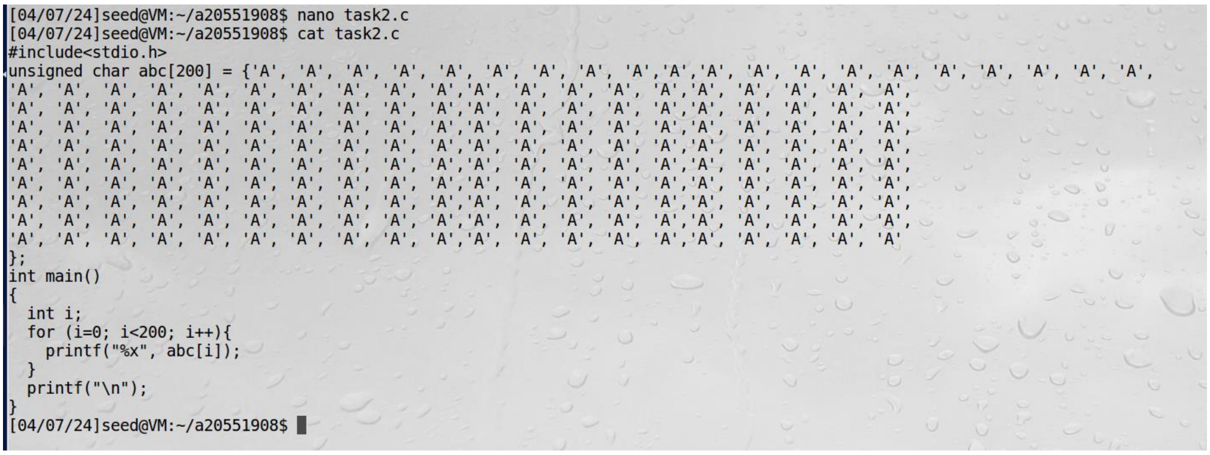
We observe that, the md5 values generated are the same for both 'out1.bin' and 'out2.bin'. Then we append and calculate the values md5 values. The freshly created MD5 hashes are different from the previously generated ones, yet they are the same, as we can see. This is due to the length extension vulnerability of the MD5 hash method. Given that both files' MD5 hashes matched, it is reasonable to assume that the internal state of the system following algorithm execution was the same.



```
[04/07/24]seed@VM:~/a20551908$ cat out1.bin out2.bin > out3.bin
[04/07/24]seed@VM:~/a20551908$ md5sum out1.bin out2.bin out3.bin
55711c50f7cb9804da312b80cfb50565  out1.bin
55711c50f7cb9804da312b80cfb50565  out2.bin
d9fc426fe6ed9ae1cac930ea6cab5f69  out3.bin
[04/07/24]seed@VM:~/a20551908$ 
```

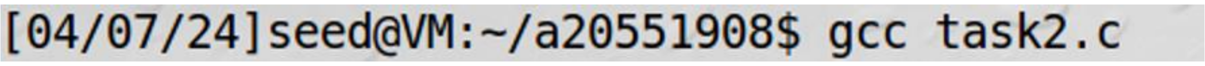
2.3 Task 3: Generating Two Executable Files with the Same MD5 Hash

The following is the code:



```
[04/07/24]seed@VM:~/a20551908$ nano task2.c
[04/07/24]seed@VM:~/a20551908$ cat task2.c
#include<stdio.h>
unsigned char abc[200] = {'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A','A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A'
};
int main()
{
  int i;
  for (i=0; i<200; i++){
    printf("%x", abc[i]);
  }
  printf("\n");
}
[04/07/24]seed@VM:~/a20551908$
```

