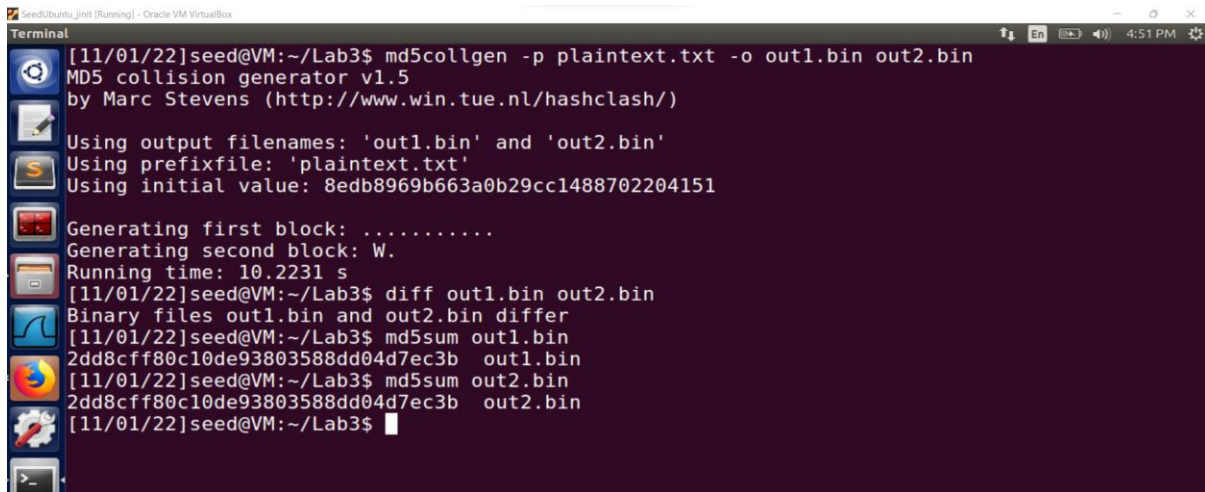


Introduction to Information Security

CS458 - Fall 2022

Lab 3 – MD5 Collision Attack Lab

Task 1



```

[11/01/22]seed@VM:~/Lab3$ md5collgen -p plaintext.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: 'plaintext.txt'
Using initial value: 8edb8969b663a0b29cc1488702204151

Generating first block: .....
Generating second block: W.
Running time: 10.2231 s
[11/01/22]seed@VM:~/Lab3$ diff out1.bin out2.bin
Binary files out1.bin and out2.bin differ
[11/01/22]seed@VM:~/Lab3$ md5sum out1.bin
2dd8cff80c10de93803588dd04d7ec3b  out1.bin
[11/01/22]seed@VM:~/Lab3$ md5sum out2.bin
2dd8cff80c10de93803588dd04d7ec3b  out2.bin
[11/01/22]seed@VM:~/Lab3$

```

Figure 1.1 Generating two files with same md5 hash values

- Created two files with same md5 hash values using 'md5collgen' program for the file 'plaintext.txt'.
- We can observe it using 'md5sum' as shown in above screenshot.

Question 1

- If the length of your prefix file is not multiple of 64 , what is going to happen?
 - The prefix file 'plaintext.txt' has 1129 bytes. According to algorithm, the input is treated as 64 bytes blocks. So to make it divisible by 64 algorithm does padding. So in our case we need 23 bytes padding to make it divisible by 64.
 - It appends **'0' bits for padding**.
 - We can observe in Figure 1.2 that 23 bytes (each '0' (underlined)) is appended to text to make it divisible by 64.
 - So file contents: - text + padding + P (data generated by md5collgen)

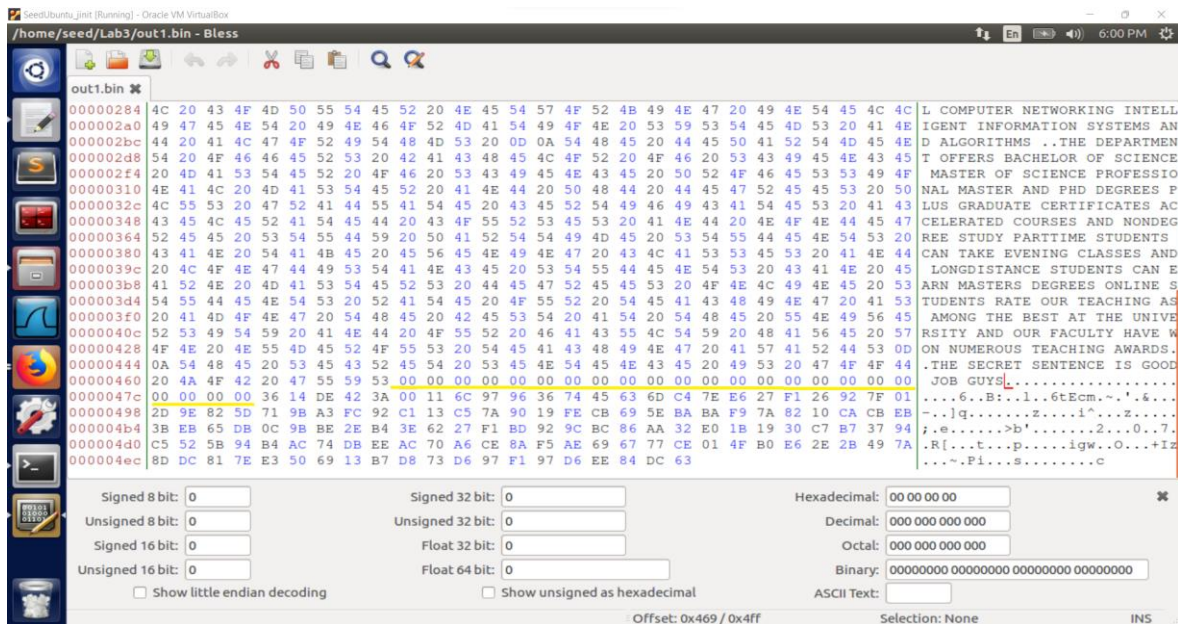


Figure 1.2 Padding bits in out1.bin (Same goes for out2.bin)

Question 2

- Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.
 - In figure 1.3 we can see the size of bin files becomes 192 bytes.

