

Name – MANSI Patil

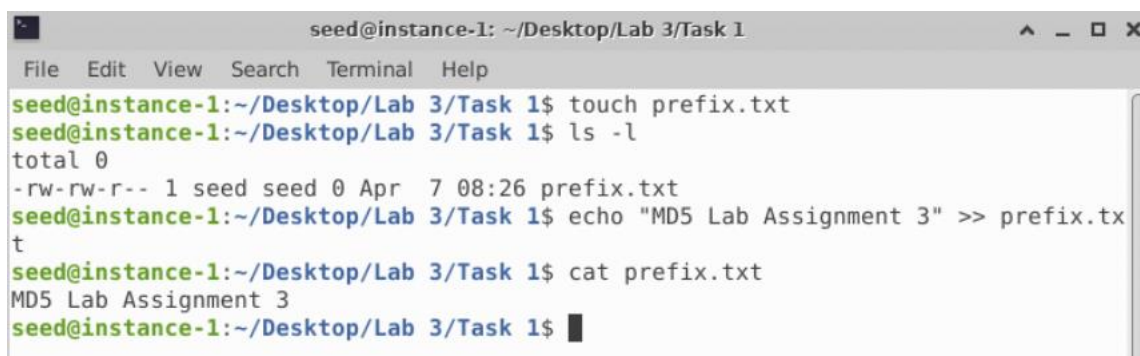
CWID – A20549858

Lab 3 - MD5 Collision Attack Lab

CS 458 – Introduction to Information Security

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

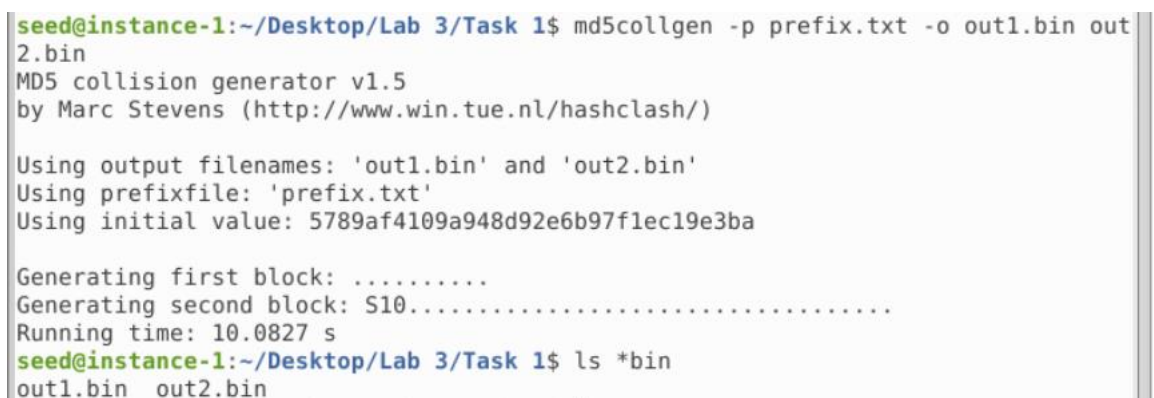
- 1) Let's begin with creating a prefix file as shown below.

A terminal window titled 'seed@instance-1: ~/Desktop/Lab 3/Task 1' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ touch prefix.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ ls -l
total 0
-rw-rw-r-- 1 seed seed 0 Apr  7 08:26 prefix.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ echo "MD5 Lab Assignment 3" >> prefix.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ cat prefix.txt
MD5 Lab Assignment 3
seed@instance-1:~/Desktop/Lab 3/Task 1$
```

- touch: It is used to create empty files.
- Cat: It is used to concatenate and display the content of files.
- echo: It is used to display a line of text or to redirect text into a file.

- 2) Next, run the md5collgen command as follow:

A terminal window showing the execution of the md5collgen command. The output is as follows:

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ 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: 5789af4109a948d92e6b97f1ec19e3ba

Generating first block: .....
Generating second block: S10.....
Running time: 10.0827 s
seed@instance-1:~/Desktop/Lab 3/Task 1$ ls *bin
out1.bin out2.bin
```

- md5collgen is a tool used to generate MD5 hash collisions.
- MD5 is a cryptographic hash function that produces a 128-bit hash value.
- A hash collision occurs when two different inputs produce the same hash value.

