**ACS 54500 Cryptography and Network Security**

Lab 9: MD5 Collision Attack Lab

**Name:** Vijayagiridharan Subramanian

**Task 1 (Generating Two Different Files with the Same MD5 Hash) – 20pts**

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

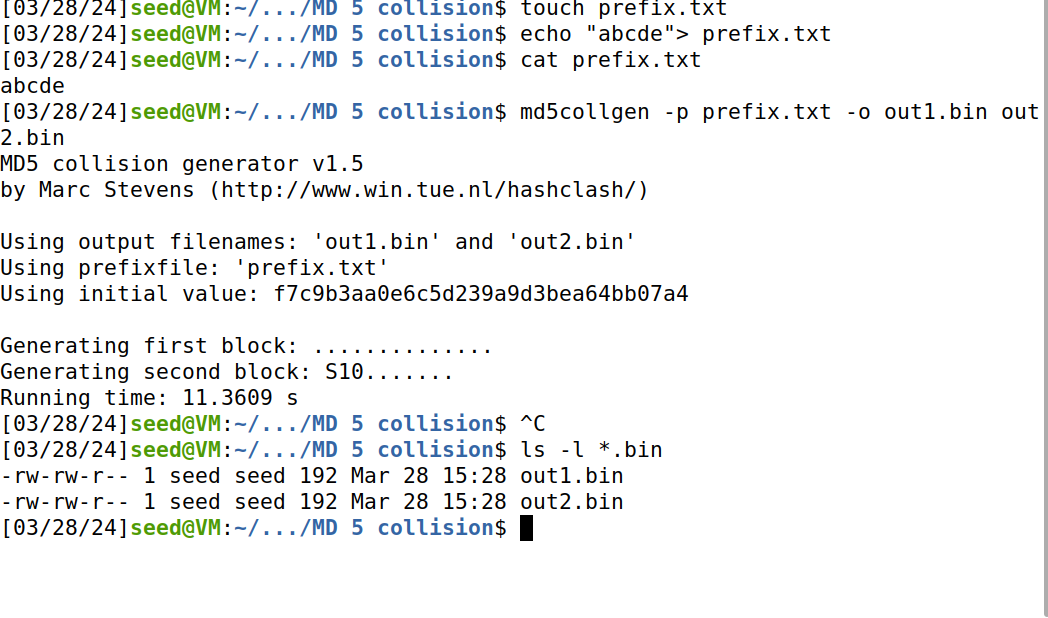
We are creating 2 different files with MD5 hash values. **md5 collagen** tool makes 2 binary files out1.bin and out2.bin with the same MD5 hash value for the given prefix. When we check both out1 and out2 we will find the same MD5 hash value. But the content of both out1.bin and out2.bin will be different. We are going to see the binary files using editor software called **bless**.

These out1.bin and out2.bin will have 64+128 bytes individually. 64 bytes for user data that is Prefix, If we are giving only 4 bytes as prefix, 64 bytes will be padded with 0 values.

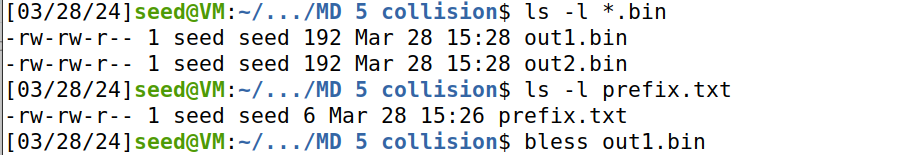
**Question 1. If the length of your prefix file is not multiple of 64?**

We are doing it practically,

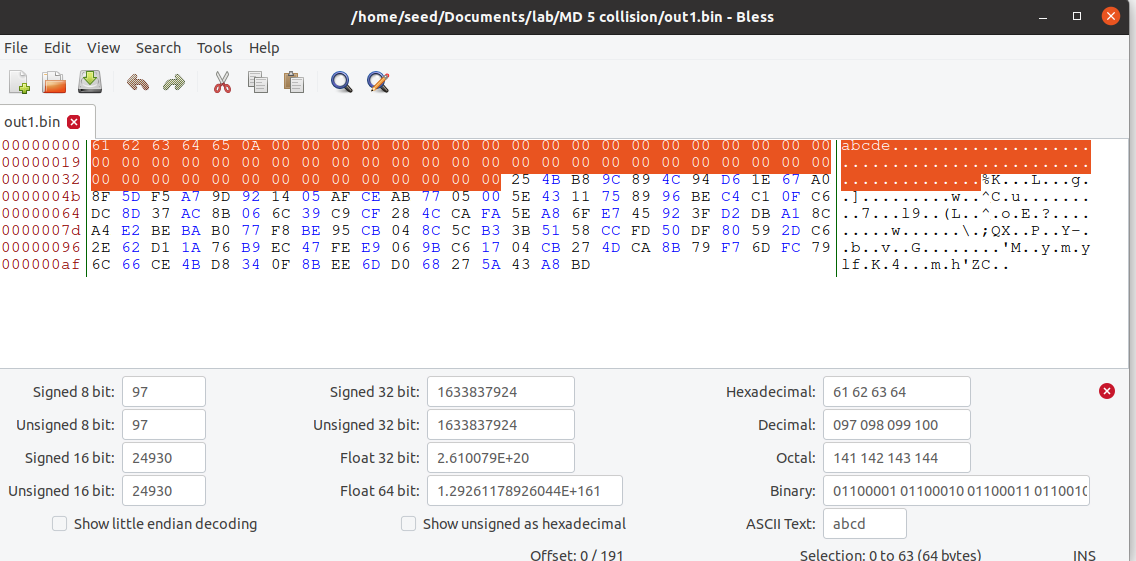
We are inserting values inside the file using echo "abcde" which will go inside prefix.txt. Each character will take 1 byte, 8 bits of data. **md5collgen -p prefix.txt -o out1.bin out2.bin** generates a prefix file: 'prefix.txt'. Then we see binary files out1.bin and out2.bin using **ls -l \*.bin**.