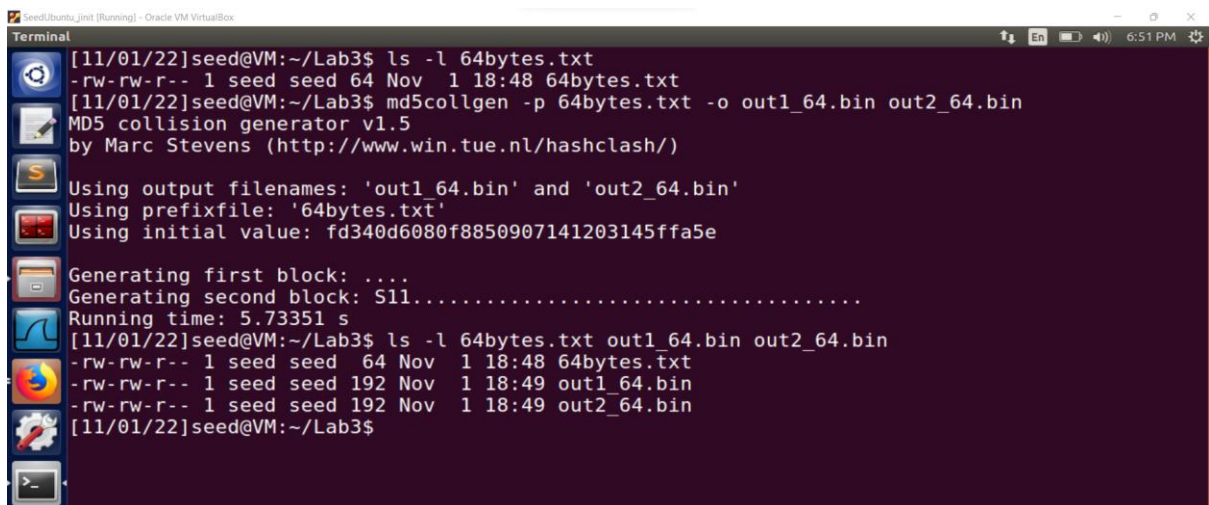


Figure 1.3 generating md5 hash values for file having size 64 bytes

- From figure 1.4 we can see that **no padding of 0's are there**, only 64 bytes of file contents + P (128 bytes) (data generated by md5collgen)

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.
 - They are not all different. **Some bytes have differences.** Differences in bytes are circled with red colour in figure 1.4
 - To find difference I have used 'diff' command.

```

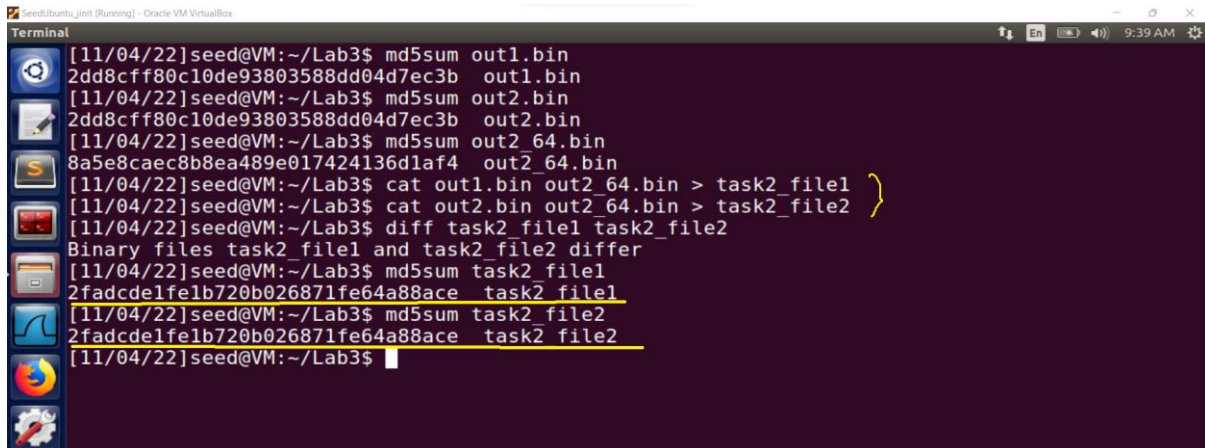
[11/01/22]seed@VM:~/Lab3$ hexdump out1.bin>hex1
[11/01/22]seed@VM:~/Lab3$ hexdump out2.bin>hex2
[11/01/22]seed@VM:~/Lab3$ diff hex1 hex2 --normal
74,76c74,76
< 0000490 e67e f127 9226 017f 9e2d 5d82 9b71 fca3
< 00004a0 c192 c513 907a fe19 69cb ba5e f9ba 827a
< 00004b0 ca10 ebc9 eb3b db65 9b0c 2ebe 3eb4 2762
---
> 0000490 e67e 7127 9226 017f 9e2d 5d82 9b71 fca3
> 00004a0 c192 c513 907a fe19 69cb ba5e 79ba 827b
> 00004b0 ca10 ebc9 eb3b db65 9b0c aebe 3eb4 2762
78,80c78,80
< 00004d0 52c5 945b acb4 db74 acee a670 8ace aef5
< 00004e0 6769 ce77 4f01 e6b0 2b2e 7a49 dc8d 7e81
< 00004f0 50e3 1369 d8b7 d673 f197 d697 84ee 63dc
---
> 00004d0 52c5 145b acb4 db74 acee a670 8ace aef5
> 00004e0 6769 ce77 4f01 e6b0 2b2e 7a49 5c8d 7e81
> 00004f0 50e3 1369 d8b7 d673 f197 5697 84ee 63dc
[11/01/22]seed@VM:~/Lab3$

