

## Lab 3 - MD5 Collision Attack Lab

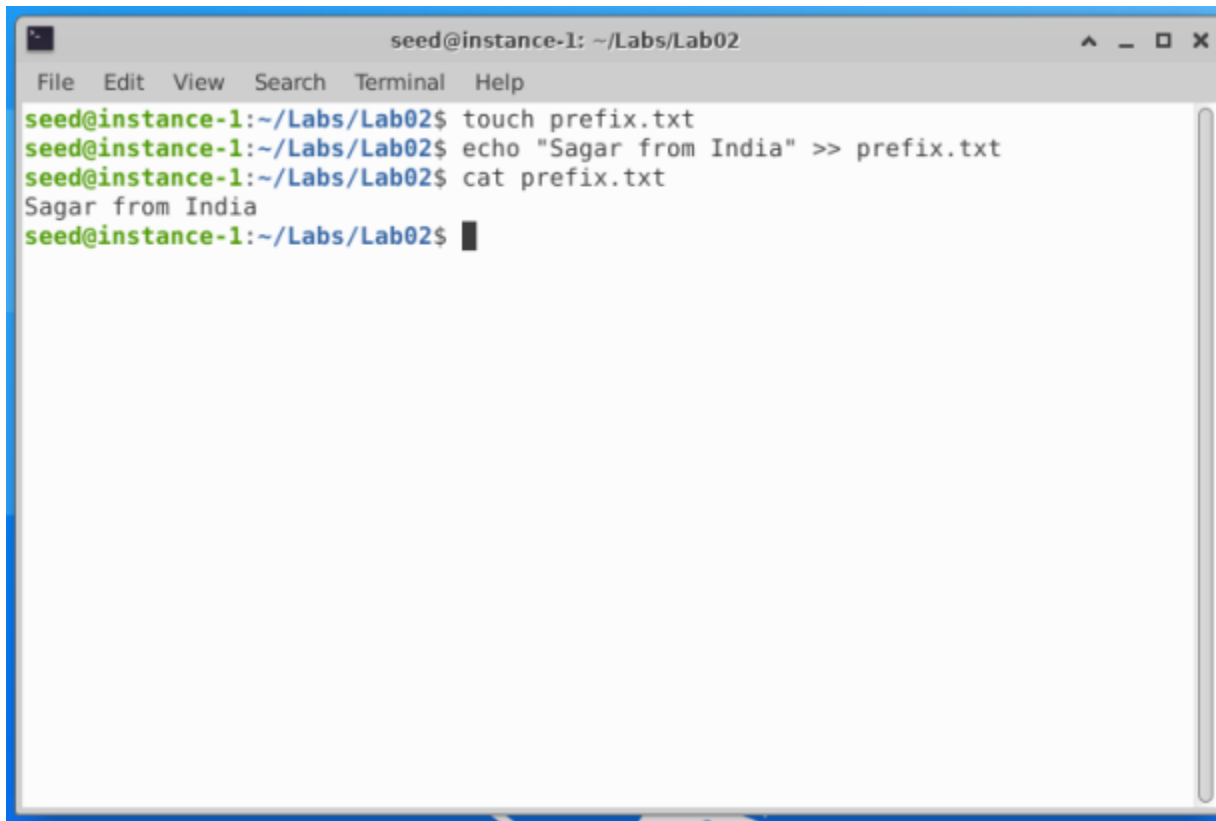
**MD5 algorithm**-MD5 is a one way cryptographic hashing algorithm which accepts any length of data as input and produces the fixed 128 bit size of output or digest. It was initially designed for Digital signatures.

**Collision attack**:A collision attack is the one where we try to find the two hash functions which inputs the string and produces the same hash value or digest.

