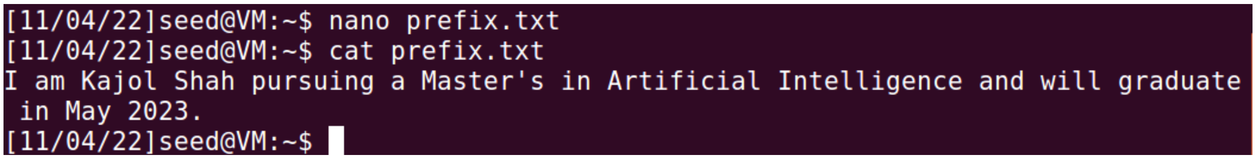
Kajol Tanesh Shah
A20496724

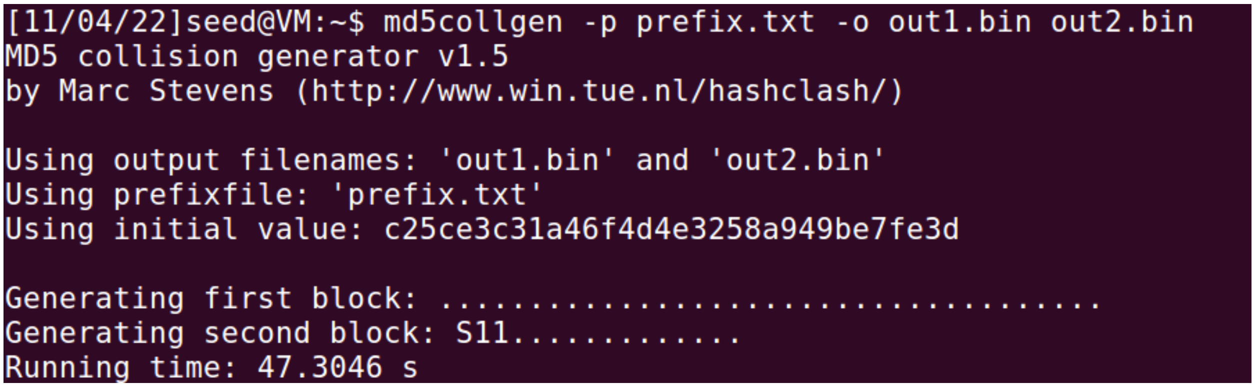
Introduction to Information Security - CS 458 - Fall 2022
Lab 3 - MD5 Collision Attack Lab 1

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

In this task, we have generated two different files with the same MD5 hash values. The beginning parts of these two files are the same, i.e., they share the same prefix. We have achieved this using the md5collgen program.
The following command generated two output files, out1.bin and out2.bin , for a given a prefix file prefix.txt :
$ md5collgen -p prefix.txt -o out1.bin out2.bin

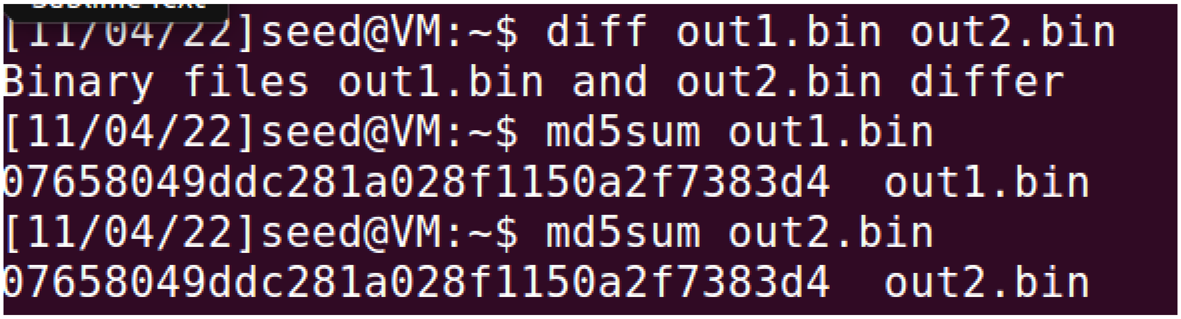


```
[11/04/22]seed@VM:~$ nano prefix.txt
[11/04/22]seed@VM:~$ cat prefix.txt
I am Kajol Shah pursuing a Master's in Artificial Intelligence and will graduate
 in May 2023.
[11/04/22]seed@VM:~$ 
```



```
[11/04/22]seed@VM:~$ 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: c25ce3c31a46f4d4e3258a949be7fe3d

Generating first block: ....................................
Generating second block: S11............
Running time: 47.3046 s
```

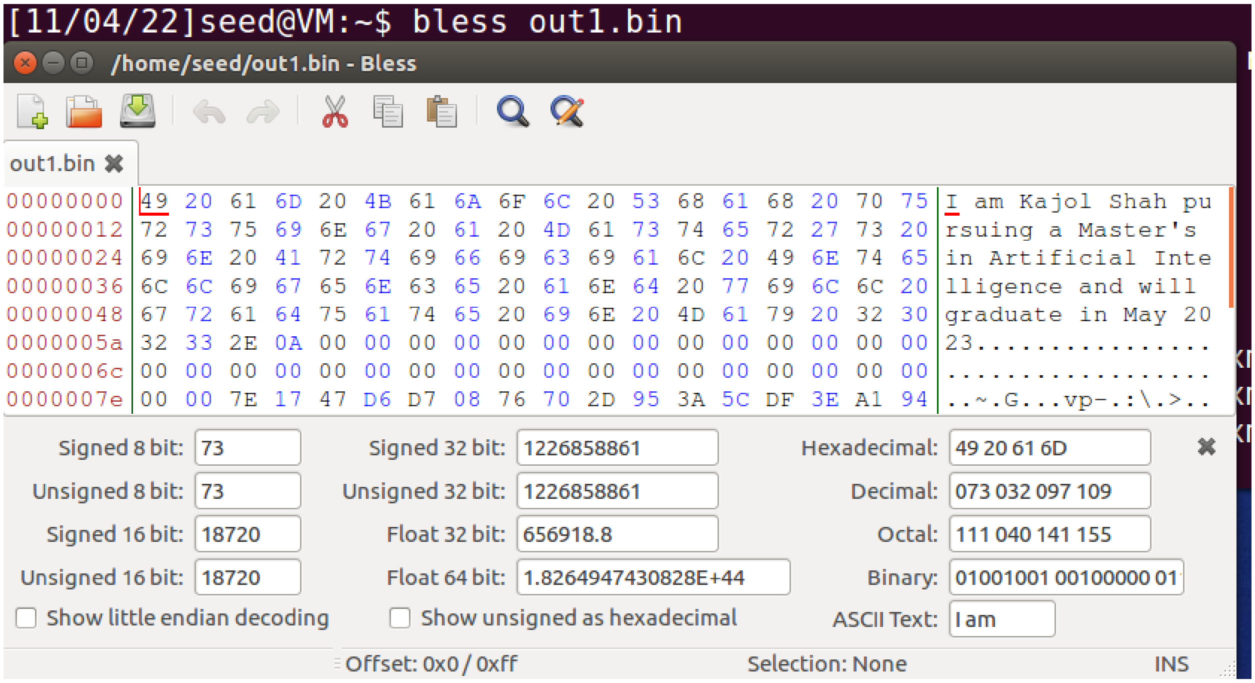
We have checked whether the output files are distinct or not using the diff command. We have also used the md5sum command to check the MD5 hash of each output file.



```
[11/04/22]seed@VM:~$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[11/04/22]seed@VM:~$ md5sum out1.bin
07658049ddc281a028f1150a2f7383d4  out1.bin
[11/04/22]seed@VM:~$ md5sum out2.bin
07658049ddc281a028f1150a2f7383d4  out2.bin
```

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

Ans. If the length of our prefix file is not a multiple of 64, zeros wil be padded to the file. This is because MD5 processes the file in blocks of size 64 bytes. From the below screenshot, we can see that as the size of the file was not a multiple of 64, zeros were padded to the file



```
[11/04/22]seed@VM:~$ bless out1.bin
```