```

Figure 1.4 Differences in generated bytes by md5collgen

Task 2

- For this task let's take two bin files created in first task namely 'out1.bin' and 'out2.bin' as they have same md5 hash values. As seen in Figure 2.1, concatenation of 'out2_64.bin' file to both files result in same hash values.



```

[11/04/22]seed@VM:~/Lab3$ md5sum out1.bin
2dd8cff80c10de93803588dd04d7ec3b  out1.bin
[11/04/22]seed@VM:~/Lab3$ md5sum out2.bin
2dd8cff80c10de93803588dd04d7ec3b  out2.bin
[11/04/22]seed@VM:~/Lab3$ md5sum out2_64.bin
8a5e8caec8b8ea489e017424136d1af4  out2_64.bin
[11/04/22]seed@VM:~/Lab3$ cat out1.bin out2_64.bin > task2_file1
[11/04/22]seed@VM:~/Lab3$ cat out2.bin out2_64.bin > task2_file2
[11/04/22]seed@VM:~/Lab3$ diff task2_file1 task2_file2
Binary files task2_file1 and task2_file2 differ
[11/04/22]seed@VM:~/Lab3$ md5sum task2_file1
2fadcdelfe1b720b026871fe64a88ace  task2_file1
[11/04/22]seed@VM:~/Lab3$ md5sum task2_file2
2fadcdelfe1b720b026871fe64a88ace  task2_file2
[11/04/22]seed@VM:~/Lab3$

```

Figure 2.1 comparing md5 value of two files

- The MD5 algorithm uses compression function to produce Intermediate Hash value. As md5 value of both files are same, there last IHV will be same. Now appending same file to both files will result in running compression function with same blocks. Hence, this will result in having same hash values after appending.
- So, from above screenshot we can summarise that $MD5(out1.bin) = MD5(out2.bin)$, i.e., the MD5 hashes of out1.bin and out2.bin are the same, then for any input out2_64.bin, $MD5(out1.bin || out2_64.bin) = MD5(out2.bin || out2_64.bin)$

Task 3

The aim of this task is to generate two executable files such a way that its md5 hash values are same. The two files will do different task (in our case, generate different array contents). As we saw in task 2 that $\text{MD5}(\mathbf{M}) = \text{MD5}(\mathbf{N})$ then for any input \mathbf{T} , $\text{MD5}(\mathbf{M} \parallel \mathbf{T}) = \text{MD5}(\mathbf{N} \parallel \mathbf{T})$. We will use the same property in this task.

- First, we created the executable file of the given C program in lab. (output in Figure 3.1)