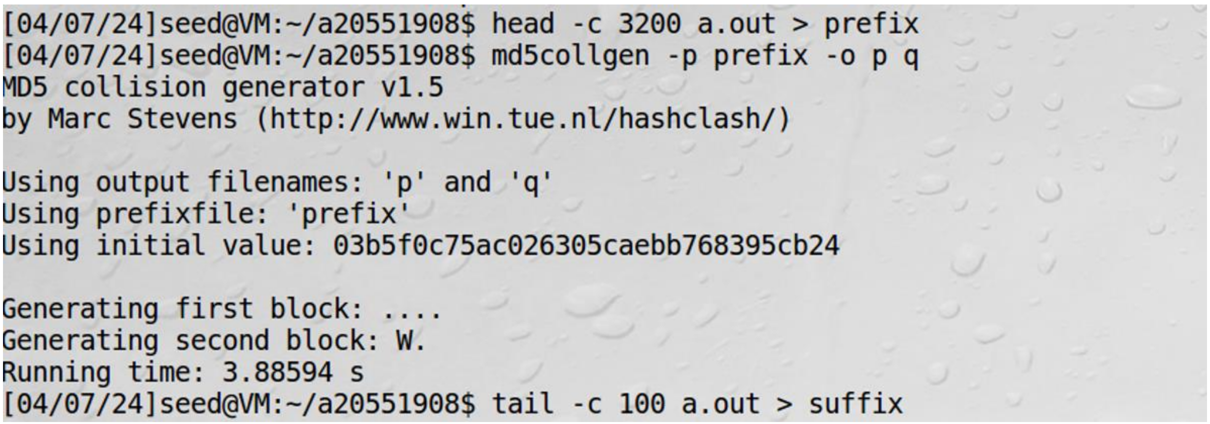
Inorder to execute this code we use ‘gcc task2.c’. This generates a file ‘a.out’.



```
[04/07/24]seed@VM:~/a20551908$ gcc task2.c
```

Divide file into 3 sections, using the following commands

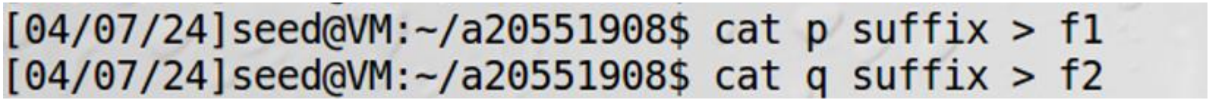
head -c 3200 a.out > prefix
md5collgen -p prefix -o p q
tail -c 100 a.out > suffix
tail -c +3300 a.out > suffix



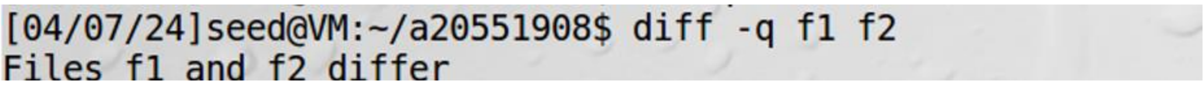
```
[04/07/24]seed@VM:~/a20551908$ head -c 3200 a.out > prefix
[04/07/24]seed@VM:~/a20551908$ md5collgen -p prefix -o p q
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'p' and 'q'
Using prefixfile: 'prefix'
Using initial value: 03b5f0c75ac026305caebb768395cb24

Generating first block: ....
Generating second block: W.
Running time: 3.88594 s
[04/07/24]seed@VM:~/a20551908$ tail -c 100 a.out > suffix
```

Now we will concatenate the suffix to the two individual files.

cat p suffix > f1
cat q suffix > f2



```
[04/07/24]seed@VM:~/a20551908$ cat p suffix > f1
[04/07/24]seed@VM:~/a20551908$ cat q suffix > f2
```



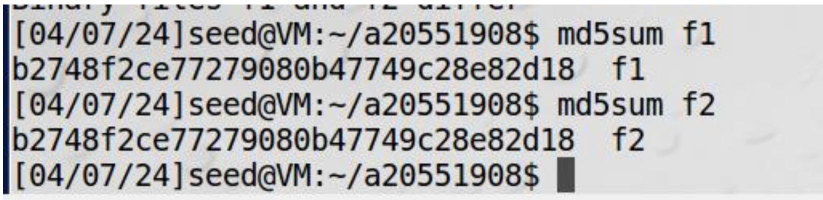
```
[04/07/24]seed@VM:~/a20551908$ diff -q f1 f2
Files f1 and f2 differ
```