**MD5collagen**: This is the program which is used to generate different files having the same MD5 hash value or digest.

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

Step1: Creating of the prefix.txt file

A screenshot of a terminal window titled 'seed@instance-1: ~/Labs/Lab02'. The terminal shows a series of commands and their outputs: 'touch prefix.txt' is executed, followed by 'echo "Sagar from India" >> prefix.txt', and then 'cat prefix.txt' which outputs 'Sagar from India'. The prompt 'seed@instance-1:~/Labs/Lab02\$' is visible at the end of the last line.

```
seed@instance-1: ~/Labs/Lab02
File Edit View Search Terminal Help
seed@instance-1:~/Labs/Lab02$ touch prefix.txt
seed@instance-1:~/Labs/Lab02$ echo "Sagar from India" >> prefix.txt
seed@instance-1:~/Labs/Lab02$ cat prefix.txt
Sagar from India
seed@instance-1:~/Labs/Lab02$
```

Step2: Creating the two different files with same hash values using md5collgen command with same prefix and then using diff command to check if the files are distinct

A terminal window titled 'seed@instance-1: ~/Labs/Lab02' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

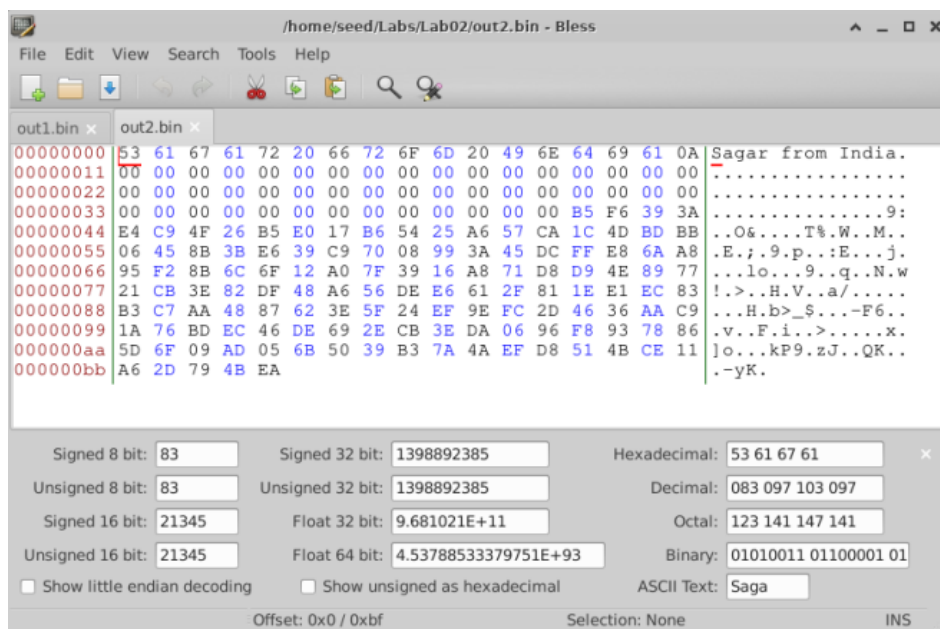
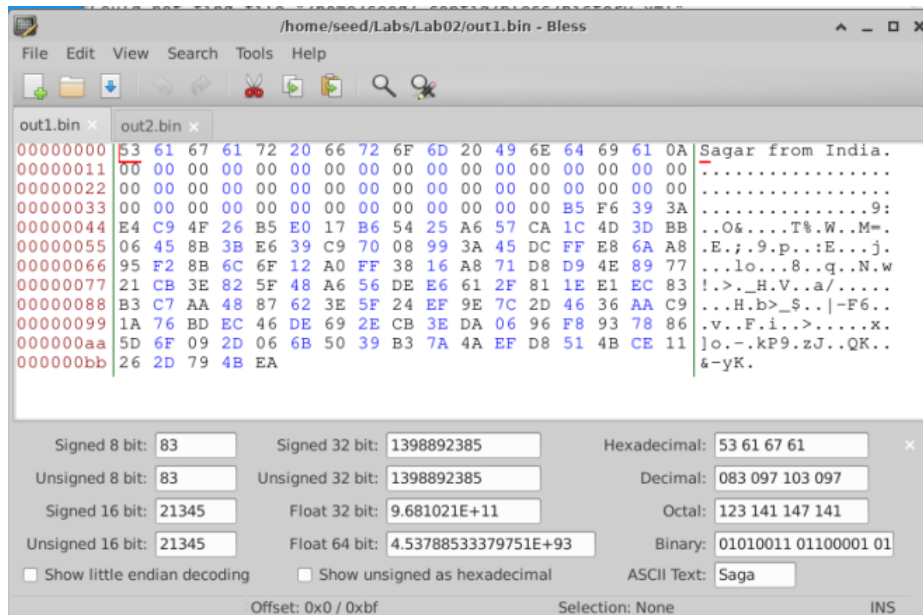
```
seed@instance-1:~/Labs/Lab02$ touch prefix.txt
seed@instance-1:~/Labs/Lab02$ echo "Sagar from India" >> prefix.txt
seed@instance-1:~/Labs/Lab02$ cat prefix.txt
Sagar from India
seed@instance-1:~/Labs/Lab02$ ./md5collgen -p prefix.txt -o out1.bin out2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'out1.bin' and 'out2.bin'
Using prefixfile: 'prefix.txt'
Using initial value: fc33df27e87b3b3e855d2a6939fe376e

Generating first block: ..
Generating second block: S01...
Running time: 2.98932 s
seed@instance-1:~/Labs/Lab02$ diff out
out1.bin out2.bin
seed@instance-1:~/Labs/Lab02$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
seed@instance-1:~/Labs/Lab02$
```