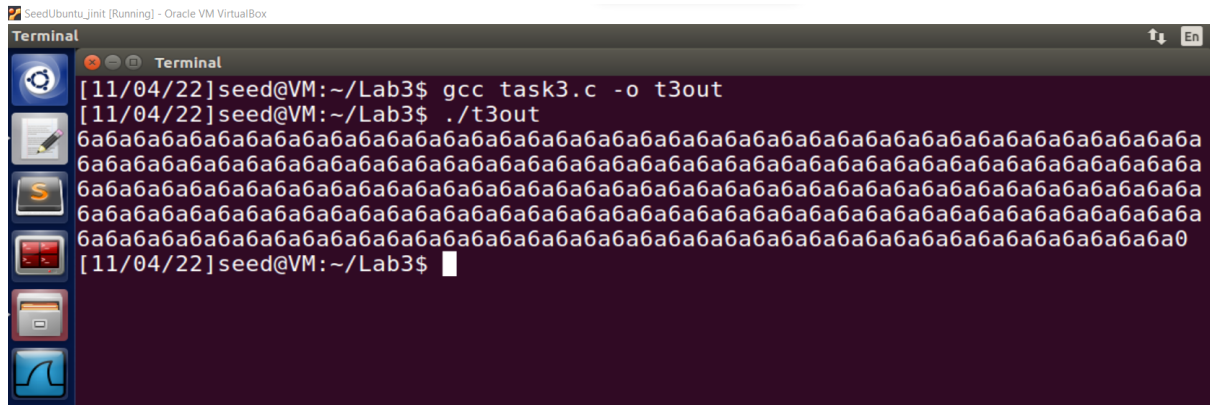


Figure 3.1 output of c program

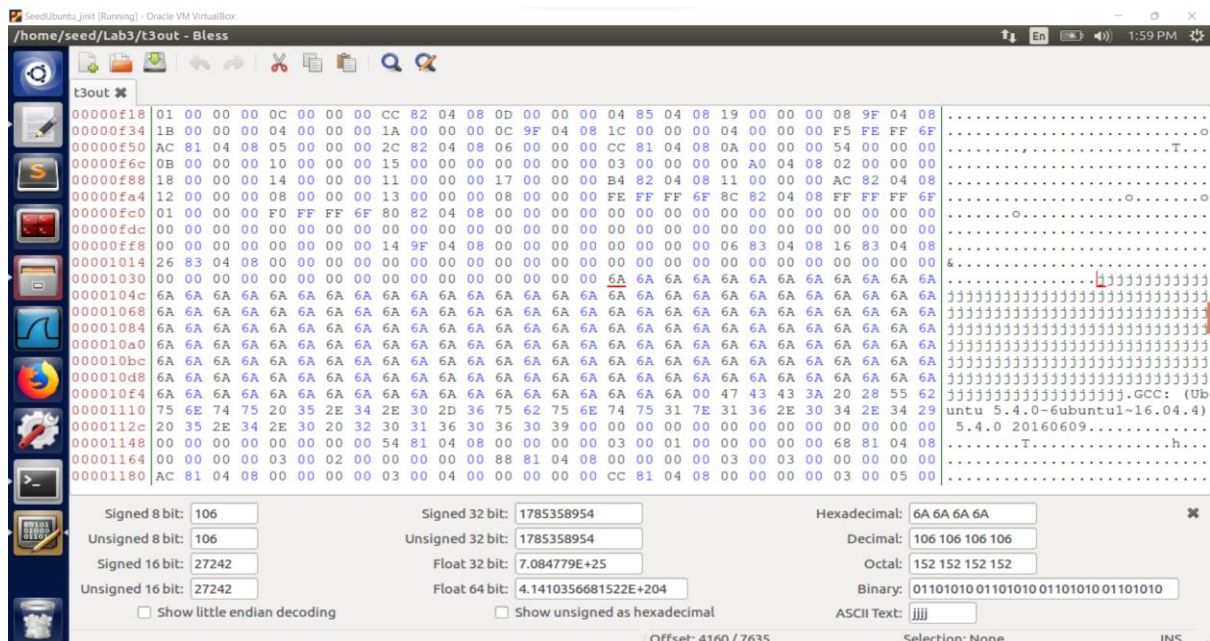
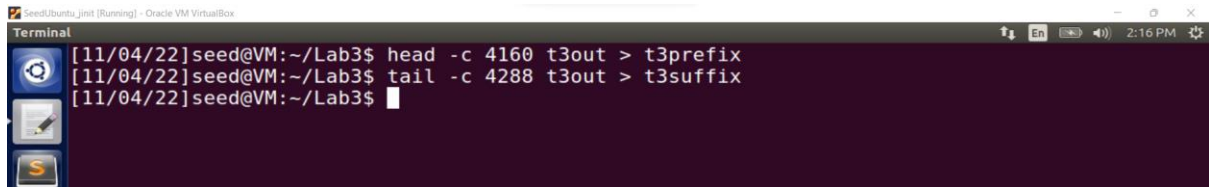


Figure 3.2 array characters start from offset 4160 in exe file

- To make prefix file, we will take all the contents till offset 4160 from t3out file. It is also a multiple of 64 hence requirements are satisfied. (prefix file name:- 't3prefix')
- To make suffix file, we will take all contents after offset $4160 + 128 = 4288$. (Suffix file name: - 't3suffix')
- By above two steps we have divide the executable file in three parts. (prefix+128 bytes region+suffix)
- Commands to make this file are shown in figure 3.3



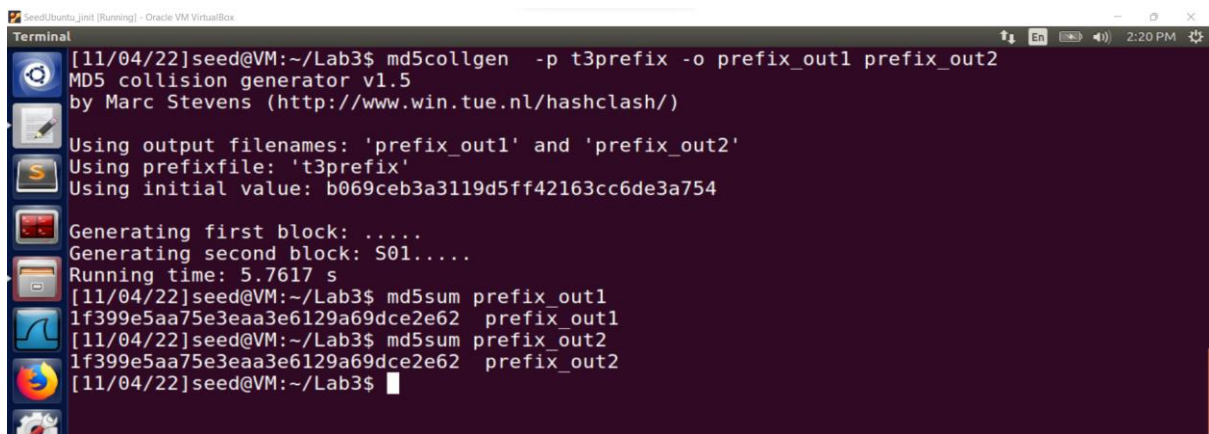
```

[11/04/22]seed@VM:~/Lab3$ head -c 4160 t3out > t3prefix
[11/04/22]seed@VM:~/Lab3$ tail -c 4288 t3out > t3suffix
[11/04/22]seed@VM:~/Lab3$

```

Figure 3.3 prefix and suffix file generation

- Now let us create two files of prefix file having same md5 value with md5collgen. So we will now have the following
 $\text{MD5}(\text{prefix} || P) = \text{MD5}(\text{prefix} || Q)$
 Command is shown in figure 3.4