We see that the 2 files f1 and f2 differ, now lets see their MD5 hashes we use the following commands

diff -q f1 f2

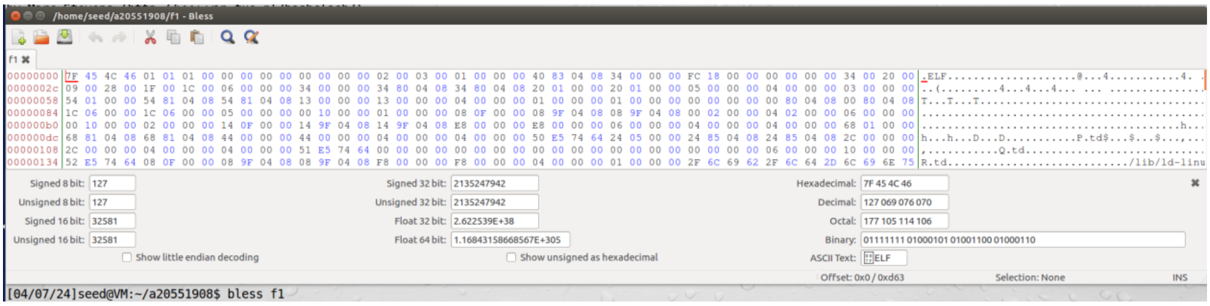
md5sum f1

md5sum f2



```
Binary files f1 and f2 differ
[04/07/24]seed@VM:~/a20551908$ md5sum f1
b2748f2ce77279080b47749c28e82d18  f1
[04/07/24]seed@VM:~/a20551908$ md5sum f2
b2748f2ce77279080b47749c28e82d18  f2
[04/07/24]seed@VM:~/a20551908$ 
```

We observe that their MD5 hash values are similar.



```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 00 00 00 00 34 00 20 00 .ELF....................@...4...........4. .
0000002c 09 00 28 00 1F 00 1C 00 06 00 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 ..(........4...4...4... ... ...............
00000058 54 01 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08 00 80 04 08 T...T...T...................................
00000084 1C 06 00 00 1C 06 00 00 05 00 00 00 00 10 00 00 01 00 00 00 08 0F 00 00 08 9F 04 08 08 9F 04 08 00 02 00 00 04 02 00 00 06 00 00 00 ............................................
000000b0 00 10 00 00 02 00 00 00 14 0F 00 00 14 9F 04 08 14 9F 04 08 E8 00 00 00 E8 00 00 00 06 00 00 00 04 00 00 00 04 00 00 00 68 01 00 00 ........................................h...
000000dc 68 81 04 08 68 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00 04 00 00 00 50 E5 74 64 24 05 00 00 24 85 04 08 24 85 04 08 2C 00 00 00 h...h...D...D...........P.td$...$...$...,...
00000108 2C 00 00 00 04 00 00 00 04 00 00 00 51 E5 74 64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 10 00 00 00 ,...........Q.td............................
00000134 52 E5 74 64 08 0F 00 00 08 9F 04 08 08 9F 04 08 F8 00 00 00 F8 00 00 00 04 00 00 00 01 00 00 00 2F 6C 69 62 2F 6C 64 2D 6C 69 6E 75 R.td............................/lib/ld-linu
```

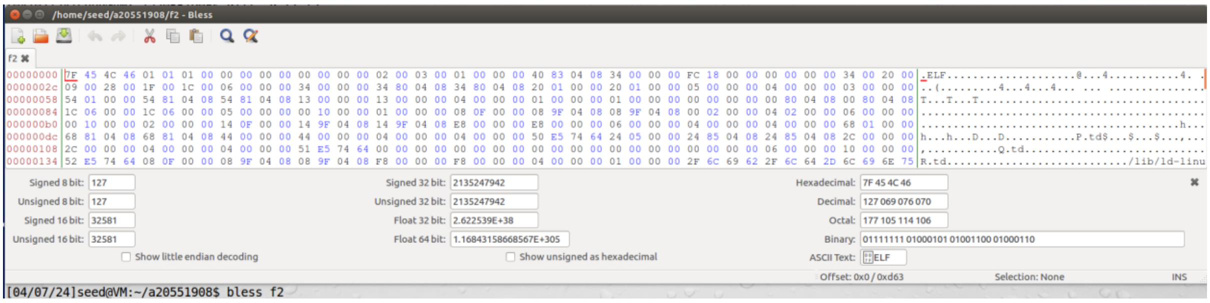
Signed 8 bit: 127
Unsigned 8 bit: 127
Signed 16 bit: 32581
Unsigned 16 bit: 32581
□ Show little endian decoding

Signed 32 bit: 2135247942
Unsigned 32 bit: 2135247942
Float 32 bit: 2.622539E+38
Float 64 bit: 1.16843158668567E+305
□ Show unsigned as hexadecimal

Hexadecimal: 7F 45 4C 46
Decimal: 127 069 076 070
Octal: 177 105 114 106
Binary: 01111111 01000101 01001100 01000110
ASCII Text: ELF