As shown above the generated 2 binary files differ.

Files opened in bless as shown below:



Using md5sum to show the hash values of the both the output files:

```
Could not find file "/home/seed/.config/bleess/history.xml"
seed@instance-1:~/Labs/Lab02$ md5sum out1.bin
b95518e521bef5c26f7c7efeb86e2349 out1.bin
seed@instance-1:~/Labs/Lab02$ md5sum out2.bin
b95518e521bef5c26f7c7efeb86e2349 out2.bin
seed@instance-1:~/Labs/Lab02$
```

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

Step 1).I created two prefix texts one is prefix1.txt(contains text as “Sa”) as shown below:

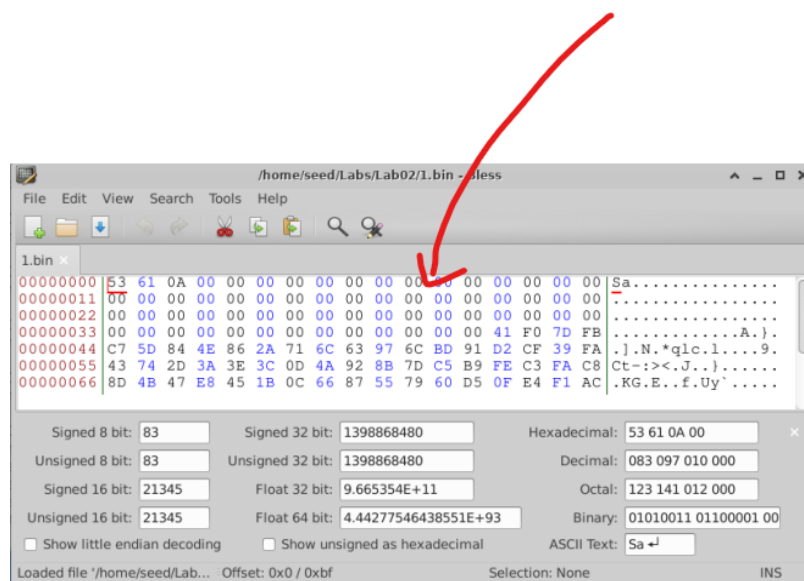
```
seed@instance-1:~/Labs/Lab02$ touch prefix1.txt
seed@instance-1:~/Labs/Lab02$ echo "Sa" >> prefix1.txt
seed@instance-1:~/Labs/Lab02$ cat prefix1.txt
Sa
```