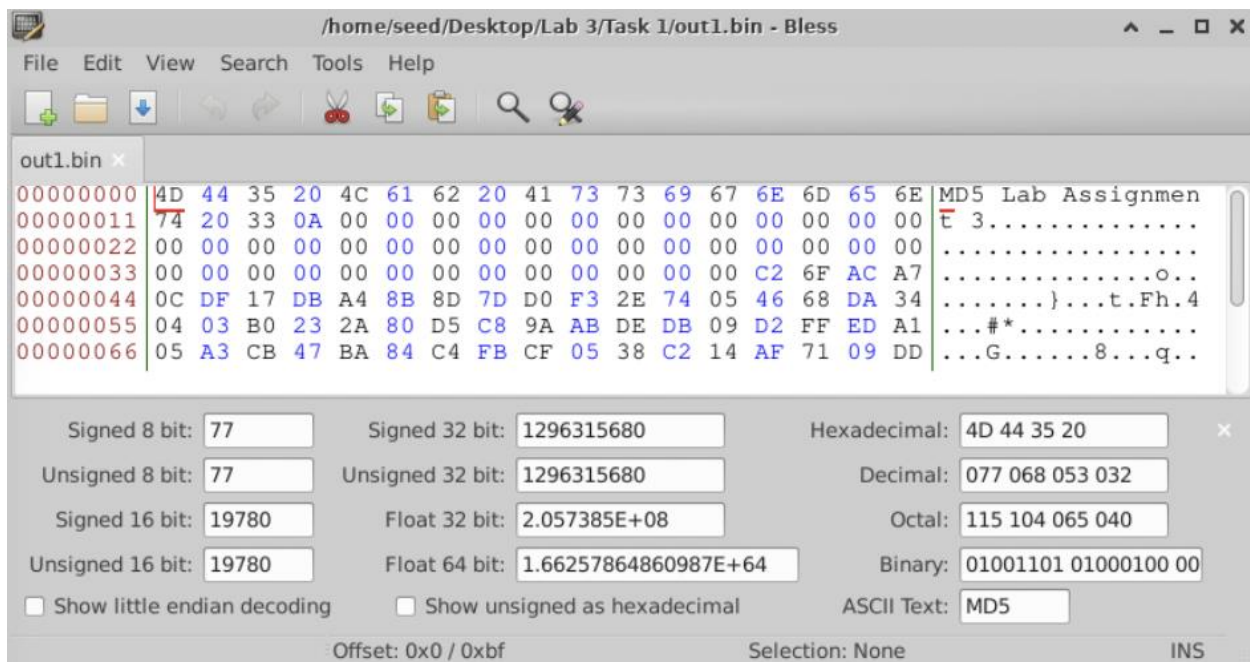
The above command generates two output files, out1.bin and out2.bin, for a given a prefix file prefix.txt.

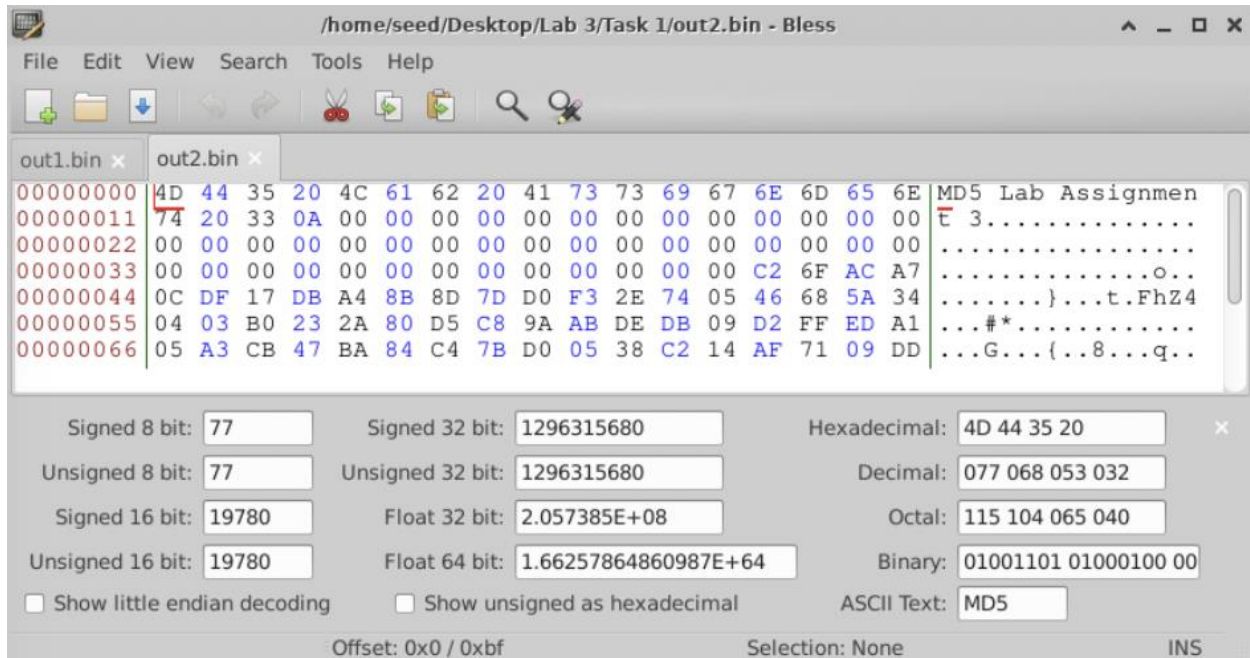
3) To check that the output files are distinct or not, we use the diff command.

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
```

4) Next, we use the bless editor to check the binary files for out1, out2.

- Bless is a hex editor designed for reading and manipulating binary files. It allows you to view and change the hexadecimal and ASCII representations of files.





- We can see that the values for both output files out1 and out2 are the same.

5) Next, we use the md5sum command to check the MD5 hash of each output file.

- md5sum calculates and verifies MD5 checksums for files.
- It takes a file as input, computes its MD5 hash, and outputs a 32-character hexadecimal number representing the hash value.