Offset: 0x0 / 0xd63 Selection: None INS

```
[04/07/24]seed@VM:~/a20551908$ bless f1
```



```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 00 00 00 00 34 00 20 00 .ELF....................@...4...........4. .
0000002c 09 00 28 00 1F 00 1C 00 06 00 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 ..(........4...4...4... ... ...............
00000058 54 01 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08 00 80 04 08 T...T...T...................................
00000084 1C 06 00 00 1C 06 00 00 05 00 00 00 00 10 00 00 01 00 00 00 08 0F 00 00 08 9F 04 08 08 9F 04 08 00 02 00 00 04 02 00 00 06 00 00 00 ............................................
000000b0 00 10 00 00 02 00 00 00 14 0F 00 00 14 9F 04 08 14 9F 04 08 E8 00 00 00 E8 00 00 00 06 00 00 00 04 00 00 00 04 00 00 00 68 01 00 00 ........................................h...
000000dc 68 81 04 08 68 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00 04 00 00 00 50 E5 74 64 24 05 00 00 24 85 04 08 24 85 04 08 2C 00 00 00 h...h...D...D...........P.td$...$...$...,...
00000108 2C 00 00 00 04 00 00 00 04 00 00 00 51 E5 74 64 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 10 00 00 00 ,...........Q.td............................
00000134 52 E5 74 64 08 0F 00 00 08 9F 04 08 08 9F 04 08 F8 00 00 00 F8 00 00 00 04 00 00 00 01 00 00 00 2F 6C 69 62 2F 6C 64 2D 6C 69 6E 75 R.td............................/lib/ld-linu
```

Signed 8 bit: 127
Unsigned 8 bit: 127
Signed 16 bit: 32581
Unsigned 16 bit: 32581
□ Show little endian decoding

Signed 32 bit: 2135247942
Unsigned 32 bit: 2135247942
Float 32 bit: 2.622539E+38
Float 64 bit: 1.16843158668567E+305
□ Show unsigned as hexadecimal

Hexadecimal: 7F 45 4C 46
Decimal: 127 069 076 070
Octal: 177 105 114 106
Binary: 01111111 01000101 01001100 01000110
ASCII Text: ELF

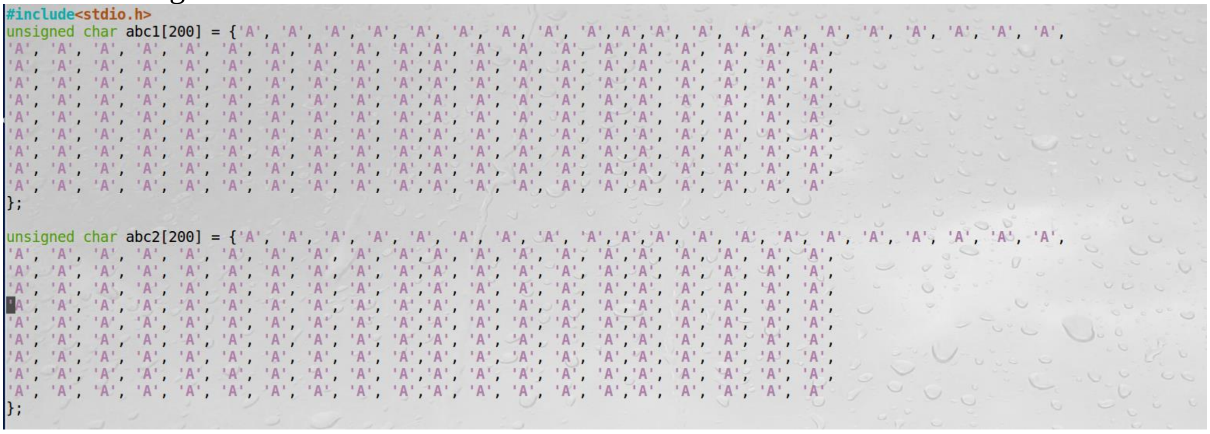
Offset: 0x0 / 0xd63 Selection: None INS

```
[04/07/24]seed@VM:~/a20551908$ bless f2
```