/home/seed/out1.bin - Bless

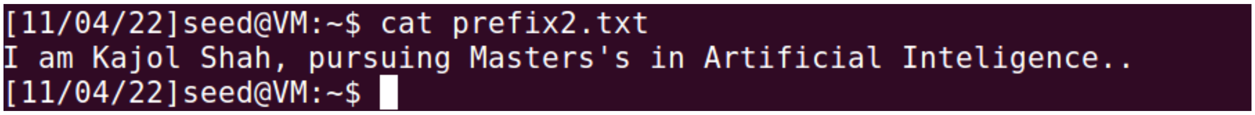
```
00000000 49 20 61 6D 20 4B 61 6A 6F 6C 20 53 68 61 68 20 70 75 I am Kajol Shah pu
00000012 72 73 75 69 6E 67 20 61 20 4D 61 73 74 65 72 27 73 20 rsuing a Master's
00000024 69 6E 20 41 72 74 69 66 69 63 69 61 6C 20 49 6E 74 65 in Artificial Inte
00000036 6C 6C 69 67 65 6E 63 65 20 61 6E 64 20 77 69 6C 6C 20 lligence and will
00000048 67 72 61 64 75 61 74 65 20 69 6E 20 4D 61 79 20 32 30 graduate in May 20
0000005a 32 33 2E 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 23................
0000006c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..................
0000007e 00 00 7E 17 47 D6 D7 08 76 70 2D 95 3A 5C DF 3E A1 94 ..~.G...vp-.:\.>..
```

Signed 8 bit: 73 | Signed 32 bit: 1226858861 | Hexadecimal: 49 20 61 6D
Unsigned 8 bit: 73 | Unsigned 32 bit: 1226858861 | Decimal: 073 032 097 109
Signed 16 bit: 18720 | Float 32 bit: 656918.8 | Octal: 111 040 141 155
Unsigned 16 bit: 18720 | Float 64 bit: 1.8264947430828E+44 | Binary: 01001001 00100000 01
☐ Show little endian decoding | ☐ Show unsigned as hexadecimal | ASCII Text: I am

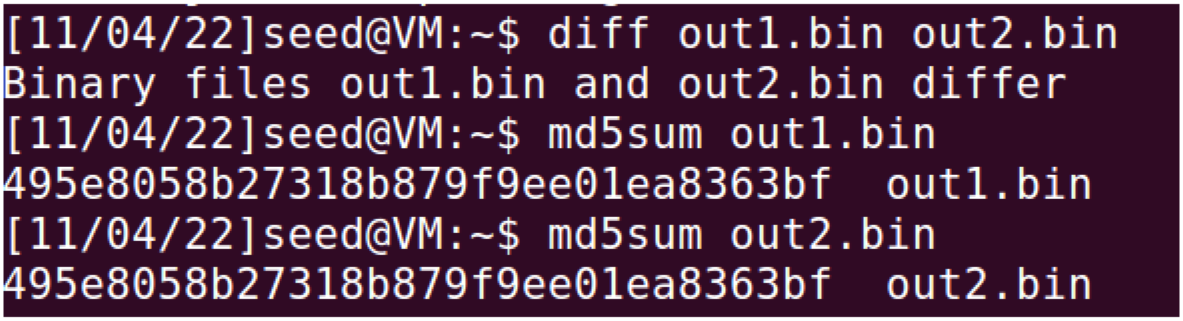
Offset: 0x0 / 0xff | Selection: None | INS

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

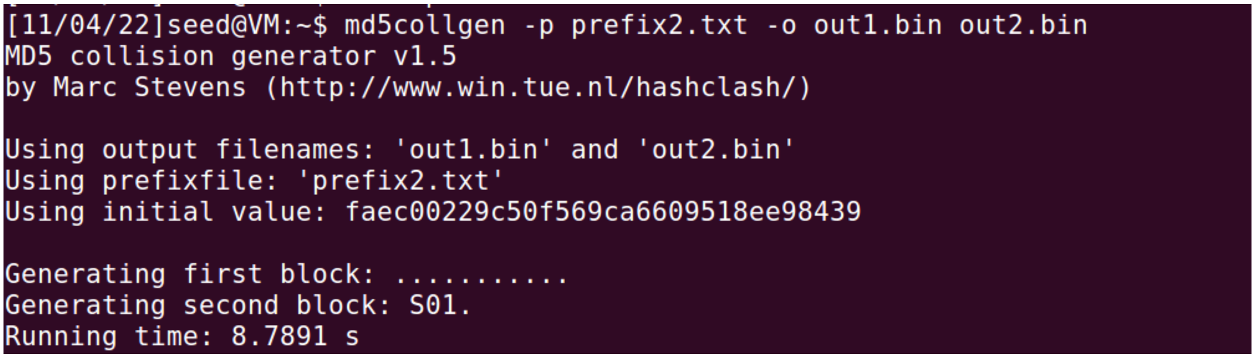
Ans. When we create a prefix file with exactly 64 bytes, no zeros are padded to the file.
We can see that no zeros are added to the below-created file as it is exactly 64 bytes in size in bless editor



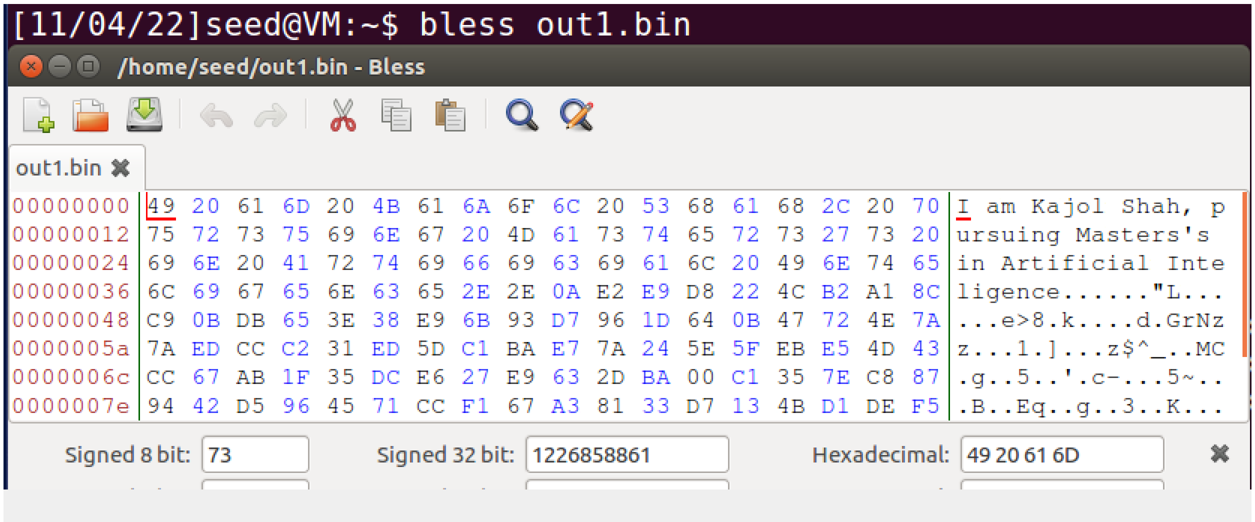
```
[11/04/22]seed@VM:~$ cat prefix2.txt
I am Kajol Shah, pursuing Masters's in Artificial Inteligence..
[11/04/22]seed@VM:~$ 
```



```
[11/04/22]seed@VM:~$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[11/04/22]seed@VM:~$ md5sum out1.bin
495e8058b27318b879f9ee01ea8363bf  out1.bin
[11/04/22]seed@VM:~$ md5sum out2.bin
495e8058b27318b879f9ee01ea8363bf  out2.bin
```



```
[11/04/22]seed@VM:~$ 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: faec00229c50f569ca6609518ee98439

Generating first block: ..........
Generating second block: S01.
Running time: 8.7891 s
```