```
seed@instance-1: ~/Desktop/Lab 3/Task 1
File Edit View Search Terminal Help
seed@instance-1:~/Desktop/Lab 3/Task 1$ md5sum out1.bin
639a937d6597b8c355102469778c3f88 out1.bin
seed@instance-1:~/Desktop/Lab 3/Task 1$ md5sum out2.bin
639a937d6597b8c355102469778c3f88 out2.bin
seed@instance-1:~/Desktop/Lab 3/Task 1$
```

We can observe that the MD5 hash values for both the output files out1, out2 are same.

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

- Files are padded with 0's.
- Message is padded with bits to produce the length of the message \approx modulo of 512.
- This padding includes a single '1' bit followed by '0' bits to fill the space, and then the length of the original message in bits is appended to the end
- if the length of the prefix file is not a multiple of 64 bytes, padding is added to the file to meet requirement.

Create new prefix text file:

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ echo "Mansi Patil CSIIT" >> prefix_1.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ echo "$(\python -c 'print("A"*63)')'" >> p
refix_64.txt
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3
seed@instance-1:~/Desktop/Lab 3/Task 1$ echo "$(\python3 -c 'print("A"*63)')'" >>
prefix_64.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ ls -l
total 24
-rw-rw-r-- 1 seed seed 192 Apr  7 08:32 out1.bin
-rw-rw-r-- 1 seed seed 192 Apr  7 08:32 out2.bin
-rw-rw-r-- 1 seed seed  2 Apr  7 16:27 prefix
-rw-rw-r-- 1 seed seed 21 Apr  7 08:27 prefix.txt
-rw-rw-r-- 1 seed seed 18 Apr  7 16:31 prefix_1.txt
-rw-rw-r-- 1 seed seed 65 Apr  7 16:32 prefix_64.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ ls -l *.txt
-rw-rw-r-- 1 seed seed 21 Apr  7 08:27 prefix.txt
-rw-rw-r-- 1 seed seed 18 Apr  7 16:31 prefix_1.txt
-rw-rw-r-- 1 seed seed 65 Apr  7 16:32 prefix_64.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$
```

Output filenames: Out1_1.bin and out1_64.bin

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ md5collgen -p prefix_1.txt -o out1_1.bin
out1_2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_1.bin' and 'out1_2.bin'
Using prefixfile: 'prefix_1.txt'
Using initial value: ae155900cc63a0eb7d97f42f4f1d7b8f

Generating first block: .....
Generating second block: S00..
Running time: 11.4852 s
seed@instance-1:~/Desktop/Lab 3/Task 1$

seed@instance-1:~/Desktop/Lab 3/Task 1$ md5collgen -p prefix_64.txt -o out64_1.b
in out64_2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out64_1.bin' and 'out64_2.bin'
Using prefixfile: 'prefix_64.txt'
Using initial value: d46b988132ccf1a01611c6fb99de1691

Generating first block: .....
Generating second block: S11.....
.....
Running time: 17.3553 s
seed@instance-1:~/Desktop/Lab 3/Task 1$
```


Bless: out1_1.bin & out64_1.bin

/home/seed/Desktop/Lab 3/Task 1/out1_1.bin - Bless

File Edit View Search Tools Help

out1_1.bin x

00000000	4D	61	6E	73	69	20	50	61	74	69	6C	20	43	53	49	49	54	Mansi Patil CSIIT
00000011	0A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000022	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000033	00	00	00	00	00	00	00	00	00	00	00	00	00	B0	46	20	35F 5
00000044	A2	87	00	02	49	C6	AD	EE	B8	D6	7B	16	03	A8	47	F1	90I.....{...G..
00000055	D6	8E	BE	23	BA	3A	19	87	B4	31	DF	34	B1	B6	E6	DE	6E	...#.:.:..1.4....n