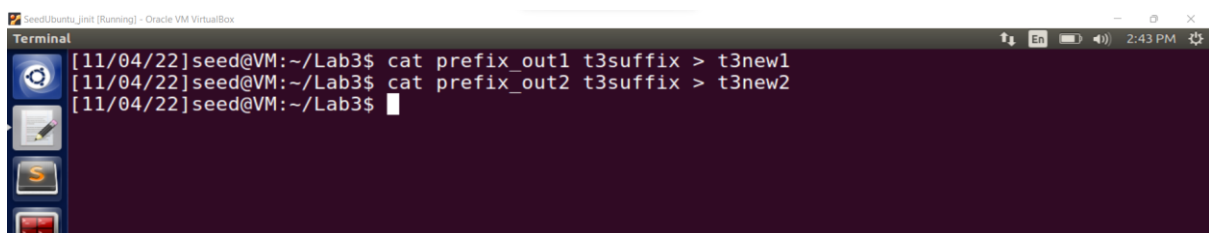
```

[11/04/22]seed@VM:~/Lab3$ md5collgen -p t3prefix -o prefix_out1 prefix_out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)
Using output filenames: 'prefix_out1' and 'prefix_out2'
Using prefixfile: 't3prefix'
Using initial value: b069ceb3a3119d5ff42163cc6de3a754
Generating first block: .....
Generating second block: S01.....
Running time: 5.7617 s
[11/04/22]seed@VM:~/Lab3$ md5sum prefix_out1
1f399e5aa75e3eaa3e6129a69dce2e62 prefix_out1
[11/04/22]seed@VM:~/Lab3$ md5sum prefix_out2
1f399e5aa75e3eaa3e6129a69dce2e62 prefix_out2
[11/04/22]seed@VM:~/Lab3$

```

Figure 3.4 md5 collision of prefix file

- As per the property of md5, we know that if we append same 'suffix' to the above generated files, 'prefix_out1' and 'prefix_out2', the resultant data will also have same hash value.
- Figure 3.5 shows commands which append 't3suffix' to both files.



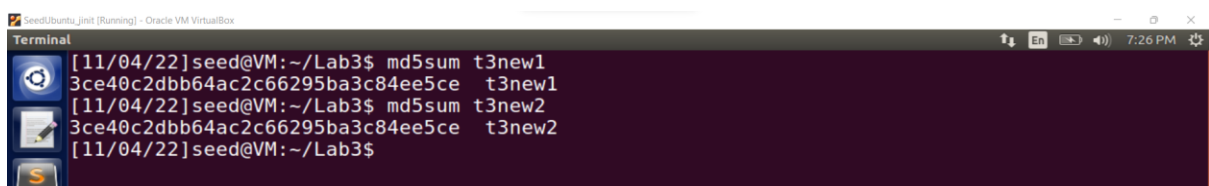
```

[11/04/22]seed@VM:~/Lab3$ cat prefix_out1 t3suffix > t3new1
[11/04/22]seed@VM:~/Lab3$ cat prefix_out2 t3suffix > t3new2
[11/04/22]seed@VM:~/Lab3$

```

Figure 3.5 appending suffix to prefix files

- From figure 3.6 you can see the new files have same md5 hash values.



```

[11/04/22]seed@VM:~/Lab3$ md5sum t3new1
3ce40c2dbb64ac2c66295ba3c84ee5ce t3new1
[11/04/22]seed@VM:~/Lab3$ md5sum t3new2
3ce40c2dbb64ac2c66295ba3c84ee5ce t3new2
[11/04/22]seed@VM:~/Lab3$

```

Figure 3.6 hash values of new files

- Now in order to see if both arrays have different values, we need to make 't3new1' and 't3new2' executable.

- In figure 3.7 we have given the necessary permission to both files.

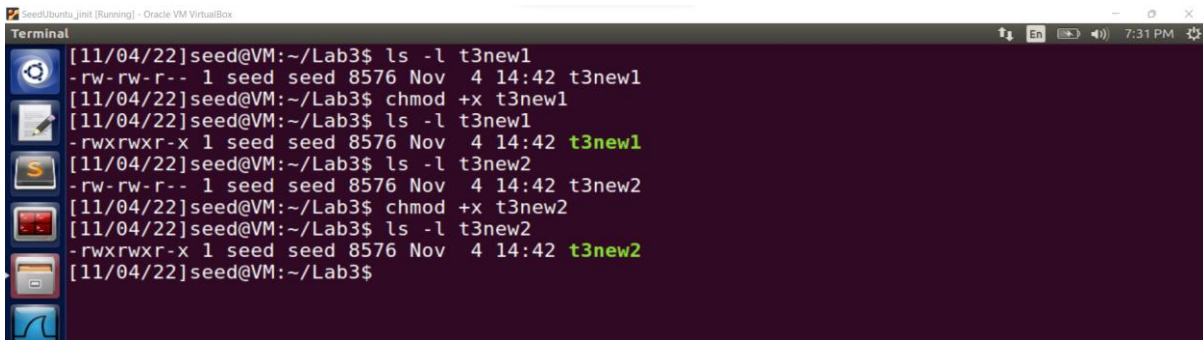


Figure 3.7 giving executable permission

- Now let's see the contents of array of 't3new1' and 't3new2' (modified files).