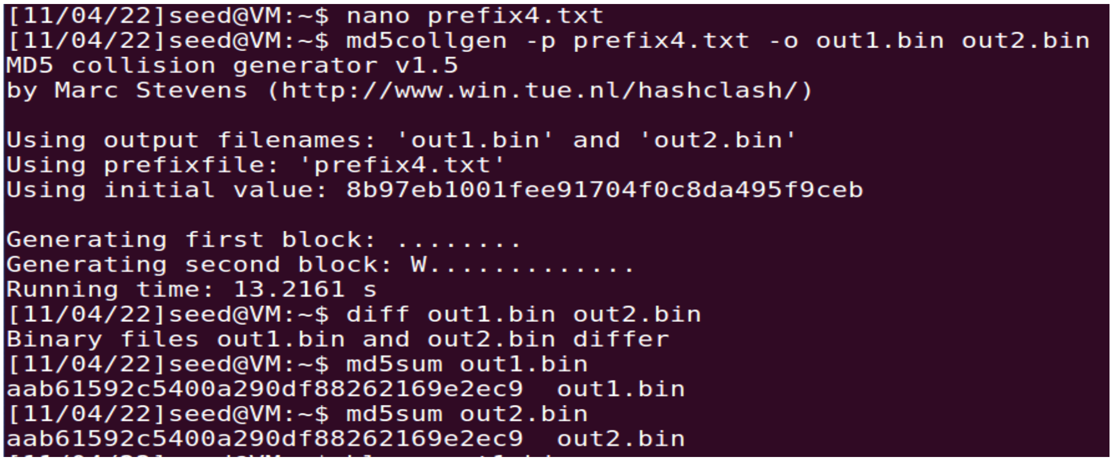
```
[11/04/22]seed@VM:~$ bless out1.bin
```

```
00000000 49 20 61 6D 20 4B 61 6A 6F 6C 20 53 68 61 68 2C 20 70 I am Kajol Shah, p
00000012 75 72 73 75 69 6E 67 20 4D 61 73 74 65 72 73 27 73 20 ursuing Masters's 
00000024 69 6E 20 41 72 74 69 66 69 63 69 61 6C 20 49 6E 74 65 in Artificial Inte
00000036 6C 69 67 65 6E 63 65 2E 2E 0A E2 E9 D8 22 4C B2 A1 8C ligence......"L...
00000048 C9 0B DB 65 3E 38 E9 6B 93 D7 96 1D 64 0B 47 72 4E 7A ...e>8.k....d.GrNz
0000005a 7A ED CC C2 31 ED 5D C1 BA E7 7A 24 5E 5F EB E5 4D 43 z...1.]...z$^_..MC
0000006c CC 67 AB 1F 35 DC E6 27 E9 63 2D BA 00 C1 35 7E C8 87 .g..5..'.c-...5~..
0000007e 94 42 D5 96 45 71 CC F1 67 A3 81 33 D7 13 4B D1 DE F5 .B..Eq..g..3..K...
```

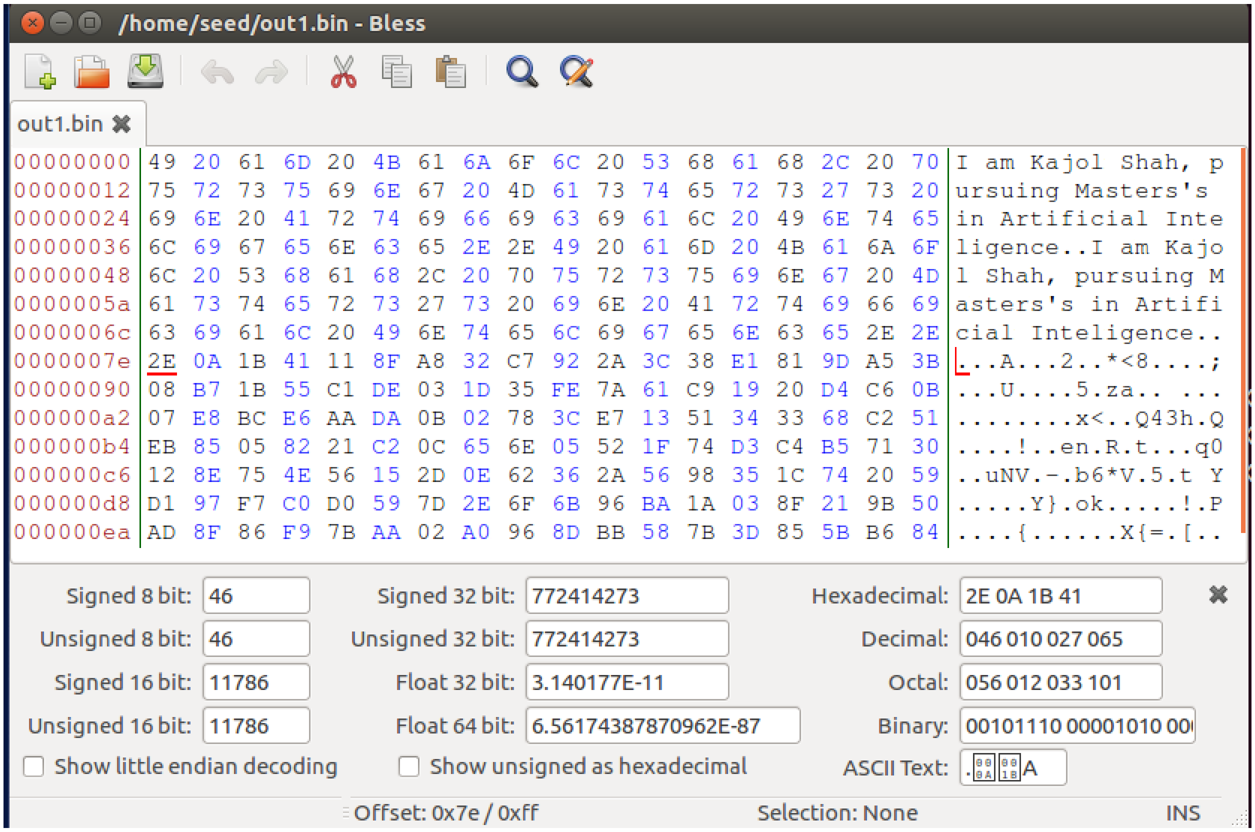
Signed 8 bit: 73   Signed 32 bit: 1226858861   Hexadecimal: 49 20 61 6D

- **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.**

Ans. The data generated by md5collgen is not completely different for the two output files. We observe that only some bytes differ in both files. The byte that differs in both the files is at the position 173 where out1.bin has the value as 13 and out2.bin has the value 93.



```
[11/04/22]seed@VM:~$ nano prefix4.txt
[11/04/22]seed@VM:~$ md5collgen -p prefix4.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: 'prefix4.txt'
Using initial value: 8b97eb1001fee91704f0c8da495f9ceb

Generating first block: ........
Generating second block: W............
Running time: 13.2161 s
[11/04/22]seed@VM:~$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[11/04/22]seed@VM:~$ md5sum out1.bin
aab61592c5400a290df88262169e2ec9  out1.bin
[11/04/22]seed@VM:~$ md5sum out2.bin
aab61592c5400a290df88262169e2ec9  out2.bin
```