00000066	4C	90	6B	26	D4	C4	C3	9D	29	48	60	7A	A6	43	70	5F	EC	L.k&....)H`z.Cp_.

Signed 8 bit: 77 Signed 32 bit: 1298230899 Hexadecimal: 4D 61 6E 73

Unsigned 8 bit: 77 Unsigned 32 bit: 1298230899 Decimal: 077 097 110 115

Signed 16 bit: 19809 Float 32 bit: 2.36382E+08 Octal: 115 141 156 163

Unsigned 16 bit: 19809 Float 64 bit: 5.73670559829661E+64 Binary: 01001101 01100001 01

☐ Show little endian decoding ☐ Show unsigned as hexadecimal ASCII Text: Mans

/home/seed/Desktop/Lab 3/Task 1/out64_1.bin - Bless

File Edit View Search Tools Help

out64_1.bin x

00000000	0A	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	.AAAAAAAAAAAAAAAAA
00000011	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAAA
00000022	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAAA
00000033	41	41	41	41	41	41	41	41	41	41	41	41	41	0A	00	00	00	AAAAAAAAAAAAA....
00000044	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000055	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000066	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Signed 8 bit: 10 Signed 32 bit: 172048705 Hexadecimal: 0A 41 41 41

Unsigned 8 bit: 10 Unsigned 32 bit: 172048705 Decimal: 010 065 065 065

Signed 16 bit: 2625 Float 32 bit: 9.304885E-33 Octal: 012 101 101 101

Unsigned 16 bit: 2625 Float 64 bit: 2.80560313421922E-259 Binary: 00001010 01000001 01

☐ Show little endian decoding ☐ Show unsigned as hexadecimal ASCII Text: ↵AAA

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

- The 64-byte prefix eliminates the need for padding with zero bytes.
- The prefix file is followed by some randomly generated data for collision.

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.

- We can observe that there are minor difference between out1_1.bin and out1_2.bin files.

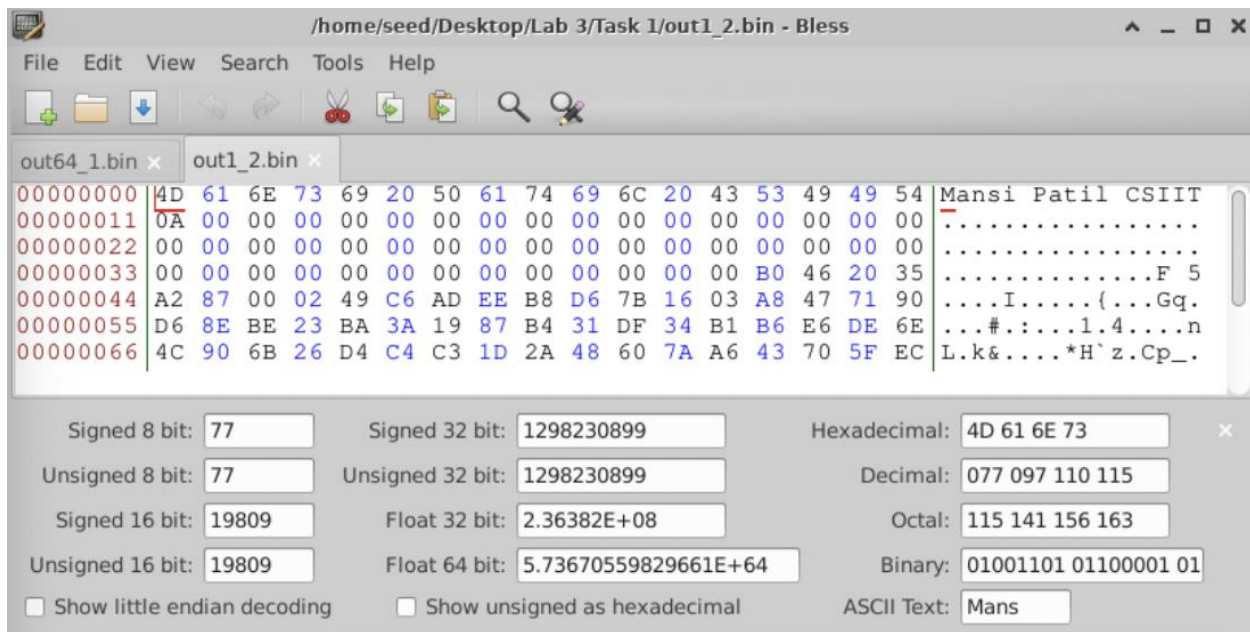
The screenshot shows the Bless hex editor interface. The title bar indicates the file path is `/home/seed/Desktop/Lab 3/Task 1/out1_1.bin - Bless`. The menu bar includes File, Edit, View, Search, Tools, and Help. The toolbar contains icons for file operations and editing. The main window displays a hex dump of the file `out1_1.bin`. The first 64 bytes (addresses 00000000 to 00000066) are highlighted in red. The hex dump shows the following data:

Address	Hex	ASCII
00000000	4D 61 6E 73 69 20 50 61 74 69 6C 20 43 53 49 49 54	Mansi Patil CSIIT
00000011	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000022	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000033	00 00 00 00 00 00 00 00 00 00 00 00 00 00 B0 46 20 35F 5
00000044	A2 87 00 02 49 C6 AD EE B8 D6 7B 16 03 A8 47 F1 90I.....{ ...G..
00000055	D6 8E BE 23 BA 3A 19 87 B4 31 DF 34 B1 B6 E6 DE 6E	...#.:.1.4....n
00000066	4C 90 6B 26 D4 C4 C3 9D 29 48 60 7A A6 43 70 5F EC	L.k&....)H`z.Cp_.

Below the hex dump, the Bless editor provides various conversion options for the selected data (hex value 4D 61 6E 73):

Conversion	Value
Signed 8 bit:	77
Unsigned 8 bit:	77
Signed 16 bit:	19809
Unsigned 16 bit:	19809
Signed 32 bit:	1298230899
Unsigned 32 bit:	1298230899
Float 32 bit:	2.36382E+08
Float 64 bit:	5.73670559829661E+64
Hexadecimal:	4D 61 6E 73
Decimal:	077 097 110 115
Octal:	115 141 156 163
Binary:	01001101 01100001 01
ASCII Text:	Mans

At the bottom, there are checkboxes for ☐ Show little endian decoding and ☐ Show unsigned as hexadecimal.