Step 2): using md5collgen to generate the bin files

```
seed@instance-1:~/Labs/Lab02$ ./md5collgen -p prefix1.txt -o 1.bin 2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

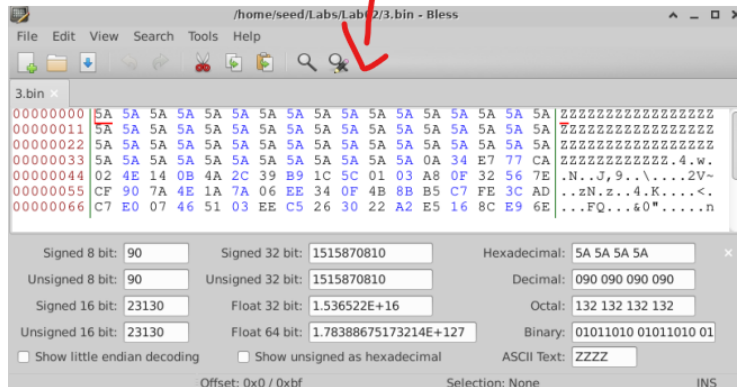
Using output filenames: '1.bin' and '2.bin'
Using prefixfile: 'prefix1.txt'
Using initial value: 71f39ed2bd8e84d2b546c999d2ca6cb9
```

Padding was done as the md5 processes for block size of 64 bytes at a time. So, zeros were padded as shown below:





Step2). Using md5collgen to produce two files



No Padding is observed in this case.

**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.**

```
seed@instance-1:~/Labs/Lab02$ xxd 3.bin > 3.txt
seed@instance-1:~/Labs/Lab02$ xxd 4.bin > 4.txt
seed@instance-1:~/Labs/Lab02$ diff 3.txt 4.txt
6,8c6,8
< 00000050: a80f 3256 7ecf 907a 4e1a 7a06 ee34 0f4b  ..2V~...zN.z..4.K
< 00000060: 8bb5 c7fe 3cad c7e0 0746 5103 eec5 2630  ....<....FQ...&0
< 00000070: 22a2 e516 8ce9 6e00 36fa 23c9 6eb2 1dd3  ".....n.6.#.n...
---
> 00000050: a80f 32d6 7ecf 907a 4e1a 7a06 ee34 0f4b  ..2.~...zN.z..4.K
> 00000060: 8bb5 c7fe 3cad c7e0 0746 5103 ee45 2730  ....<....FQ..E'0
> 00000070: 22a2 e516 8ce9 6e00 36fa 2349 6eb2 1dd3  ".....n.6.#In...
10,12c10,12
< 00000090: 58f7 9952 147f 805c d99b 17c5 545a 591e  X..R...\\....TZY.
< 000000a0: 4b91 d6b5 69f6 9673 7588 a12f 7f4e 64a4  K...i..su../.Nd.
< 000000b0: 7d0d 6112 1af9 d1b4 4f25 a6a6 de25 5919  }.a....0%...%Y.
---
> 00000090: 58f7 99d2 147f 805c d99b 17c5 545a 591e  X.....\\....TZY.
> 000000a0: 4b91 d6b5 69f6 9673 7588 a12f 7fce 63a4  K...i..su../.c.
> 000000b0: 7d0d 6112 1af9 d1b4 4f25 a626 de25 5919  }.a....0%.&%Y.
seed@instance-1:~/Labs/Lab02$
```

```
seed@instance-1:~/Labs/Lab02$ cmp -lb 3.bin 4.bin
84 126 V      326 M-V
110 305 M-E   105 E
111  46 &      47 '
124 311 M-I   111 I
148 122 R      322 M-R
174 116 N      316 M-N
175 144 d      143 c
188 246 M-&    46 &
seed@instance-1:~/Labs/Lab02$
```

Not all the bytes are different but the bytes mentioned in the above screenshots only differ. There are 8 bytes that differ between two files.

## 2.2 Task 2: Understanding MD5's Property

We can append the same text to the files generated by the md5collgen which changes the MD5 hash but both the new hashes will be the same. This is because the hash function has an internal state and processes the message in fixed blocks by applying the function to the current state and block.

So when we apply the MD5 hash for both the files will be the same as the function will be applied on the internal state and current block.