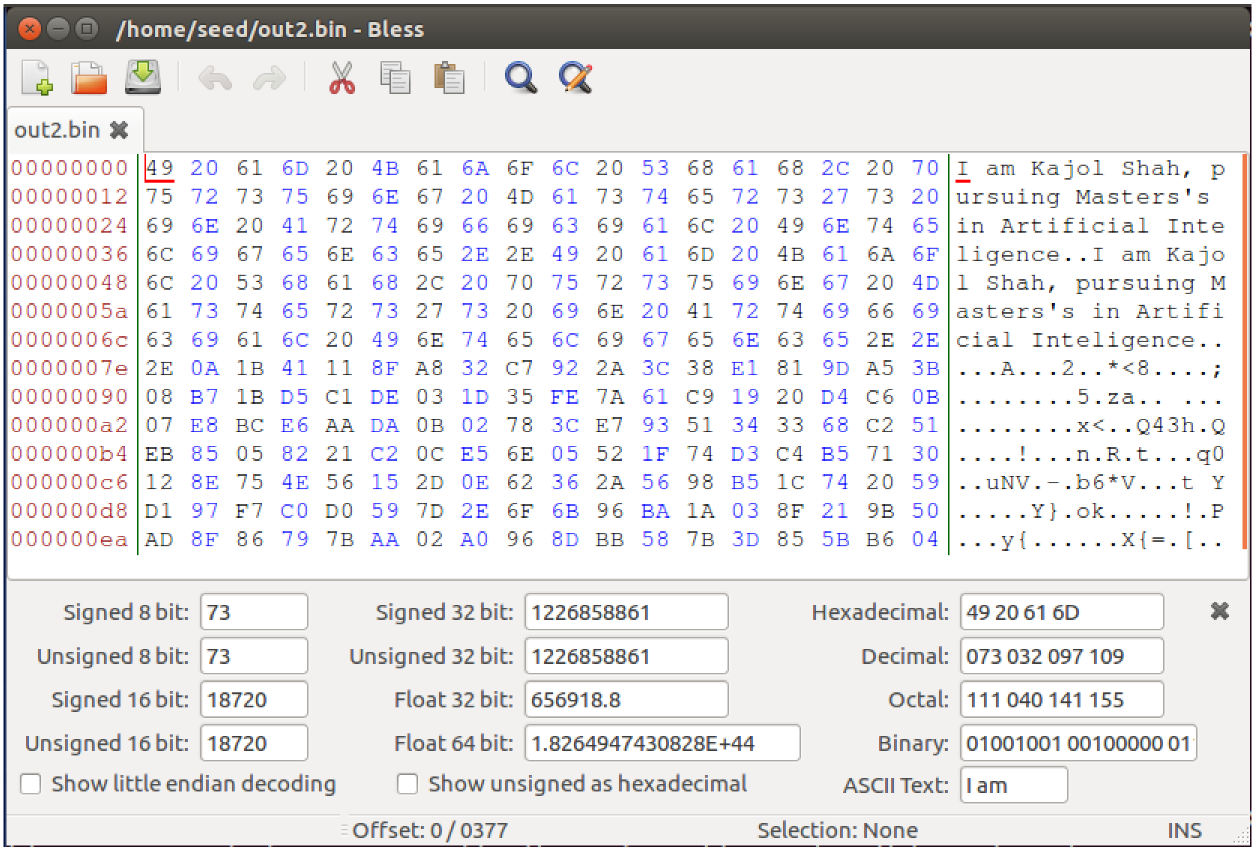
/home/seed/out1.bin - Bless

out1.bin

```
00000000 49 20 61 6D 20 4B 61 6A 6F 6C 20 53 68 61 68 2C 20 70 I am Kajol Shah, p
00000012 75 72 73 75 69 6E 67 20 4D 61 73 74 65 72 73 27 73 20 ursuing Masters's 
00000024 69 6E 20 41 72 74 69 66 69 63 69 61 6C 20 49 6E 74 65 in Artificial Inte
00000036 6C 69 67 65 6E 63 65 2E 2E 49 20 61 6D 20 4B 61 6A 6F ligence..I am Kajo
00000048 6C 20 53 68 61 68 2C 20 70 75 72 73 75 69 6E 67 20 4D l Shah, pursuing M
0000005a 61 73 74 65 72 73 27 73 20 69 6E 20 41 72 74 69 66 69 asters's in Artifi
0000006c 63 69 61 6C 20 49 6E 74 65 6C 69 67 65 6E 63 65 2E 2E cial Inteligence..
0000007e 2E 0A 1B 41 11 8F A8 32 C7 92 2A 3C 38 E1 81 9D A5 3B ...A...2..*<8....;
00000090 08 B7 1B 55 C1 DE 03 1D 35 FE 7A 61 C9 19 20 D4 C6 0B ...U....5.za.. ...
000000a2 07 E8 BC E6 AA DA 0B 02 78 3C E7 13 51 34 33 68 C2 51 ........x<..Q43h.Q
000000b4 EB 85 05 82 21 C2 0C 65 6E 05 52 1F 74 D3 C4 B5 71 30 ....!..en.R.t...q0
000000c6 12 8E 75 4E 56 15 2D 0E 62 36 2A 56 98 35 1C 74 20 59 ..uNV.-.b6*V.5.t Y
000000d8 D1 97 F7 C0 D0 59 7D 2E 6F 6B 96 BA 1A 03 8F 21 9B 50 .....Y}.ok.....!.P
000000ea AD 8F 86 F9 7B AA 02 A0 96 8D BB 58 7B 3D 85 5B B6 84 ....{......X{=.[..
```

Signed 8 bit:	46	Signed 32 bit:	772414273	Hexadecimal:	2E 0A 1B 41
Unsigned 8 bit:	46	Unsigned 32 bit:	772414273	Decimal:	046 010 027 065
Signed 16 bit:	11786	Float 32 bit:	3.140177E-11	Octal:	056 012 033 101
Unsigned 16 bit:	11786	Float 64 bit:	6.56174387870962E-87	Binary:	00101110 00001010 00
☐ Show little endian decoding		☐ Show unsigned as hexadecimal		ASCII Text:	.A

Offset: 0x7e / 0xff   Selection: None   INS



/home/seed/out2.bin - Bless

out2.bin

```
00000000 49 20 61 6D 20 4B 61 6A 6F 6C 20 53 68 61 68 2C 20 70 I am Kajol Shah, p
00000012 75 72 73 75 69 6E 67 20 4D 61 73 74 65 72 73 27 73 20 ursuing Masters's 
00000024 69 6E 20 41 72 74 69 66 69 63 69 61 6C 20 49 6E 74 65 in Artificial Inte
00000036 6C 69 67 65 6E 63 65 2E 2E 49 20 61 6D 20 4B 61 6A 6F ligence..I am Kajo
00000048 6C 20 53 68 61 68 2C 20 70 75 72 73 75 69 6E 67 20 4D l Shah, pursuing M
0000005a 61 73 74 65 72 73 27 73 20 69 6E 20 41 72 74 69 66 69 asters's in Artifi
0000006c 63 69 61 6C 20 49 6E 74 65 6C 69 67 65 6E 63 65 2E 2E cial Inteligence..
0000007e 2E 0A 1B 41 11 8F A8 32 C7 92 2A 3C 38 E1 81 9D A5 3B ...A...2..*<8....;
00000090 08 B7 1B D5 C1 DE 03 1D 35 FE 7A 61 C9 19 20 D4 C6 0B ........5.za.. ...
000000a2 07 E8 BC E6 AA DA 0B 02 78 3C E7 93 51 34 33 68 C2 51 ........x<..Q43h.Q
000000b4 EB 85 05 82 21 C2 0C E5 6E 05 52 1F 74 D3 C4 B5 71 30 ....!...n.R.t...q0
000000c6 12 8E 75 4E 56 15 2D 0E 62 36 2A 56 98 B5 1C 74 20 59 ..uNV.-.b6*V...t Y
000000d8 D1 97 F7 C0 D0 59 7D 2E 6F 6B 96 BA 1A 03 8F 21 9B 50 .....Y}.ok.....!.P
000000ea AD 8F 86 79 7B AA 02 A0 96 8D BB 58 7B 3D 85 5B B6 04 ...y{......X{=.[..
```

Signed 8 bit:	73	Signed 32 bit:	1226858861	Hexadecimal:	49 20 61 6D
Unsigned 8 bit:	73	Unsigned 32 bit:	1226858861	Decimal:	073 032 097 109
Signed 16 bit:	18720	Float 32 bit:	656918.8	Octal:	111 040 141 155
Unsigned 16 bit:	18720	Float 64 bit:	1.8264947430828E+44	Binary:	01001001 00100000 01
☐ Show little endian decoding		☐ Show unsigned as hexadecimal		ASCII Text:	I am

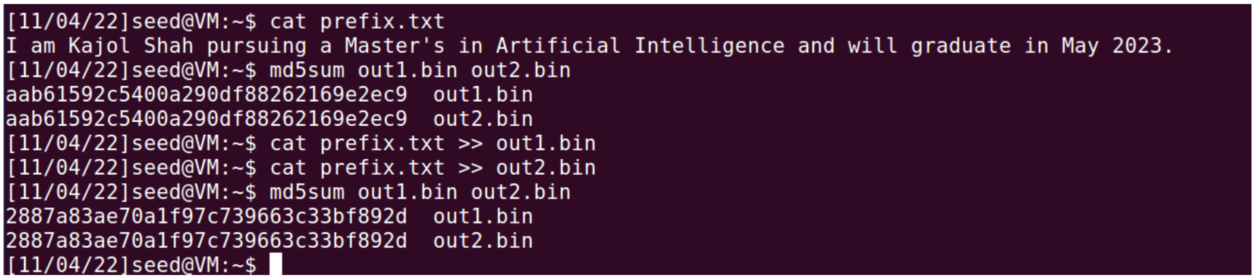
Offset: 0 / 0377   Selection: None   INS

2.2

Task 2: Understanding MD5's Property

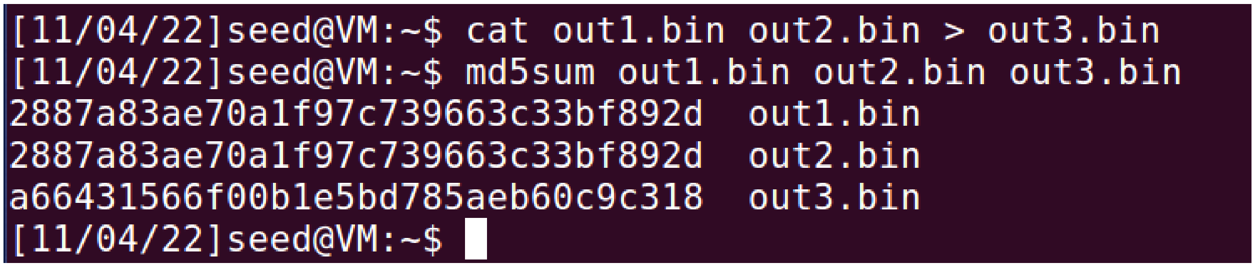
- Given two inputs M and N , if MD5(M) = MD5(N) , i.e., the MD5 hashes of M and N are the same, then for any input T , MD5(M || T) = MD5(N || T) , where || represents concatenation.
- That is, if inputs M and N have the same hash, adding the same suffix T to them will result in two outputs that have the same hash value.