```
seed@instance-1:~/Desktop/Lab 3/Task 1$ md5sum out1_1.bin
25c3885f2c2002384f1b0c0a02462c73 out1_1.bin
seed@instance-1:~/Desktop/Lab 3/Task 1$ md5sum out64_1.bin
68760d3b96d19e064ada4be27422f7fa out64_1.bin
seed@instance-1:~/Desktop/Lab 3/Task 1$
```

- They look relatively similar but there exist some byte difference between them.

```
seed@instance-1:~/Desktop/Lab 3/Task 1$ xxd out1_1.bin > 1.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ xxd out1_2.bin > 2.txt
seed@instance-1:~/Desktop/Lab 3/Task 1$ diff 1.txt 2.txt
6,8c6,8
< 00000050: 03a8 47f1 90d6 8ebe 23ba 3a19 87b4 31df ..G....#....1.
< 00000060: 34b1 b6e6 de6e 4c90 6b26 d4c4 c39d 2948 4....nL.k&....)H
< 00000070: 607a a643 705f ecca de80 8ea8 68a7 a424 `z.Cp_.....h..$
---
> 00000050: 03a8 4771 90d6 8ebe 23ba 3a19 87b4 31df ..Gq....#....1.
> 00000060: 34b1 b6e6 de6e 4c90 6b26 d4c4 c31d 2a48 4....nL.k&....*H
> 00000070: 607a a643 705f ecca de80 8e28 68a7 a424 `z.Cp_.....(h..$
10,12c10,12
< 00000090: ba67 c4eb 12b9 6015 d5bb 14c1 d061 799e .g....`.....ay.
< 000000a0: 7b33 16be 6265 8361 a174 b50f 51ee 57c6 {3..be.a.t..Q.W.
< 000000b0: bdb9 fe07 26c3 e6bc f7df e57d a8d8 3822 ....&.....}.8"
---
> 00000090: ba67 c46b 12b9 6015 d5bb 14c1 d061 799e .g.k...`.....ay.
> 000000a0: 7b33 16be 6265 8361 a174 b50f 516e 57c6 {3..be.a.t..QnW.
> 000000b0: bdb9 fe07 26c3 e6bc f7df e5fd a8d8 3822 ....&.....8"
```

2.2 Task 2: Understanding MD5's Property

Create a file: file1.txt and file2.txt

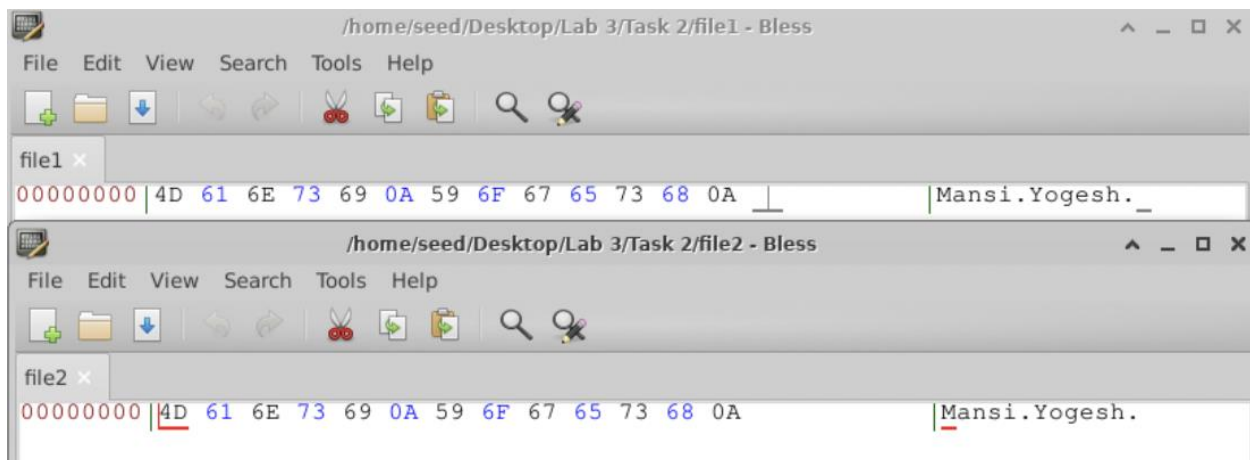
we observe they have the same checksums for MD5 algorithm.

```
seed@instance-1: ~/Desktop/Lab 3/Task 2
File Edit View Search Terminal Help
seed@instance-1:~/Desktop/Lab 3/Task 2$ echo "Mansi" >> file1.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$ echo "Mansi" >> file2.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file1.txt
983cc90d4717bfa89f9b1f58ec53cfad  file1.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file2.txt
983cc90d4717bfa89f9b1f58ec53cfad  file2.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$ echo "Yogesh" >> file3.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$
```

Concatenation and MD5sum:

Here, Cat command is used for concatenate the files.

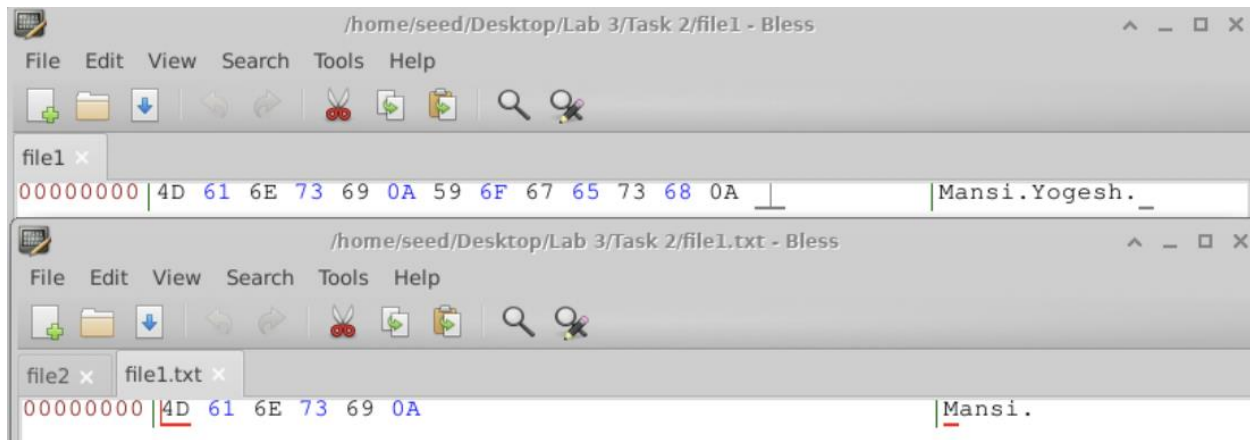
```
seed@instance-1:~/Desktop/Lab 3/Task 2$ cat file1.txt file3.txt > file1
seed@instance-1:~/Desktop/Lab 3/Task 2$ cat file2.txt file3.txt > file2
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file1
365826cf558e98b66182447ebad3adb4  file1
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file2
365826cf558e98b66182447ebad3adb4  file2
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file1.txt
983cc90d4717bfa89f9b1f58ec53cfad  file1.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$ md5sum file2.txt
983cc90d4717bfa89f9b1f58ec53cfad  file2.txt
seed@instance-1:~/Desktop/Lab 3/Task 2$
```



```
/home/seed/Desktop/Lab 3/Task 2/file1 - Bless
File Edit View Search Tools Help
00000000 | 4D 61 6E 73 69 0A 59 6F 67 65 73 68 0A | Mansi.Yogesh._

/home/seed/Desktop/Lab 3/Task 2/file2 - Bless
File Edit View Search Tools Help
00000000 | 4D 61 6E 73 69 0A 59 6F 67 65 73 68 0A | Mansi.Yogesh._
```


- From the above screenshots, we can conclude that file1 and file2 have the same checksums.



The checksums are different for file1.txt & file1, due to the concatenation.

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


1) First generate the file.



The screenshot shows a code editor window titled 'code.c' with the path '~/Desktop/Lab 3/Task 3'. The code is as follows:

```
1 #include <stdio.h>
2 unsigned char xyz[200] = {
3     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
4     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
5     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
6     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
7     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
8     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
9     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
10    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
11    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
12    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
13    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
14    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
15    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
16    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
17    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
18    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
19    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
20    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
21    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
22    0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,
23 };
24 int main()
25 {
26     int i;
27     for (i=0; i<200; i++){
28         printf("%x", xyz[i]);
29     }
30     printf("\n");
31 }
```

2) Run the file



The screenshot shows a terminal window with the prompt 'seed@instance-1: ~/Desktop/Lab 3/Task 3'. The commands and output are as follows:

```
seed@instance-1:~/Desktop/Lab 3/Task 3$ gedit code.c
seed@instance-1:~/Desktop/Lab 3/Task 3$ gcc code.c -o code.out
seed@instance-1:~/Desktop/Lab 3/Task 3$
```

- The gcc command is used to compile C code into an executable file.
- The -o option is used to specify the output file name for the executable.

$$4160 / 64 = 65$$

So, we consider the first 4224 bytes of the file & save it as prefix.

generate a hash value like prefix that was generated from execution file.



```

seed@instance-1:~/Desktop/Lab 3/Task 3$ head -c 4224 code.out > prefix
seed@instance-1:~/Desktop/Lab 3/Task 3$ md5collgen -p prefix -o code.c_a.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Error: exactly two output filenames should be specified.
seed@instance-1:~/Desktop/Lab 3/Task 3$ md5collgen -p prefix -o code.c_a.bin code.c_b.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'code.c_a.bin' and 'code.c_b.bin'
Using prefixfile: 'prefix'
Using initial value: 070f1cc7b9c5698310bb6c6f53bea269

Generating first block: .....
.....
Generating second block: W.....
Running time: 171.665 s
seed@instance-1:~/Desktop/Lab 3/Task 3$ ls *.bin
code.c_a.bin  code.c_b.bin

```

Now, generate suffix:

As prefix is a multiple of 64, we take 4224 bytes of executable file i.e., multiple of 64 bytes.

$4224 / 128 = 4352$

Generate Suffix –

```

seed@instance-1:~/Desktop/Lab 3/Task 3$ tail -c +4352 code.out > suffix
seed@instance-1:~/Desktop/Lab 3/Task 3$

```

Concatenate p & q files i.e., suffix and prefix –

```

seed@instance-1:~/Desktop/Lab 3/Task 3$ tail -c 128 code.c_a.bin > p
seed@instance-1:~/Desktop/Lab 3/Task 3$ tail -c 128 code.c_b.bin > q
seed@instance-1:~/Desktop/Lab 3/Task 3$ cat prefix p suffix > task3_1
seed@instance-1:~/Desktop/Lab 3/Task 3$ cat prefix q suffix > task3_2
seed@instance-1:~/Desktop/Lab 3/Task 3$

```

Check the hash values for task3_1 and task3_2 -

```

seed@instance-1:~/Desktop/Lab 3/Task 3$ md5sum task3_1
56de303029aec2e0968130593da5457a  task3_1
seed@instance-1:~/Desktop/Lab 3/Task 3$ md5sum task3_2
56de303029aec2e0968130593da5457a  task3_2
seed@instance-1:~/Desktop/Lab 3/Task 3$

```

- We can observe that the hash values have similar results.
- We can note that there are some differences of output even when the hash value is same.

2.4 Task 4: Making the Two Programs Behave Differently

Code file –