Let's create a file p.txt and then do md5collgen on it to produce 5.bin and 6.bin

```
seed@instance-1: ~/Labs/Lab02
File Edit View Search Terminal Help
seed@instance-1:~/Labs/Lab02$ echo "Intro to info sec" >> p.txt
seed@instance-1:~/Labs/Lab02$ ./md5collgen -p p.txt -o 5.bin 6.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: '5.bin' and '6.bin'
Using prefixfile: 'p.txt'
Using initial value: c19019473fdc73088f27a5393faf77de

Generating first block: .
Generating second block: S01.....
Running time: 1.71286 s
seed@instance-1:~/Labs/Lab02$ md5sum 5.bin 6.bin
2ab7c1e0e1b7a5efa98a3805e5156253 5.bin
2ab7c1e0e1b7a5efa98a3805e5156253 6.bin
seed@instance-1:~/Labs/Lab02$
```

Now let's add same string "Sagar" to both the hashes i.e. 5.bin and 6.bin

```
seed@instance-1:~/Labs/Lab02$ echo "Prof" >> 5.bin
seed@instance-1:~/Labs/Lab02$ echo "Prof" >> 6.bin
seed@instance-1:~/Labs/Lab02$ md5sum 5.bin 6.bin
52ac6d201d448eb8ffba0a490e8a2718  5.bin
52ac6d201d448eb8ffba0a490e8a2718  6.bin
seed@instance-1:~/Labs/Lab02$
```

Both the hashes change but remain identical.

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

I have chosen 0x41 as the content and I have put the 0x41 into the array xyz for 200 times. Below is the screenshot of my program:

[illegible]

Then I compiled the code using the gcc command as below:

```
seed@instance-1:~/Labs/Lab02$ gcc task3.c -o task3.out
seed@instance-1:~/Labs/Lab02$ gcc task3.c -o task3.o
seed@instance-1:~/Labs/Lab02$ ls *.o
task3.o
```

Now I created the prefix by running the command:

“Head -c 4224 task3.o > prefix”



```
seed@instance-1:~/Labs/Lab02$ head -c 4224 task3.o >prefix
seed@instance-1:~/Labs/Lab02$ ./md5collgen -p prefix -o task3_1.bin task3_2.bin
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'task3_1.bin' and 'task3_2.bin'
Using prefixfile: 'prefix'
Using initial value: bff76ee2a35765e4624bb4dc6c4cfe4b

Generating first block: .....
Generating second block: S10.
Running time: 7.7805 s
```

```
seed@instance-1:~/Labs/Lab02$ tail -c +4352 task3.o > suffix
seed@instance-1:~/Labs/Lab02$ tail -c 128 task3_1.bin > p
seed@instance-1:~/Labs/Lab02$ tail -c 128 task3_2.bin > q
```

```
seed@instance-1:~/Labs/Lab02$ cat prefix p suffix > task3_a
seed@instance-1:~/Labs/Lab02$ cat prefix q suffix > task3_b
seed@instance-1:~/Labs/Lab02$ md5sum task3_a task3_b
9d1a72f96d7d1f28a72acee54d6011d4 task3_a
9d1a72f96d7d1f28a72acee54d6011d4 task3_b
```

[illegible]