Here, we will create a file prefix.txt and and check whether the MD5 hashes are the same. After that, we will append a random string to the end of both the generated files out1.bin and out2.bin and check their MD5 hashes once again.



```
[11/04/22]seed@VM:~$ cat prefix.txt
I am Kajol Shah pursuing a Master's in Artificial Intelligence and will graduate in May 2023.
[11/04/22]seed@VM:~$ md5sum out1.bin out2.bin
aab61592c5400a290df88262169e2ec9  out1.bin
aab61592c5400a290df88262169e2ec9  out2.bin
[11/04/22]seed@VM:~$ cat prefix.txt >> out1.bin
[11/04/22]seed@VM:~$ cat prefix.txt >> out2.bin
[11/04/22]seed@VM:~$ md5sum out1.bin out2.bin
2887a83ae70a1f97c739663c33bf892d  out1.bin
2887a83ae70a1f97c739663c33bf892d  out2.bin
[11/04/22]seed@VM:~$ 
```

We see that the newly generated MD5 hashes differ from the previously generated ones but are identical. This is because MD5 hash algorithm is susceptible to length extensions. Since the MD5 hashes of both files were the same, it is safe to assume that the internal state after the algorithm has been run was the same.



```
[11/04/22]seed@VM:~$ cat out1.bin out2.bin > out3.bin
[11/04/22]seed@VM:~$ md5sum out1.bin out2.bin out3.bin
2887a83ae70a1f97c739663c33bf892d  out1.bin
2887a83ae70a1f97c739663c33bf892d  out2.bin
a66431566f00b1e5bd785aeb60c9c318  out3.bin
[11/04/22]seed@VM:~$ 
```

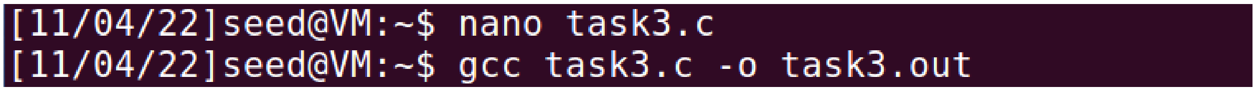
2.3

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

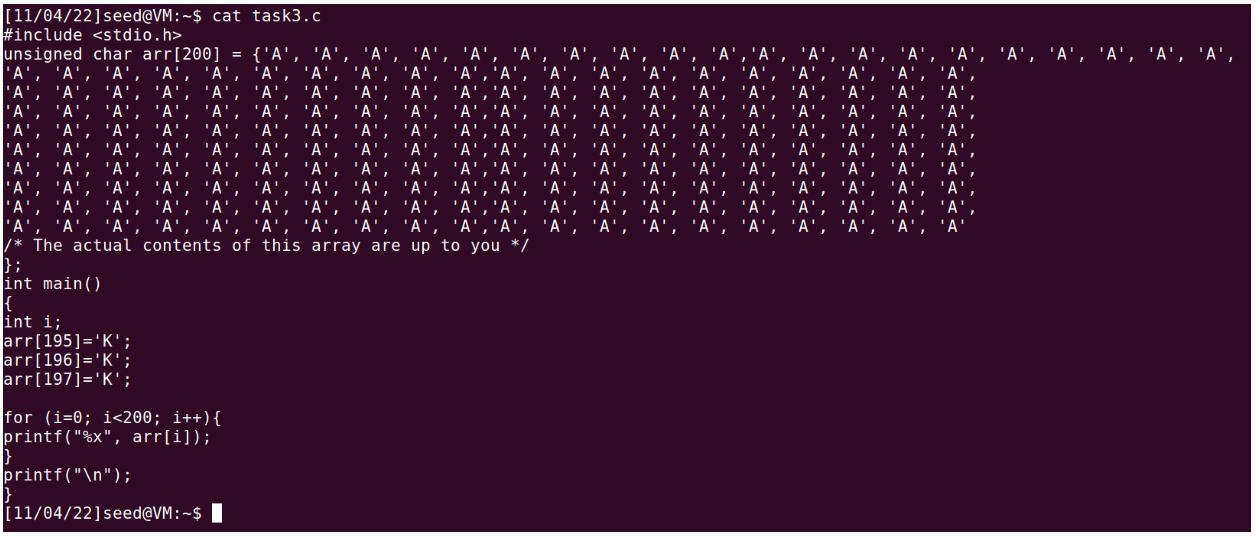
In this task, we have created two different versions of the given program, such that the contents of their xyz arrays are different, but the hash values of the executables are the same.

Writing and compiling C program

```
#include <stdio.h>
unsigned char arr[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'
/* The actual contents of this array are up to you */
};
int main()
{
int i;
arr[195]='K';
arr[196]='K';
arr[197]='K';

for (i=0; i<200; i++){
printf("%x", arr[i]);
}
printf("\n");
}
```



```
[11/04/22]seed@VM:~$ nano task3.c
[11/04/22]seed@VM:~$ gcc task3.c -o task3.out
```



```
[11/04/22]seed@VM:~$ cat task3.c
#include <stdio.h>
unsigned char arr[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'
/* The actual contents of this array are up to you */
};
int main()
{
int i;
arr[195]='K';
arr[196]='K';
arr[197]='K';

for (i=0; i<200; i++){
printf("%x", arr[i]);
}
printf("\n");
}
[11/04/22]seed@VM:~$ 
```

The byte offset is 1040 when we spot continuous blocks of A's (4160).
The executable is now down into 3 sections:

1) byte offset 0 to x -> prefix
2) x to y, and -> P
3) y to end -> suffix

MD5 (prefix || P || suffix) = MD5 (prefix || Q || suffix) is the part x to y
where the change is required, or variant.
We have kept the prefix as a multiple of 64 and a little above the byte offset of the first A.
So, as our byte offset is 4224, the prefix is the first 4288 bytes.

head -c 4288 task3.out > *prefix*

We get two files with the same hash using this prefix file for md5collgen: p1 and p2.

md5collgen -p prefix -o p1 p2

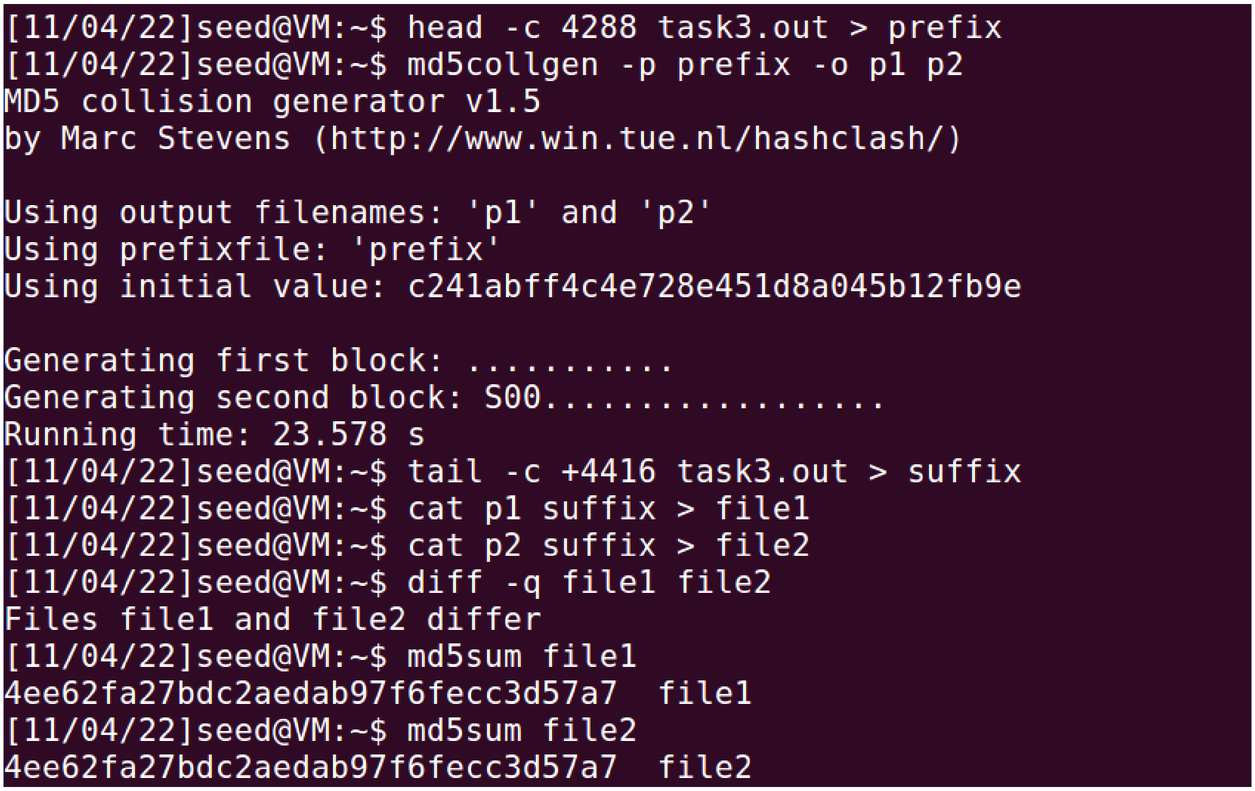
This results in files having a 10FF terminating byte offset.
So, the bytes after 10FF from the original binary are kept as the suffix.