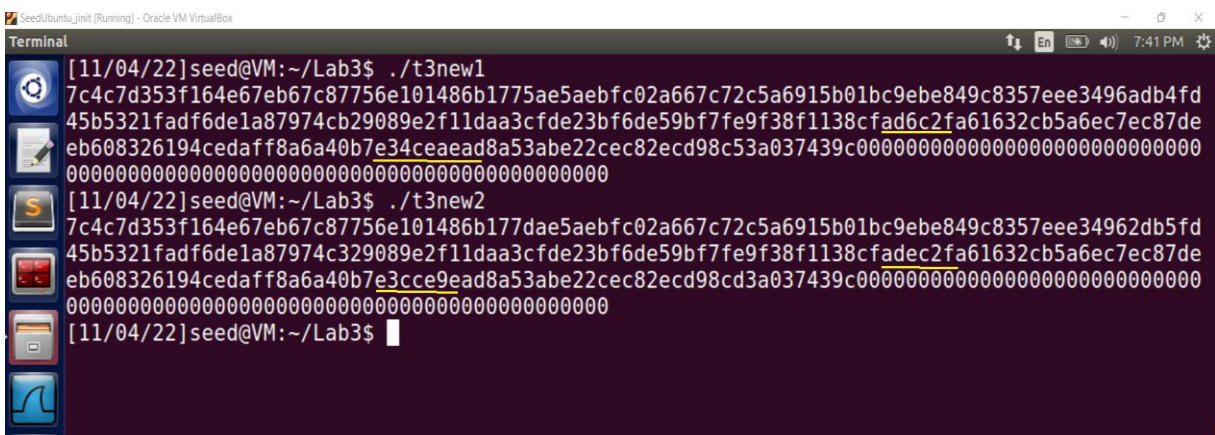


Figure 3.8 values of array of each file

- Both printed arrays have different values (also underlined it with yellow colour in figure 3.8)
- Let's check the md5 hash values of this files.

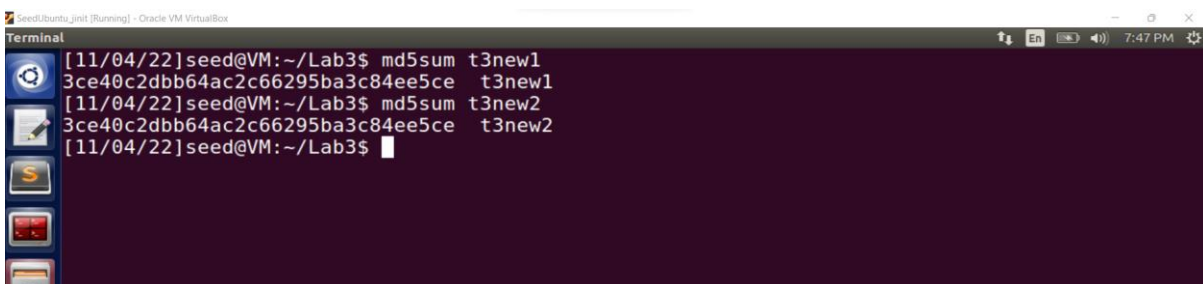


Figure 3.9 md5 hash values

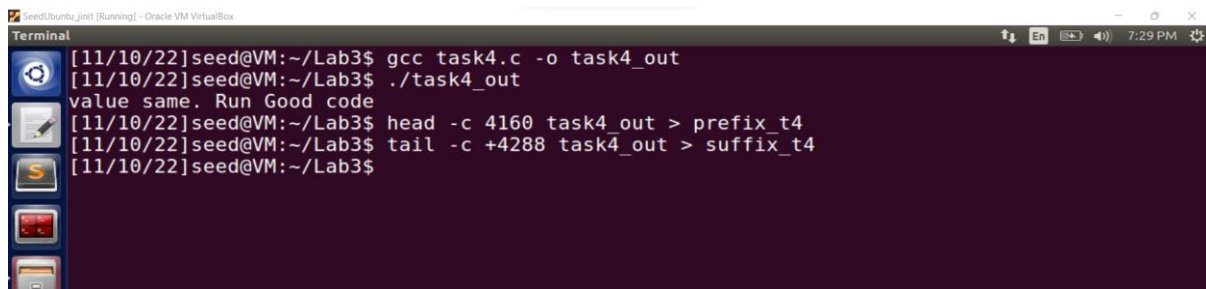
- As we can see from figure 3.9, both files have same hash.
- Hence, we have achieved generating two different executable files with same hash values.

Task 4

The main aim of this task is to create two programs with same md5 hash values but perform instructions. The one program performs good things while the other perform malicious things to harm the system.

So, in our case we will try to print statements based on array values. If both the arrays are same, we will print 'Run good code' and if they don't match, we will print 'Run malicious code'.

- First, we will do the same step as of task 3. We will divide the executable file in three parts. (prefix + 128 bytes region + suffix). Storing prefix and suffix in new files. (prefix_t4 and suffix_t4)