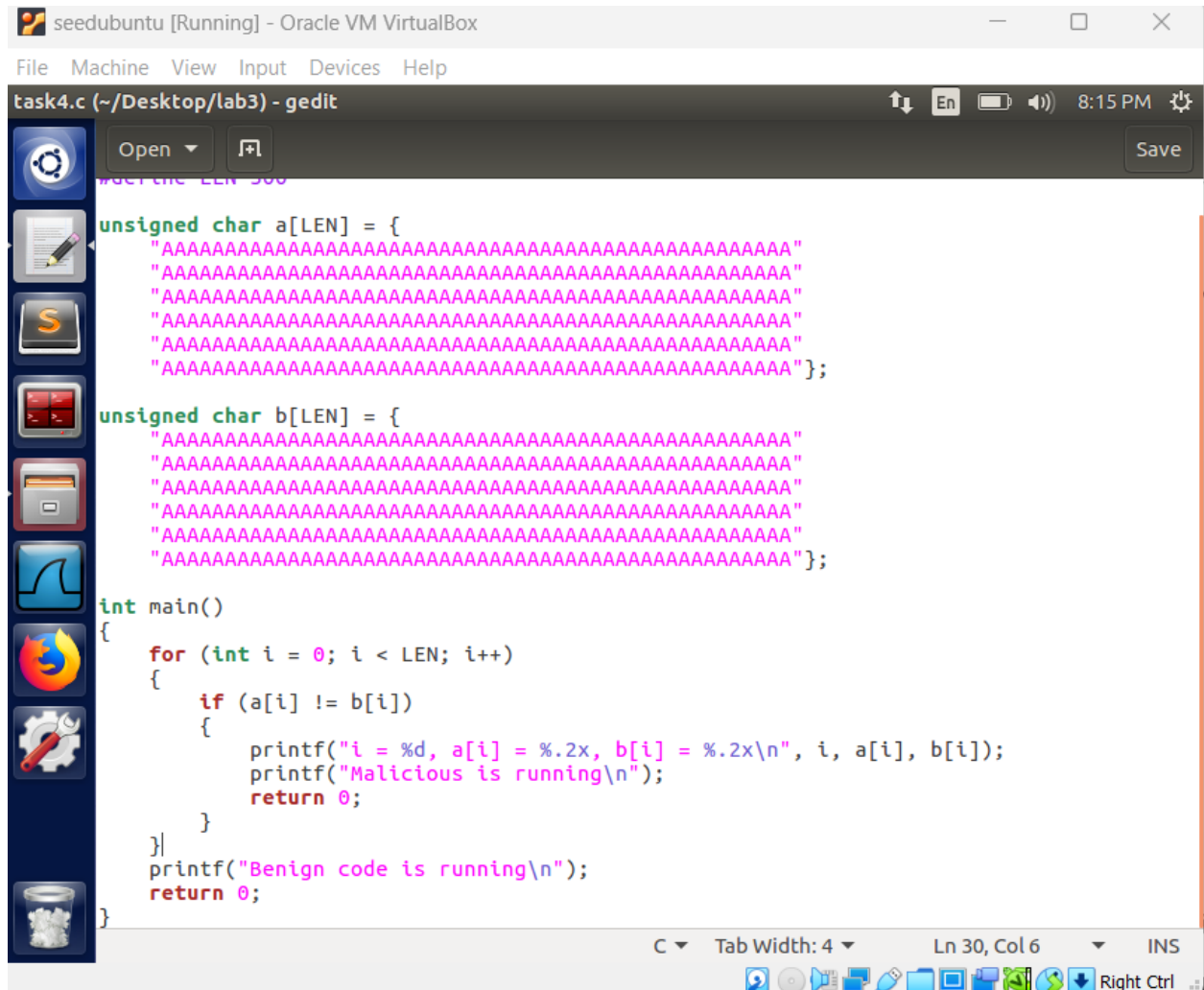
[illegible]

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

**Note:**For this task I have made use of Virtual Box as the cloud VM was not working as expected.

Below is the screenshot of my program where if  $a[i]=b[i]$  then benign code runs else malicious code runs.