tail -c +4416 task3.out > *suffix*

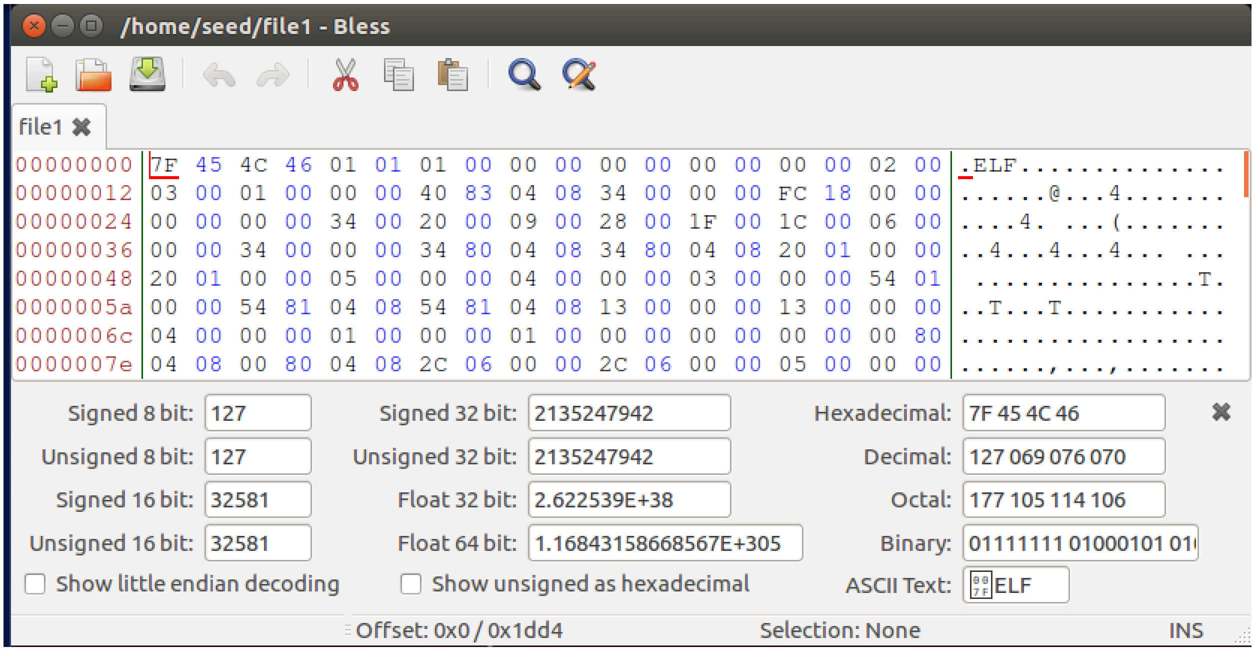
Now we concatenate the suffix to the two individual files.

cat p1 suffix > *file1*
cat p2 suffix > *file2*

We see that even though both the files file1 and file2 differ, their MD5 hashes are identical.
diff -q file1 file2
md5sum file1
md5sum file2



```
[11/04/22]seed@VM:~$ head -c 4288 task3.out > prefix
[11/04/22]seed@VM:~$ md5collgen -p prefix -o p1 p2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'p1' and 'p2'
Using prefixfile: 'prefix'
Using initial value: c241abff4c4e728e451d8a045b12fb9e

Generating first block: ..........
Generating second block: S00.................
Running time: 23.578 s
[11/04/22]seed@VM:~$ tail -c +4416 task3.out > suffix
[11/04/22]seed@VM:~$ cat p1 suffix > file1
[11/04/22]seed@VM:~$ cat p2 suffix > file2
[11/04/22]seed@VM:~$ diff -q file1 file2
Files file1 and file2 differ
[11/04/22]seed@VM:~$ md5sum file1
4ee62fa27bdc2aedab97f6fecc3d57a7  file1
[11/04/22]seed@VM:~$ md5sum file2
4ee62fa27bdc2aedab97f6fecc3d57a7  file2
```

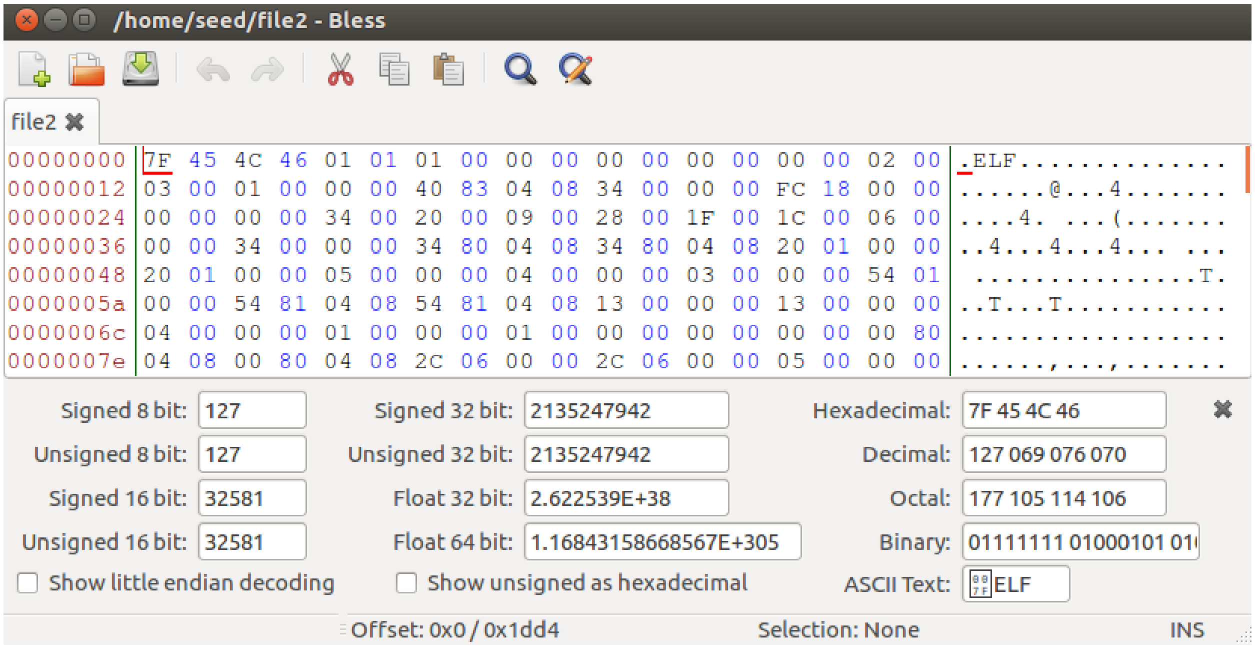


```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 .ELF..............
00000012 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 ......@...4.......
00000024 00 00 00 00 34 00 20 00 09 00 28 00 1F 00 1C 00 06 00 ....4. ...(.......
00000036 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 ..4...4...4... ...
00000048 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 54 01  ..............T.
0000005a 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 ..T...T...........
0000006c 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 ..................
0000007e 04 08 00 80 04 08 2C 06 00 00 2C 06 00 00 05 00 00 00 ......,...,.......
```