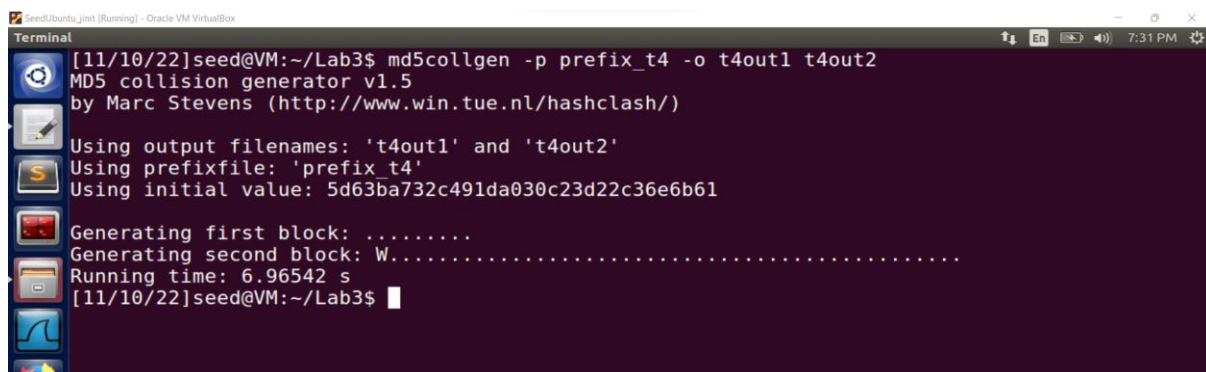
```

[11/10/22]seed@VM:~/Lab3$ gcc task4.c -o task4_out
[11/10/22]seed@VM:~/Lab3$ ./task4_out
value same. Run Good code
[11/10/22]seed@VM:~/Lab3$ head -c 4160 task4_out > prefix_t4
[11/10/22]seed@VM:~/Lab3$ tail -c +4288 task4_out > suffix_t4
[11/10/22]seed@VM:~/Lab3$

```

Figure 4.1 generating prefix and suffix file from executable file

- Now let us created two files of prefix file having same md5 value with md5collgen. So we will now have the following
 $\text{MD5}(\text{prefix} || P) = \text{MD5}(\text{prefix} || Q)$ (Figure 4.2)



```

[11/10/22]seed@VM:~/Lab3$ md5collgen -p prefix_t4 -o t4out1 t4out2
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

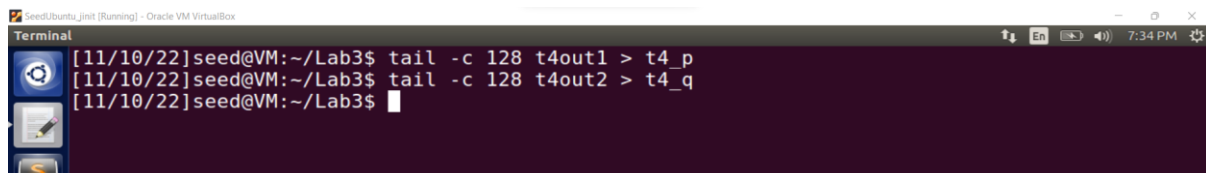
Using output filenames: 't4out1' and 't4out2'
Using prefixfile: 'prefix_t4'
Using initial value: 5d63ba732c491da030c23d22c36e6b61

Generating first block: .....
Generating second block: W.....
Running time: 6.96542 s
[11/10/22]seed@VM:~/Lab3$

```

Figure 4.2 md5collgen on prefix file

- Now taking P and Q values from t4out1 and t4out2 file and storing it into new files namely t4_p and t4_q. This will be used in later stage while creating two different programs. (Figure 4.3)



```

[11/10/22]seed@VM:~/Lab3$ tail -c 128 t4out1 > t4_p
[11/10/22]seed@VM:~/Lab3$ tail -c 128 t4out2 > t4_q
[11/10/22]seed@VM:~/Lab3$

```

Figure 4.3 storing P and Q values from bin files into new files

- Now we will separate array1 and array2 from suffix file into two different files so that we can use these files in appending while creating final files of same structure.

- From figure 4.4 we can see array2 starts from offset 97. So the contents till offset 96 will go in 'second_prefix' file and rest contents after offset 96+128 = 224 will go in 'second_suffix' file. (Commands in figure 4.5)

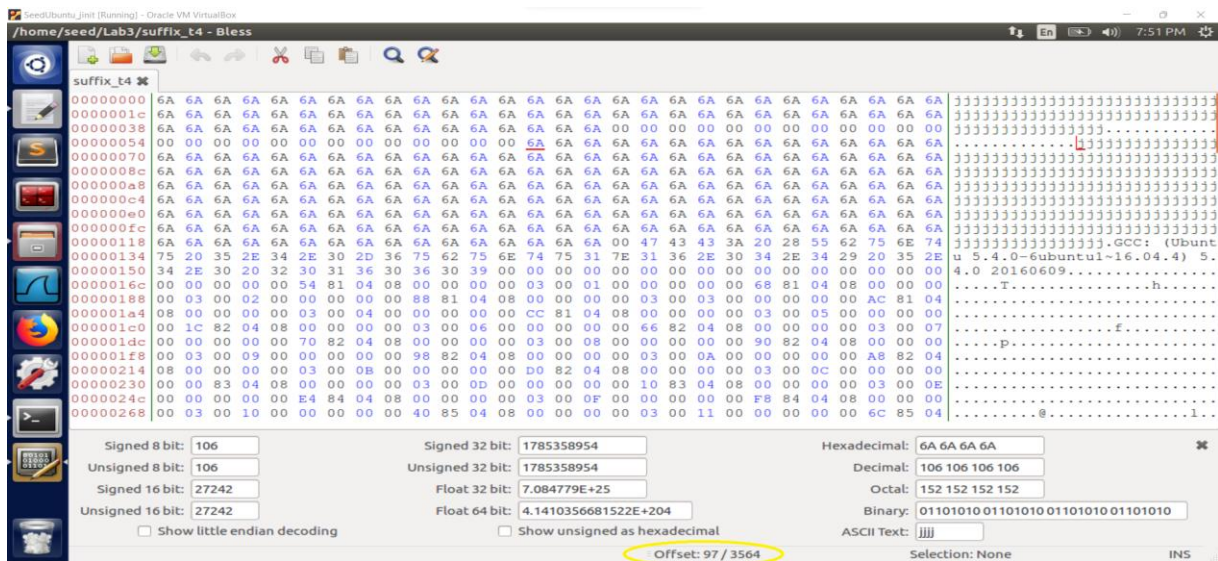


Figure 4.4 starting offset of array2 in suffix file (offset 97)