Then compiled the above program using gcc and then ran it. Since all the contents of both arrays are the same, the output is Benign code is running as shown below:



```
task4.c (~/Desktop/lab3) - gedit
#define LEN 500

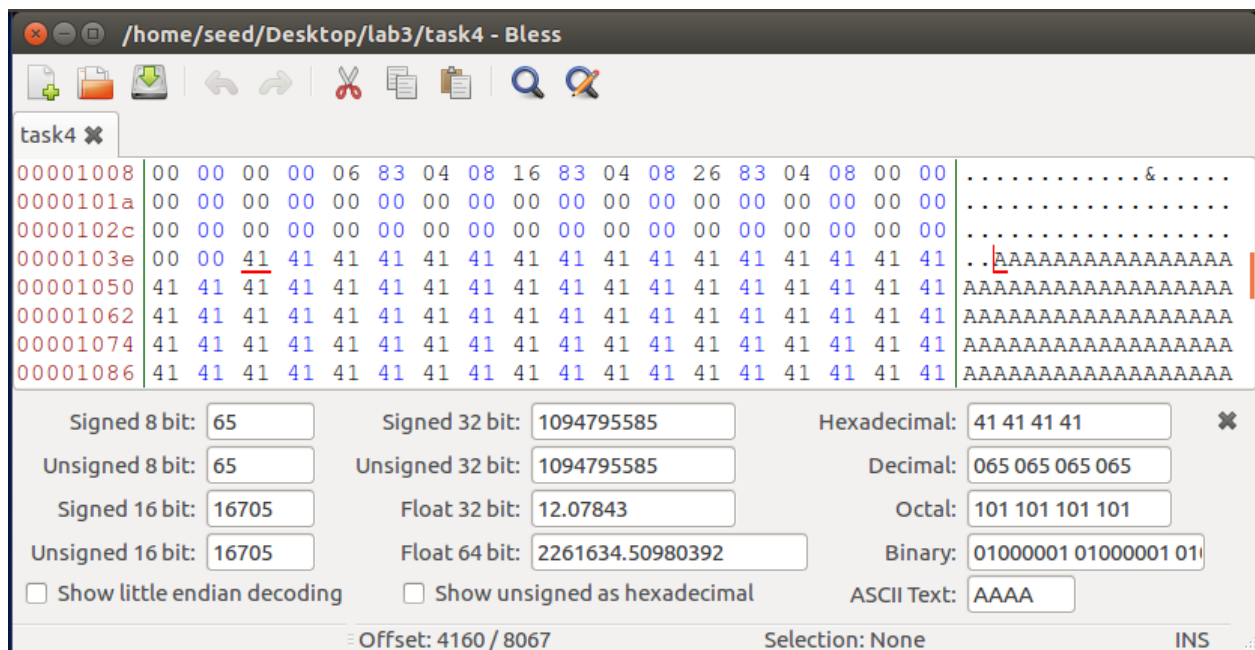
unsigned char a[LEN] = {
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"};

unsigned char b[LEN] = {
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
    "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"};

int main()
{
    for (int i = 0; i < LEN; i++)
    {
        if (a[i] != b[i])
        {
            printf("i = %d, a[i] = %.2x, b[i] = %.2x\n", i, a[i], b[i]);
            printf("Malicious is running\n");
            return 0;
        }
    }
    printf("Benign code is running\n");
    return 0;
}
```

After we compiled the above code since both the arrays has same content, the output will be Benign code is running as shown below

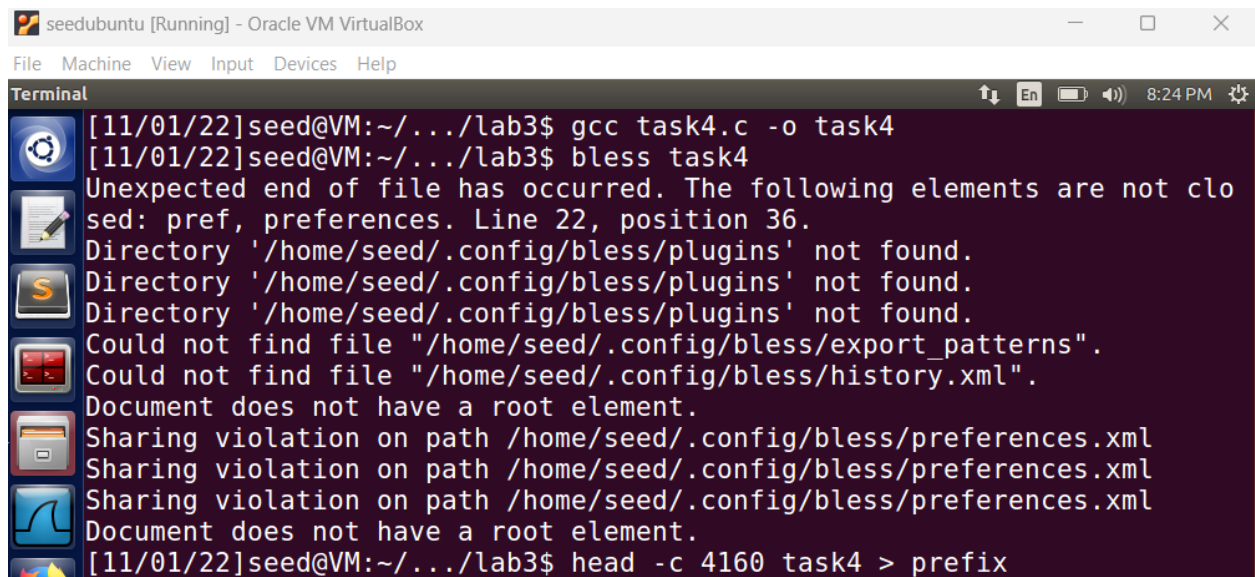
```
[11/01/22]seed@VM:~/.../lab3$ ./task4
Benign code is running
[11/01/22]seed@VM:~/.../lab3$
```



The A array is starting at 4160 and the B array is starting at 4480.