Signed 8 bit: 127
Unsigned 8 bit: 127
Signed 16 bit: 32581
Unsigned 16 bit: 32581
Signed 32 bit: 2135247942
Unsigned 32 bit: 2135247942
Float 32 bit: 2.622539E+38
Float 64 bit: 1.16843158668567E+305
Hexadecimal: 7F 45 4C 46
Decimal: 127 069 076 070
Octal: 177 105 114 106
Binary: 01111111 01000101 01
ASCII Text: ELF

□ Show little endian decoding
□ Show unsigned as hexadecimal

Offset: 0x0 / 0x1dd4
Selection: None
INS


/home/seed/file2 - Bless

```
00000000 7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 .ELF..............
00000012 03 00 01 00 00 00 40 83 04 08 34 00 00 00 FC 18 00 00 ......@...4.......
00000024 00 00 00 00 34 00 20 00 09 00 28 00 1F 00 1C 00 06 00 ....4. ...(.......
00000036 00 00 34 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 ..4...4...4... ...
00000048 20 01 00 00 05 00 00 00 04 00 00 00 03 00 00 00 54 01  ................T.
0000005a 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 ..T...T...........
0000006c 04 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00 00 80 ..................
0000007e 04 08 00 80 04 08 2C 06 00 00 2C 06 00 00 05 00 00 00 ......,...,.......
```

Signed 8 bit: 127
Unsigned 8 bit: 127
Signed 16 bit: 32581
Unsigned 16 bit: 32581
Signed 32 bit: 2135247942
Unsigned 32 bit: 2135247942
Float 32 bit: 2.622539E+38
Float 64 bit: 1.16843158668567E+305
Hexadecimal: 7F 45 4C 46
Decimal: 127 069 076 070
Octal: 177 105 114 106
Binary: 01111111 01000101 01(
ASCII Text: ELF

□ Show little endian decoding
□ Show unsigned as hexadecimal

Offset: 0x0 / 0x1dd4 Selection: None INS

So, we saw that two different binaries were created but were having the same hashes.

2.4

Task 4: Making the Two Programs Behave Differently

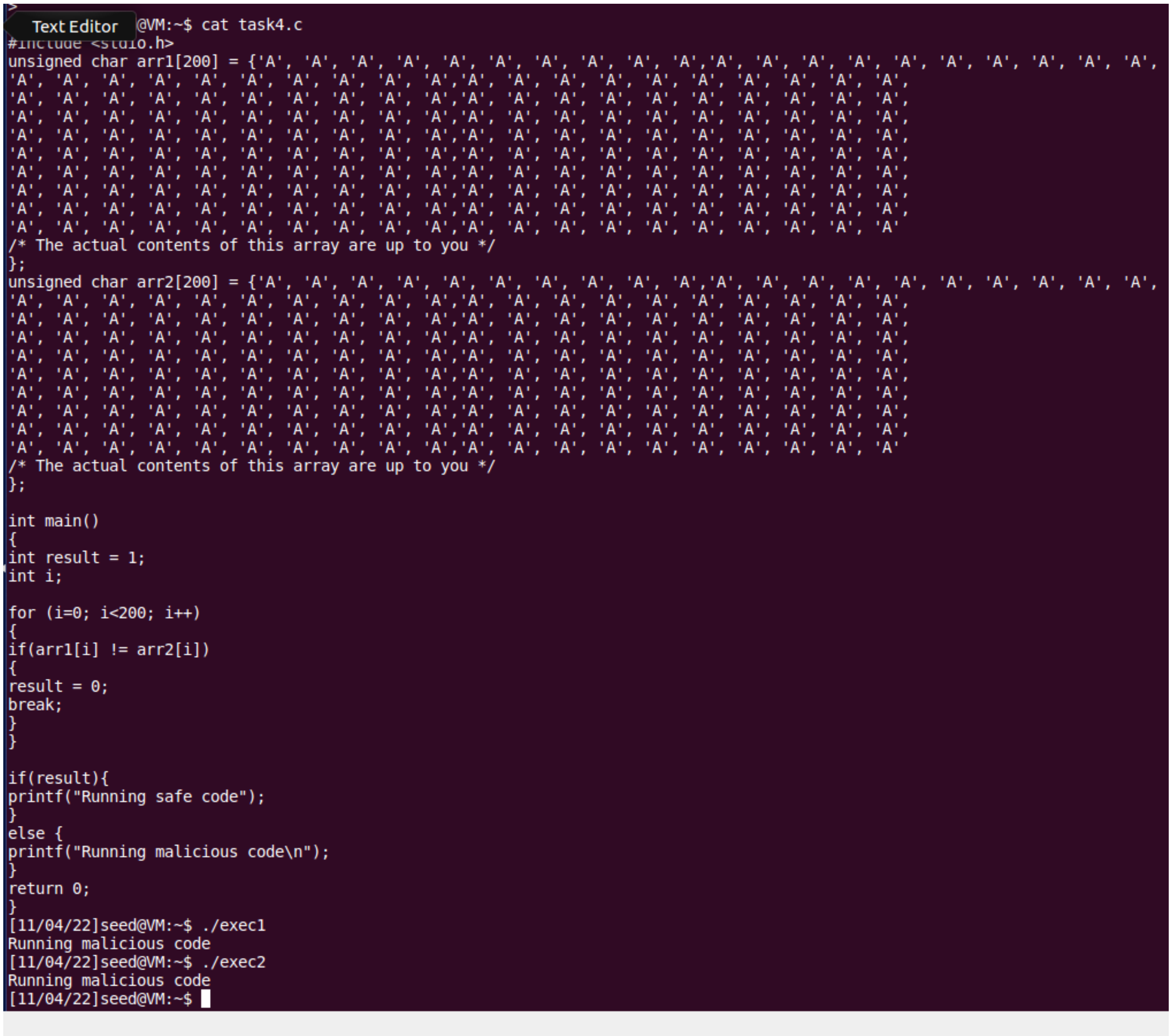
In this task, we create two different programs. One program will always execute benign instructions and do good things, while the other program will execute malicious instructions and cause damages. We find a way to get these two programs to share the same MD5 hash value .

Below is the program written in C.

```
#include <stdio.h>
unsigned char arr1[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'
/* The actual contents of this array are up to you */
};
unsigned char arr2[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'
/* The actual contents of this array are up to you */
};

int main()
{
int result = 1;
int i;

for (i=0; i<200; i++)
{
if(arr1[i] != arr2[i])
{
result = 0;
break;
}
}

if(result){
printf("Running safe code");
}
else {
printf("Running malicious code\n");
}
return 0;
}
```

We will compile the above code with the following command:

```
gcc task3.c -o task4.out
```



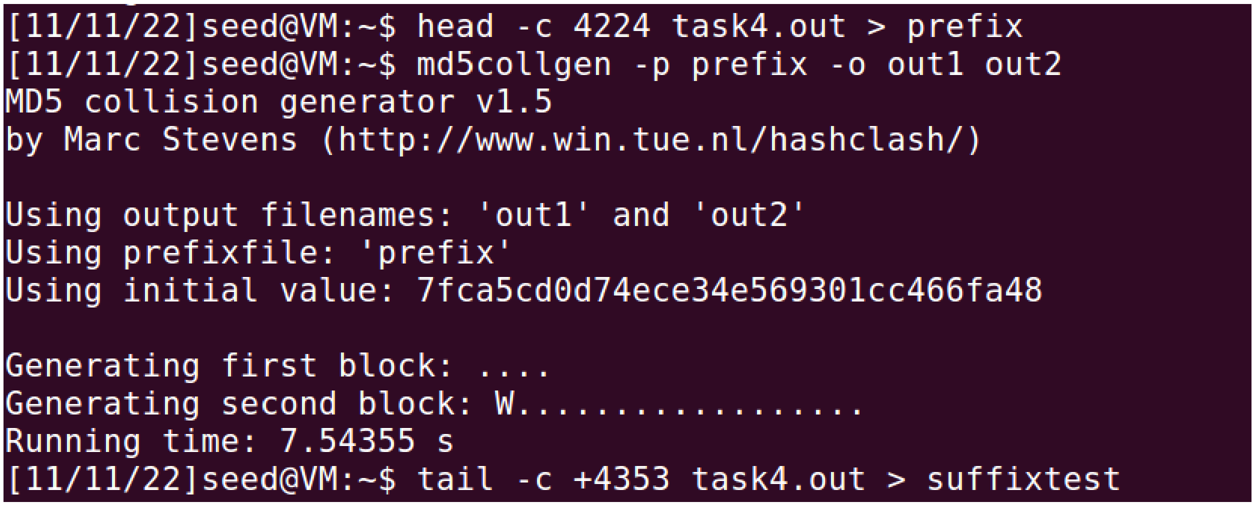
```
>
@VM:~$ cat task4.c
#include <stdio.h>
unsigned char arr1[200] = {'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A','A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', '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'
/* The actual contents of this array are up to you */
};
unsigned char arr2[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'
/* The actual contents of this array are up to you */
};

int main()
{
int result = 1;
int i;

for (i=0; i<200; i++)
{
if(arr1[i] != arr2[i])
{
result = 0;
break;
}
}

if(result){
printf("Running safe code");
}
else {
printf("Running malicious code\n");
}
return 0;
}
[11/04/22]seed@VM:~$ ./exec1
Running malicious code
[11/04/22]seed@VM:~$ ./exec2
Running malicious code
[11/04/22]seed@VM:~$ 
```