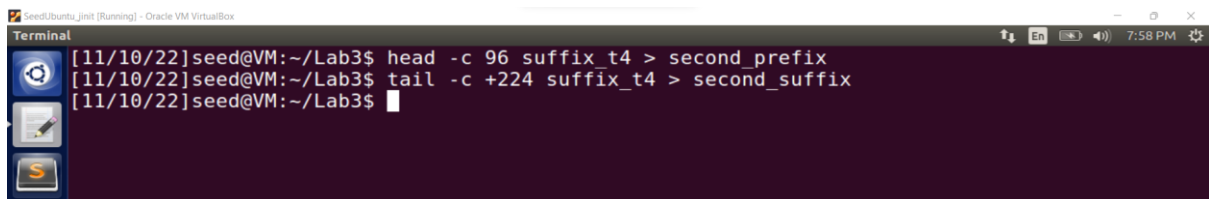
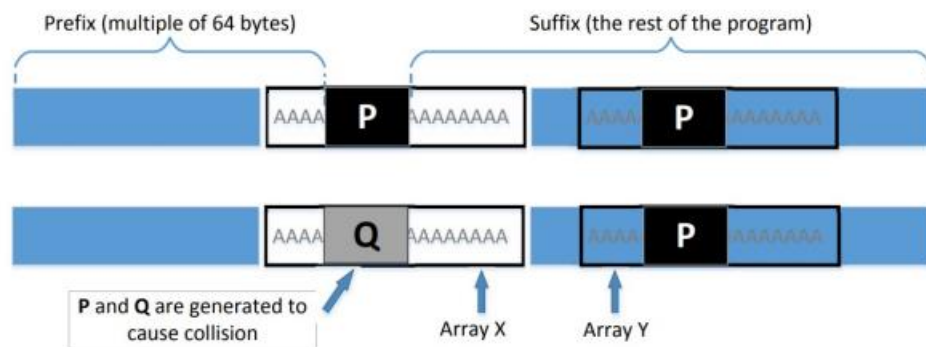
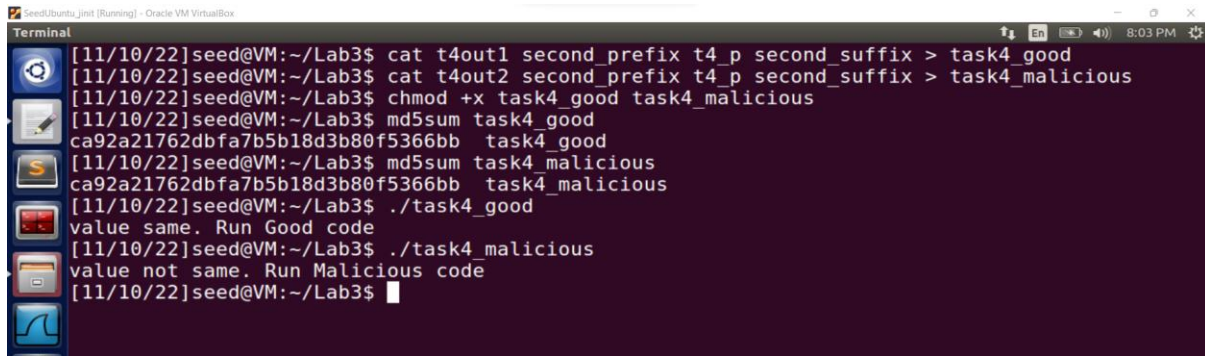


Figure 4.5



- The above structure can be satisfied by concatenating files in below order
 - T4out1 + second_prefix + t4_p + second_suffix (t4out1 = prefix + P)
 - T4out2 + second_prefix + t4_p + second_suffix (t4out2 = prefix + Q)



```

[11/10/22]seed@VM:~/Lab3$ cat t4out1 second_prefix t4_p second_suffix > task4_good
[11/10/22]seed@VM:~/Lab3$ cat t4out2 second_prefix t4_p second_suffix > task4_malicious
[11/10/22]seed@VM:~/Lab3$ chmod +x task4_good task4_malicious
[11/10/22]seed@VM:~/Lab3$ md5sum task4_good
ca92a21762dbfa7b5b18d3b80f5366bb task4_good
[11/10/22]seed@VM:~/Lab3$ md5sum task4_malicious
ca92a21762dbfa7b5b18d3b80f5366bb task4_malicious
[11/10/22]seed@VM:~/Lab3$ ./task4_good
value same. Run Good code
[11/10/22]seed@VM:~/Lab3$ ./task4_malicious
value not same. Run Malicious code
[11/10/22]seed@VM:~/Lab3$

```

Figure 4.6 programs run different code based on array value but have same md5 values

- So the final programs will be 'task4_good' and 'task4_malicious'
 - Task4_good – array1=P and array2=P
 - Task4_malicious – array1=Q and array2=P
- Gave executable permission to both files.
- We can see both final files have same md5 values in figure 4.6
- Output can be observed as expected from both files in figure 4.6.
- Hence, we have achieved the task of creating two programs with same md5 hash values. One program intended to run benign instructions and the other to run malicious instructions.