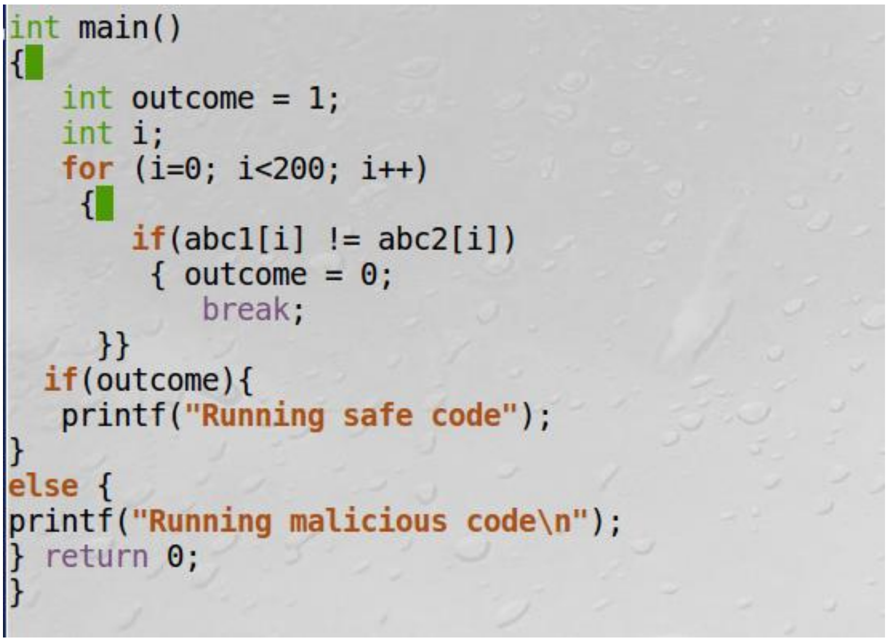
The above are the binaries of ‘f1’ and ‘f2’.

2.4 Task 4: Making the Two Programs Behave Differently

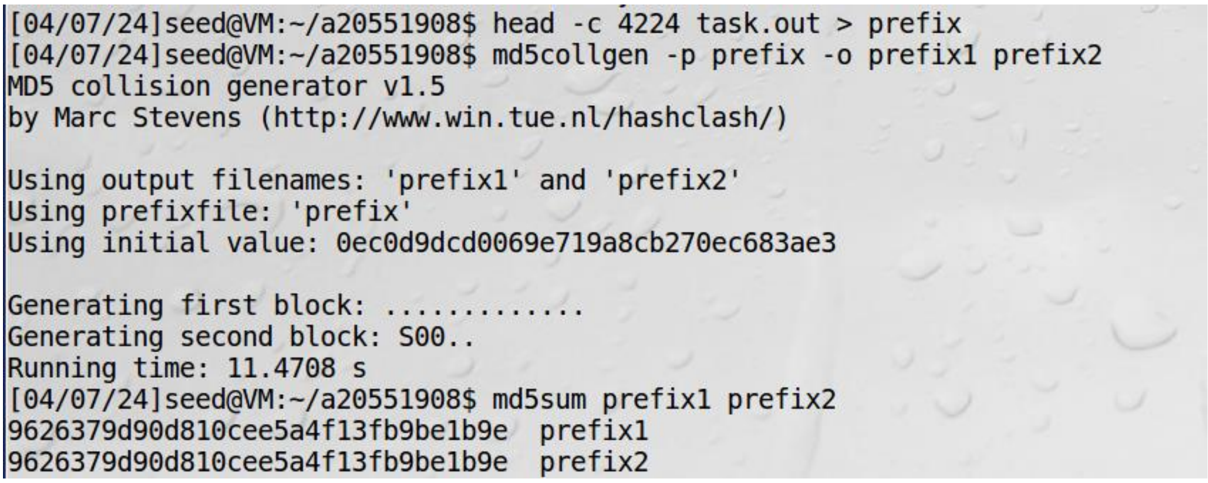
For this, we develop two distinct programs. One program will consistently carry out benign instructions, while the other program will carry out malicious instructions. We manage to make these two programs share the same MD5 hash value.
The following is the code.



```
#include<stdio.h>
unsigned char abc1[200] = {'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A','A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A'
};

unsigned char abc2[200] = {'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A','A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A'
};
```



```
int main()
{
   int outcome = 1;
   int i;
   for (i=0; i<200; i++)
    {
      if(abc1[i] != abc2[i])
       { outcome = 0;
           break;
     }}
  if(outcome){
   printf("Running safe code");
}
else {
printf("Running malicious code\n");
} return 0;
}
```