Now lets store whatever is present till the start of A array as prefix by using the command

Head -c 4160 task4 > prefix



Since we got the prefix now let's use md5collgen to generate out1 and out2

```
Document does not have a root element.
[11/01/22]seed@VM:~/.../lab3$ head -c 192 suffix > pre-suffix
[11/01/22]seed@VM:~/.../lab3$ 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: 81bbd40b2b239a38da42cc5707c570f4

Generating first block: .....
Generating second block: S00.....
Running time: 23.1155 s
[11/01/22]seed@VM:~/.../lab3$
```

Let's generate P and Q values :

```
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out1 > P
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out2 > Q
[11/01/22]seed@VM:~/.../lab3$ md5sum out1 out2
8da1c14a6aa0bfb50blac4c78b432ba7 out1
8da1c14a6aa0bfb50blac4c78b432ba7 out2
[11/01/22]seed@VM:~/.../lab3$
```

And then leaving a gap of 128 bytes and then the rest of the task4(output file) is stored as a suffix for now.

Command used: tail -c +4288 task4 > suffix

```
[11/01/22]seed@VM:~/.../lab3$ tail -c +4288 task4 > suffix
[11/01/22]seed@VM:~/.../lab3$
```

Now the rough picture:

Prefix (From 0 to 4160)	128 bytes to store A array	Suffix (Bytes from 4288)
-------------------------	----------------------------	--------------------------

Now we are dividing the suffix into two i.e. pre-suffix and post-suffix

The B array starts from 193 so the pre-suffix will contain everything till 193 and to do this we use the command : head -c 192 suffix > pre-suffix

```
Document does not have a root element.
[11/01/22]seed@VM:~/.../lab3$ head -c 192 suffix > pre-suffix
[11/01/22]seed@VM:~/.../lab3$
```

In the post suffix we are storing the remaining part after 320 (192+128=320) of the suffix.

Now we have :

out1(prefix+P)	pre-suffix	P	post-suffix
out2(prefix+Q)	pre-suffix	P	post-suffix

Let's calculate both the combination i.e. one for benign and other should be malicious.

Task4\_1 ----> cat out1 pre-suffix P post-suffix >task4\_1

Task4\_2 ----> cat out2 pre-suffix P post-suffix >task4\_2



```
seedubuntu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Terminal 8:46 PM
Generating first block: .....
Generating second block: S00.....
Running time: 23.1155 s
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out1 > P
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out2 > Q
[11/01/22]seed@VM:~/.../lab3$ md5sum P Q
55acf1f4664307fb1d3896e3a65e9d2e P
96f24650b66405f20257361a2a40ad0b Q
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out1 > P
[11/01/22]seed@VM:~/.../lab3$ tail -c 128 out2 > Q
[11/01/22]seed@VM:~/.../lab3$ md5sum out1 out2
8dalcl4a6aa0bfb50blac4c78b432ba7 out1
8dalcl4a6aa0bfb50blac4c78b432ba7 out2
[11/01/22]seed@VM:~/.../lab3$ tail -c +320 suffix > post-suffix
[11/01/22]seed@VM:~/.../lab3$ cat out1 pre-suffix P post-suffix > task4_1
[11/01/22]seed@VM:~/.../lab3$ cat out2 pre-suffix P post-suffix > task4_2
[11/01/22]seed@VM:~/.../lab3$ chmod +x task4_1
[11/01/22]seed@VM:~/.../lab3$ chmod +x task4_2
[11/01/22]seed@VM:~/.../lab3$ ./task4_1
Benign code is running
[11/01/22]seed@VM:~/.../lab3$ ./task4_2
i = 19, a[i] = 25, b[i] = a5
Malicious is running
[11/01/22]seed@VM:~/.../lab3$
```