```
*code.c  
~/Desktop/Lab_3/Task 4
```

```
1 #include <stdio.h>  
2 unsigned char a[200] = {  
3     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,  
4 };  
5  
6 unsigned char b[200] = {  
7     0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,  
8 };  
9 int main()  
10 {  
11     int i;  
12     for (i=0; i<200; i++){  
13         if(a[i] != b[i])  
14         {  
15             break;  
16         }  
17     }  
18     if(i==200)  
19     {  
20         printf("%s", "run benign code");  
21     }  
22     else  
23     {  
24         printf("%s", "WARNING: run malicious code");  
25     }  
26     printf("\n");  
27 }
```

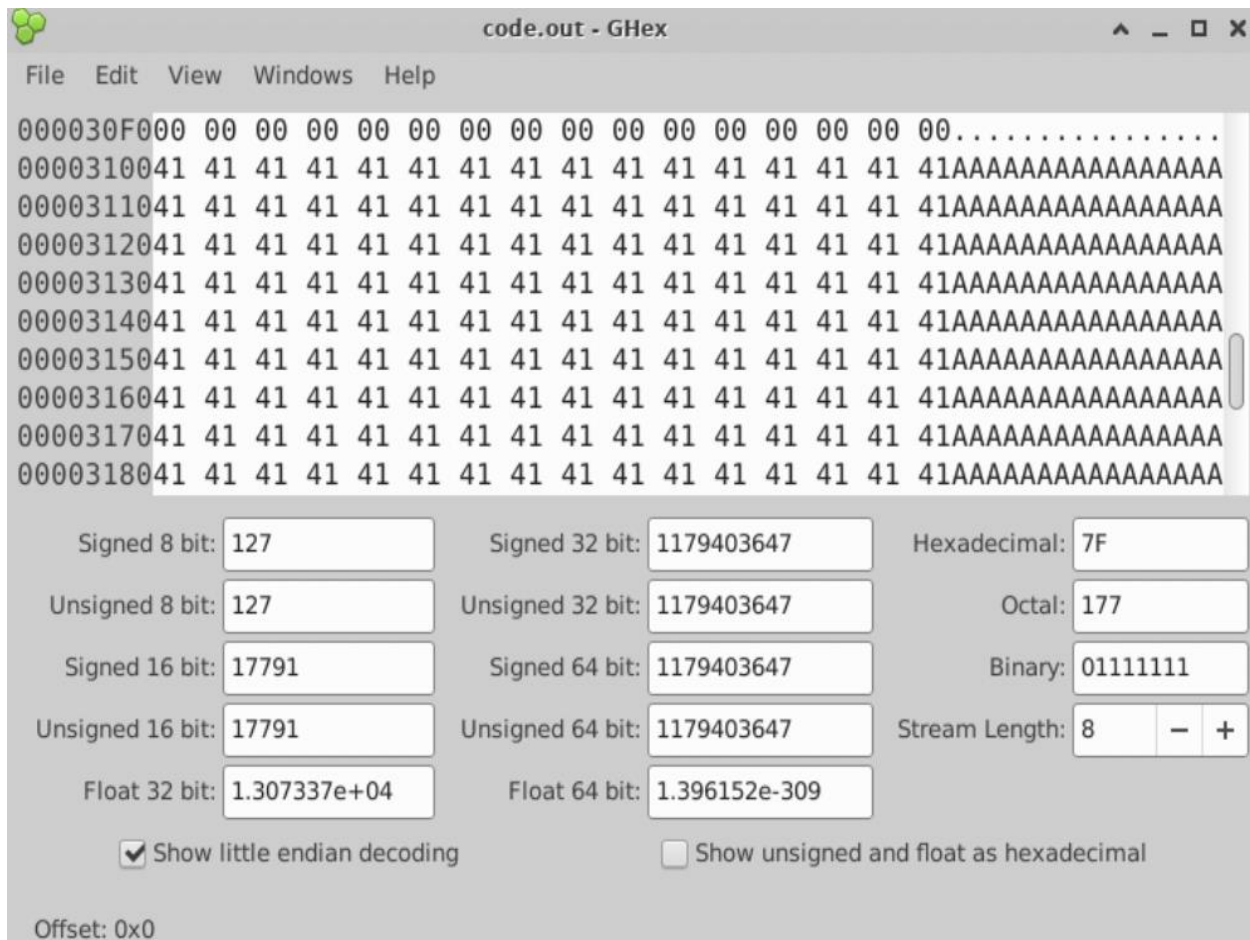
- `#include <stdio.h>`: Includes the standard input-output library for functions like `printf`.
- `unsigned char a[200] = { ... };`: Declares an array `a` of unsigned characters with a size of 200 elements and initializes it with hexadecimal values.
- `unsigned char b[200] = { ... };`: Declares another array `b` of unsigned characters with the same size and initializes it with the same hexadecimal values as array `a`.
- `int main()`: Entry point of the program, where execution begins.
- `int i;`: Declares an integer variable `i` without initializing it.
- `for(i=0; i<200; i++)`: Starts a loop that iterates from 0 to 199 (`<200` means less than 200) and increments `i` by 1 each time.
- `if(a[i] != b[i])`: Checks if the elements at index `i` in arrays `a` and `b` are not equal.
- `break;`: Breaks out of the loop immediately if a difference is found between corresponding elements of arrays `a` and `b`.
- `if(i == 200)`: Checks if the loop completed all 200 iterations without breaking early.

This program essentially compares the elements of arrays a and b byte by byte. If all elements match, it prints "run benign code"; otherwise, it prints "WARNING: run malicious code"

Compilation –

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ gcc code.c -o code.o
seed@instance-1:~/Desktop/Lab 3/Task 4$ ./code.o
run benign code
```

Now we observe the beginning of first array –



The screenshot shows the GHex hex editor window titled "code.out - GHex". The main pane displays the following hex data:

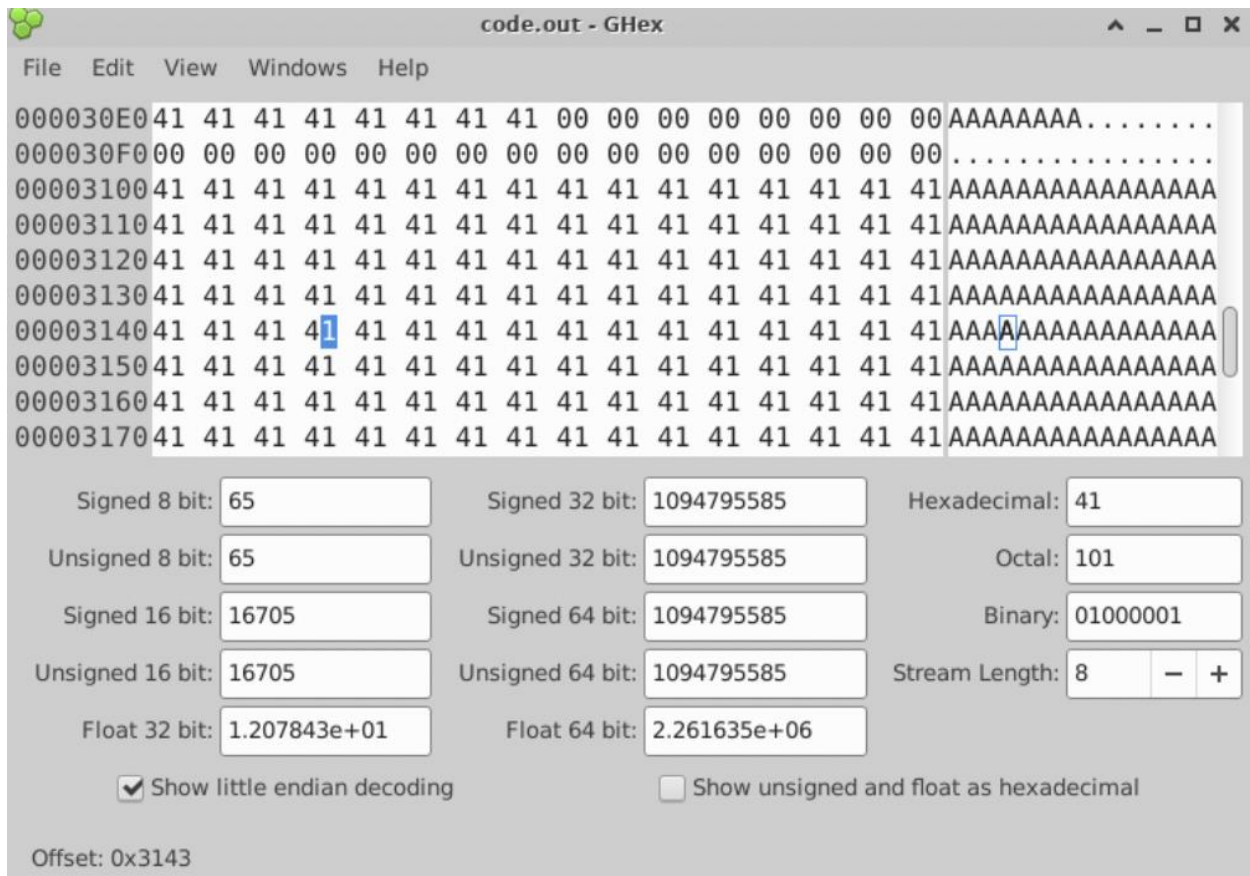
Address	Hex Data
000030F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00003100	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003110	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003120	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003130	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003140	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003150	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003160	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003170	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA
00003180	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41AAAAAAAAAAAAAAAA

The right pane shows the following decodings for the selected data:

Decoding	Value
Signed 8 bit:	127
Unsigned 8 bit:	127
Signed 16 bit:	17791
Unsigned 16 bit:	17791
Float 32 bit:	1.307337e+04
Signed 32 bit:	1179403647
Unsigned 32 bit:	1179403647
Signed 64 bit:	1179403647
Unsigned 64 bit:	1179403647
Float 64 bit:	1.396152e-309
Hexadecimal:	7F
Octal:	177
Binary:	01111111
Stream Length:	8

At the bottom, there are two checkboxes: ☒ Show little endian decoding and ☐ Show unsigned and float as hexadecimal. The offset is 0x0.

Next, we observe beginning of second array –



The first array is present at 4160. Next, we create a prefix called prefix_task4 and proceed to use the collision tool which in-turn generated 2 output files output1_task4.bin and output2_task4.bin.

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ head -c 4160 code.o > prefix_task4
seed@instance-1:~/Desktop/Lab 3/Task 4$ md5collgen -p prefix_task4 -o out1_task4
.bin out2_task4.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1_task4.bin' and 'out2_task4.bin'
Using prefixfile: 'prefix_task4'
Using initial value: accf25041503e3f9d0dced6b4b914c76

Generating first block: .....
Generating second block: S01.....
Running time: 22.0528 s
```

Next, we create p and q as follow:

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ tail -c 128 out1_task4.bin > p
seed@instance-1:~/Desktop/Lab 3/Task 4$ tail -c 128 out2_task4.bin > q
```

Next we create a suffix from program.o and use it create 2 different suffix (temp_suffix1, temp_suffix2)5c

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ tail -c +4289 code.o > suffix
seed@instance-1:~/Desktop/Lab 3/Task 4$ head -c 97 suffix > temp_suffix1
seed@instance-1:~/Desktop/Lab 3/Task 4$ tail -c +255 suffix > temp_suffix2
```

- `tail -c +4289 code.o > suffix`: This command extracts the contents of code.o starting from byte 4289 to the end of the file and saves it into a new file called suffix.
- `head -c 97 suffix > temp_suffix1`: This command takes the first 97 bytes from the suffix file and saves them into a new file called temp_suffix1.
- `tail -c +255 suffix > temp_suffix2`: This command skips the first 254 bytes of the suffix file and extracts the rest, starting from byte 255, and saves it into a new file called temp_suffix2.

Next, we concatenate prefix_task4, p, temp_suffix1, temp_suffix2 as follow to generate 2 executable files (Task4_1 and Task4_2)

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ cat prefix_task4 p temp_suffix1 p temp_s
uffix2 > task4_1
seed@instance-1:~/Desktop/Lab 3/Task 4$ cat prefix_task4 q temp_suffix1 q temp_s
uffix2 > task4_2
```

Next, we run each executable as follows:

- The command `chmod +x task4_1` is used to add executable permissions to a file named task4_1.
- This command allows the file to be executed as a program or script.

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ chmod +x task4_1
seed@instance-1:~/Desktop/Lab 3/Task 4$ ./task4_1
run benign code
```

- The command `chmod +x task4_2` is used to add executable permissions to a file named task4_2.
- This command allows the file to be executed as a program or script.

```
seed@instance-1:~/Desktop/Lab 3/Task 4$ chmod +x task4_2
seed@instance-1:~/Desktop/Lab 3/Task 4$ ./task4_2
WARNING: run malicious code
```

We can see that task4_1 will always run benign instructions (benign code refers to code that is harmless and does not have malicious intent or behavior) while task4_2 will always execute malicious instructions.

Therefore, we can conclude that a successful launch of MD5 Collision Attack is done.