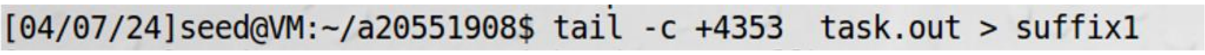
```
[04/07/24]seed@VM:~/a20551908$ head -c 4224 task.out > prefix
[04/07/24]seed@VM:~/a20551908$ md5collgen -p prefix -o prefix1 prefix2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'prefix1' and 'prefix2'
Using prefixfile: 'prefix'
Using initial value: 0ec0d9dcd0069e719a8cb270ec683ae3

Generating first block: ............
Generating second block: S00..
Running time: 11.4708 s
[04/07/24]seed@VM:~/a20551908$ md5sum prefix1 prefix2
9626379d90d810cee5a4f13fb9be1b9e  prefix1
9626379d90d810cee5a4f13fb9be1b9e  prefix2
```

After we generate 2 files using “prefix”, add all bytes after 4352 in task4.out to suffix1.

head -c 4224 task.out > prefix

md5collgen -p prefix -o prefix1 prefix2

tail -c +4353 task.out > suffix1



```
[04/07/24]seed@VM:~/a20551908$ tail -c +4353  task.out > suffix1
```

We will add first 8 bytes of suffix1 to prefix1 and prefix2 and generate prefix1_arr1 and prefix2_arr1.

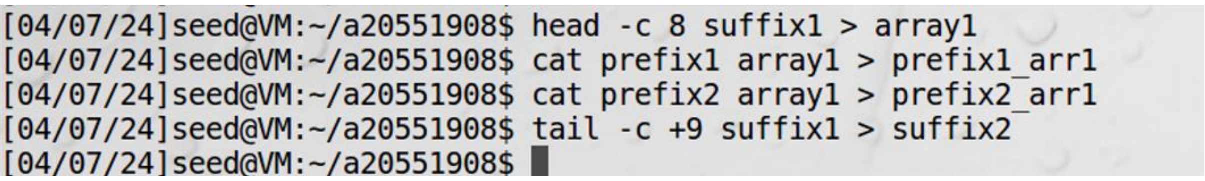
Create suffix2 file which contains all bytes after the 8th byte in suffix1.

head -c 8 suffix1> arr1

cat p1 arr1 > p1_arr1

cat q1 arr1 > q1_arr1

tail -c +9 suffix1 > suffix2



```
[04/07/24]seed@VM:~/a20551908$ head -c 8 suffix1 > array1
[04/07/24]seed@VM:~/a20551908$ cat prefix1 array1 > prefix1_arr1
[04/07/24]seed@VM:~/a20551908$ cat prefix2 array1 > prefix2_arr1
[04/07/24]seed@VM:~/a20551908$ tail -c +9 suffix1 > suffix2
[04/07/24]seed@VM:~/a20551908$ 
```

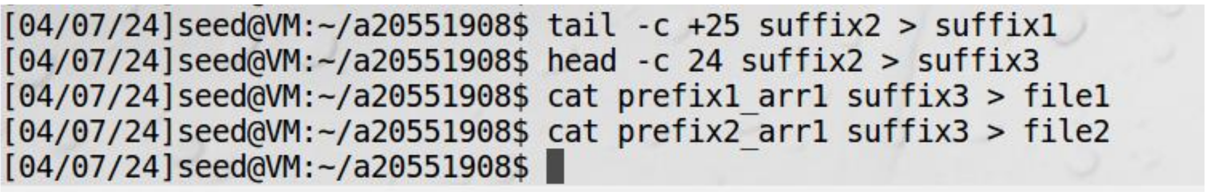
We add the bytes between the ending of the first array and the beginning of the second array to create a file suffix3.In order to generate file1 and file2, we save the bytes starting with the second array in suffix to suffix1 and add them to prefix1_arr1 and prefix2_arr1.

tail -c +25 suffix2 > suffix1

head -c 24 suffix2 > suffix3

cat prefix1_arr1 suffix3> file1

cat prefix2_arr1 suffix3> file2



```
[04/07/24]seed@VM:~/a20551908$ tail -c +25 suffix2 > suffix1
[04/07/24]seed@VM:~/a20551908$ head -c 24 suffix2 > suffix3
[04/07/24]seed@VM:~/a20551908$ cat prefix1_arr1 suffix3 > file1
[04/07/24]seed@VM:~/a20551908$ cat prefix2_arr1 suffix3 > file2
[04/07/24]seed@VM:~/a20551908$ 
```

If one executable outputs "run safe code" while the other outputs "run malicious code," then the attack is successful. The abc2 arraycontent must match one of the created arrays in order to do this. Thus, we append the bytes in suffix1 to suffix2 after the second array.