We set the prefix and then generate two files using this prefix, which gives out1 and out2 files having all except the last 8 elements of the first array. Then, we add all bytes after the 4352nd byte in task4.out to suffixtest.

head -c 4224 task4.out > prefix

md5collgen -p prefix -o out1 out2

tail -c +4353 task4.out > suffixtest



```
[11/11/22]seed@VM:~$ head -c 4224 task4.out > prefix
[11/11/22]seed@VM:~$ 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: 7fca5cd0d74ece34e569301cc466fa48

Generating first block: ....
Generating second block: W..................
Running time: 7.54355 s
[11/11/22]seed@VM:~$ tail -c +4353 task4.out > suffixtest
```

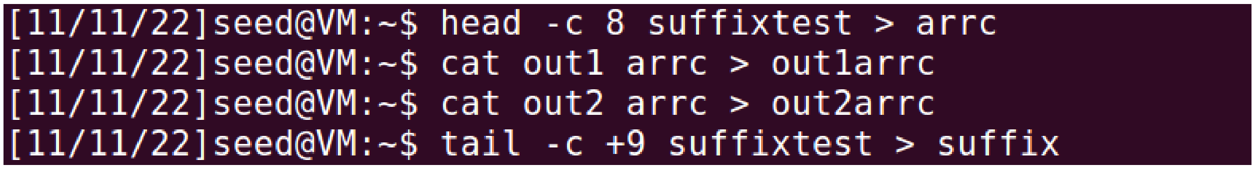
We then add the first 8 bytes of suffixtest to both out1 and out2 which gives files out1arrc and out2arrc. After that, we create the suffix file which contains all bytes after the 8th byte in suffixtest.

head -c 8 suffixtest > arrc

cat out1 arrc > out1arrc

cat out2 arrc > out2arrc

tail -c +9 suffixtest > suffix



```
[11/11/22]seed@VM:~$ head -c 8 suffixtest > arrc
[11/11/22]seed@VM:~$ cat out1 arrc > out1arrc
[11/11/22]seed@VM:~$ cat out2 arrc > out2arrc
[11/11/22]seed@VM:~$ tail -c +9 suffixtest > suffix
```

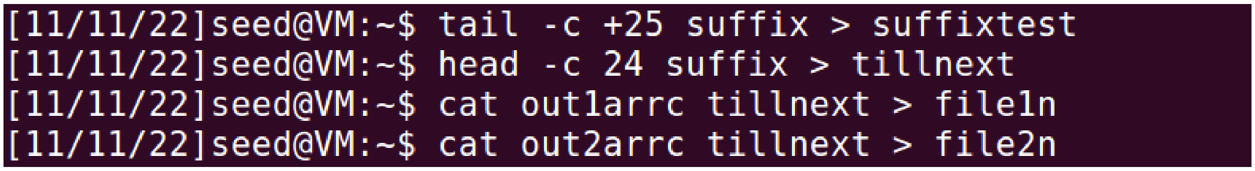
Now, we add the bytes between the ending of the first array and the beginning of the second array and create a file tillnext. We store the bytes beginning with the second array in suffix to suffixtest and add these bytes to out1arrc and out2arrc which gives file1n and file2n.

tail -c +25 suffix > suffixtest

head -c 24 suffix > tillnext

cat out1arrc tillnext > file1n

cat out2arrc tillnext > file2n



```
[11/11/22]seed@VM:~$ tail -c +25 suffix > suffixtest
[11/11/22]seed@VM:~$ head -c 24 suffix > tillnext
[11/11/22]seed@VM:~$ cat out1arrc tillnext > file1n
[11/11/22]seed@VM:~$ cat out2arrc tillnext > file2n
```

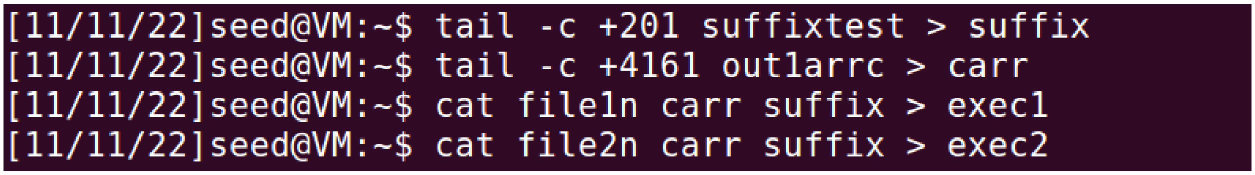
The two result files are the two separate parts which have the contents up to beginning of the second array. The attack will be successful if one file prints "Running safe code!!!" while the other prints "Running malicious code!!!". For this, the contents of the second array needs to be equal to one of the generated arrays. So, we put the bytes after the second array in suffixtest to suffix. The we copy the first array from out1arrc to carr. The file carr can be appended to file1n and file2n along with suffix which gives the final executables exec1 and exec2.

tail -c +201 suffixtest > suffix

tail -c +4161 out1arrc > carr

cat file1n carr suffix > exec1

cat file2n carr suffix > exec2



```
[11/11/22]seed@VM:~$ tail -c +201 suffixtest > suffix
[11/11/22]seed@VM:~$ tail -c +4161 out1arrc > carr
[11/11/22]seed@VM:~$ cat file1n carr suffix > exec1
[11/11/22]seed@VM:~$ cat file2n carr suffix > exec2
```

We calculate the md5sum and make both the files executable

md5sum exec1

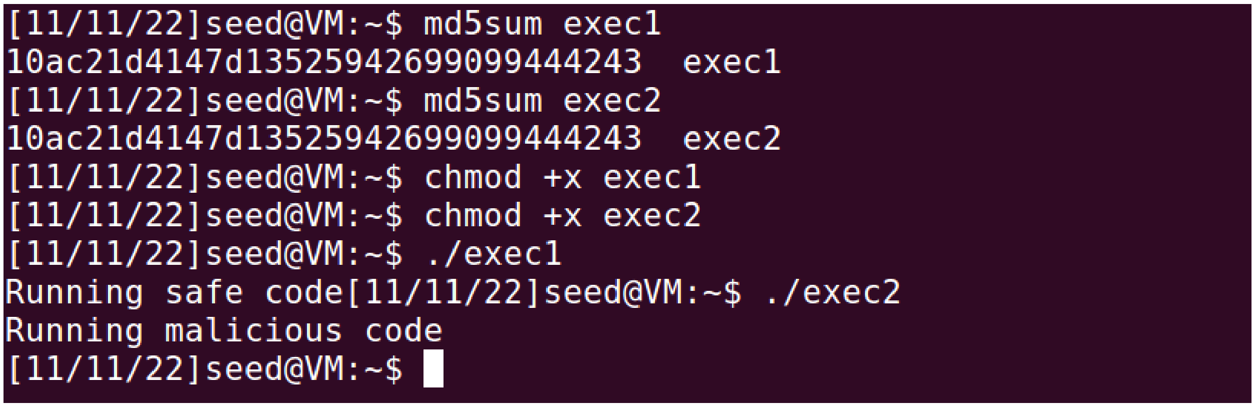
md5sum exec2

chmod +x exec1

chmod +x exec2

./exec1

./exec2



```
[11/11/22]seed@VM:~$ md5sum exec1
10ac21d4147d13525942699099444243  exec1
[11/11/22]seed@VM:~$ md5sum exec2
10ac21d4147d13525942699099444243  exec2
[11/11/22]seed@VM:~$ chmod +x exec1
[11/11/22]seed@VM:~$ chmod +x exec2
[11/11/22]seed@VM:~$ ./exec1
Running safe code[11/11/22]seed@VM:~$ ./exec2
Running malicious code
[11/11/22]seed@VM:~$ █
```

In this way, we exploit md5 vulenerability