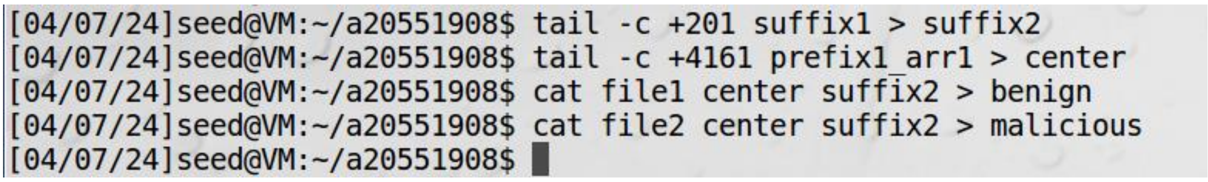
Next, we transfer the initial array from prefix1_arr1 to the center section. To create executable files that are malicious and benign, the center file can be concatenated with file1 and file2 together with suffix 2.

tail -c +201 suffix1 > suffix2

tail -c +4161 prefix1_arr1> center

cat file1 central suffix2 > benign

cat file2 central suffix2 > malicious



```
[04/07/24]seed@VM:~/a20551908$ tail -c +201 suffix1 > suffix2
[04/07/24]seed@VM:~/a20551908$ tail -c +4161 prefix1_arr1 > center
[04/07/24]seed@VM:~/a20551908$ cat file1 center suffix2 > benign
[04/07/24]seed@VM:~/a20551908$ cat file2 center suffix2 > malicious
[04/07/24]seed@VM:~/a20551908$ 
```

Then we execute the following commands

md5sum benign

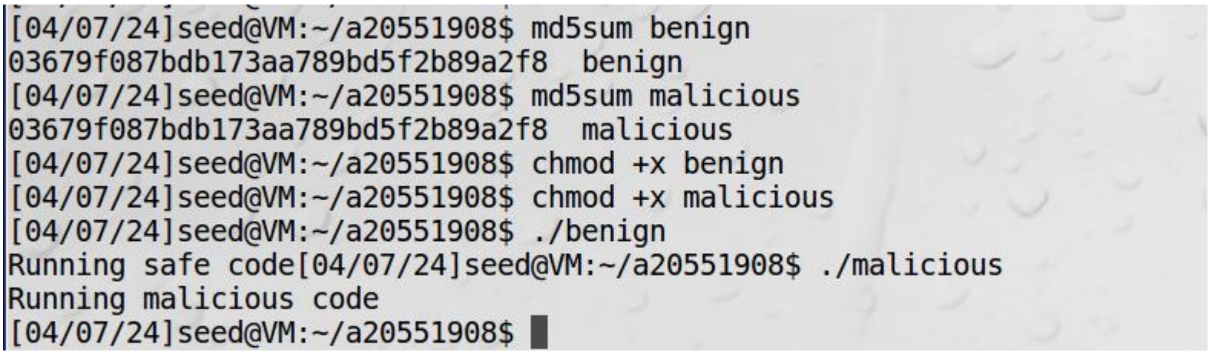
md5sum malicious

chmod +x benign

chmod +x malicious

./benign

./malicious



```
[04/07/24]seed@VM:~/a20551908$ md5sum benign
03679f087bdb173aa789bd5f2b89a2f8  benign
[04/07/24]seed@VM:~/a20551908$ md5sum malicious
03679f087bdb173aa789bd5f2b89a2f8  malicious
[04/07/24]seed@VM:~/a20551908$ chmod +x benign
[04/07/24]seed@VM:~/a20551908$ chmod +x malicious
[04/07/24]seed@VM:~/a20551908$ ./benign
Running safe code[04/07/24]seed@VM:~/a20551908$ ./malicious
Running malicious code
[04/07/24]seed@VM:~/a20551908$ █
```

We see that upon execution benign runs safe code and malicious runs malicious code, upon observing we can see that both generate the same md5 hash which implies, MD5 collision attack is achieved.