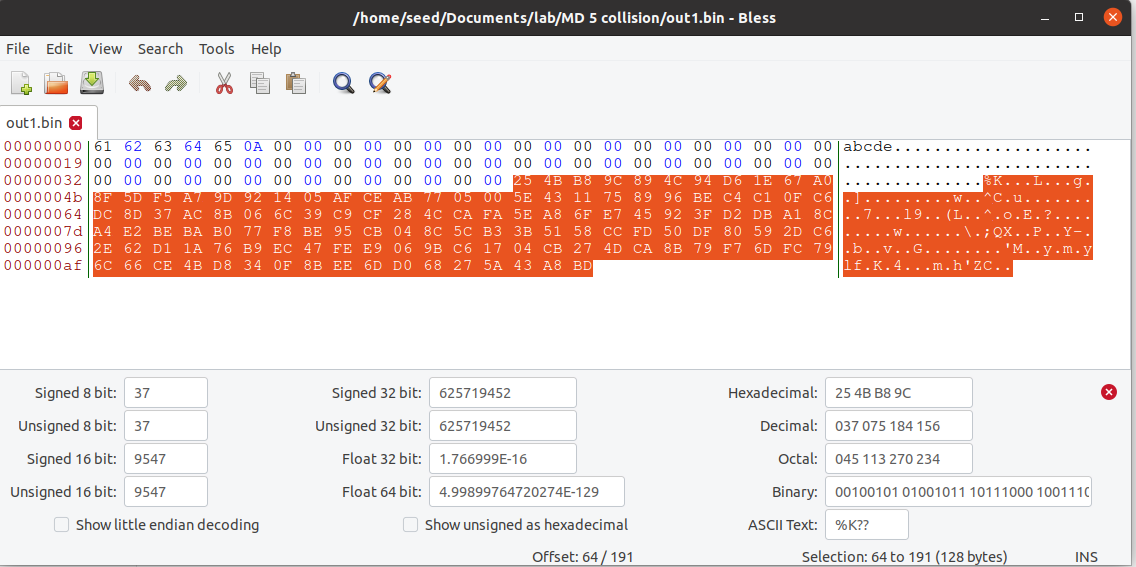
We can find prefix.txt using this command ls -l prefix.txt. The size of prefix.txt is 6 which is abcde(5) with space(1).Now, we are going to see the binary file out1.bin using bless.



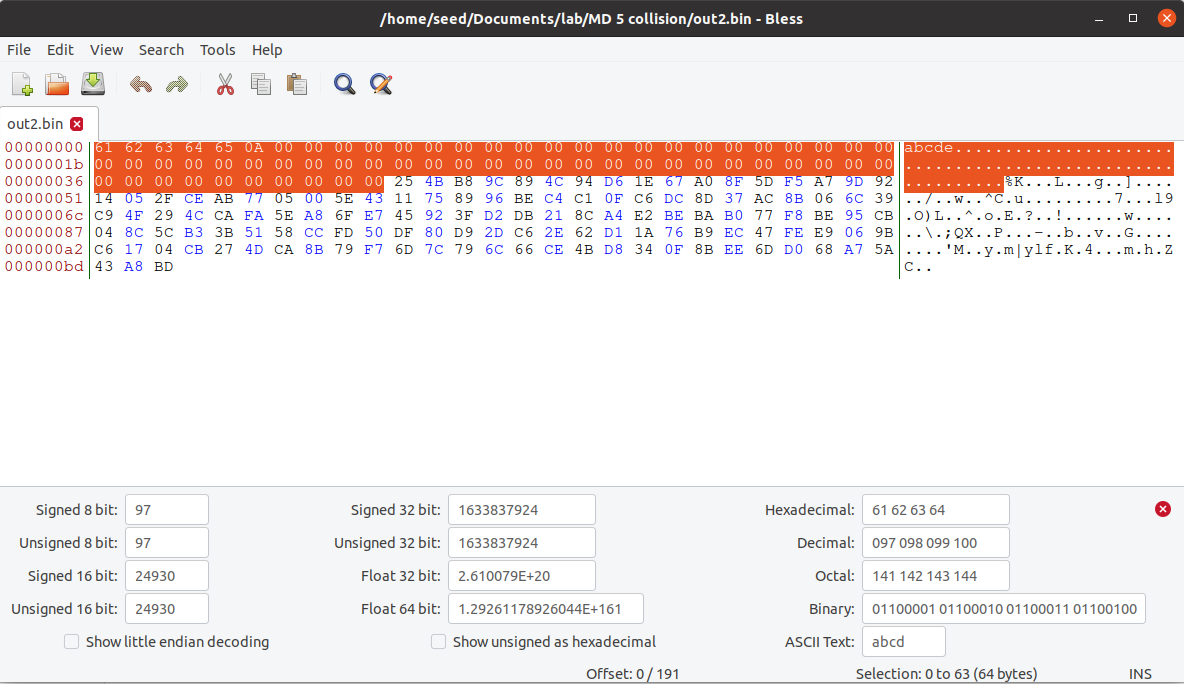
As we give only 5 bytes of data, the sixth byte is an empty character. "." after abcde indicates the extra space, then after the sixth byte till the last value it is padded with 0. In the selection we can see (0 to 63 bytes) which is highlighted.



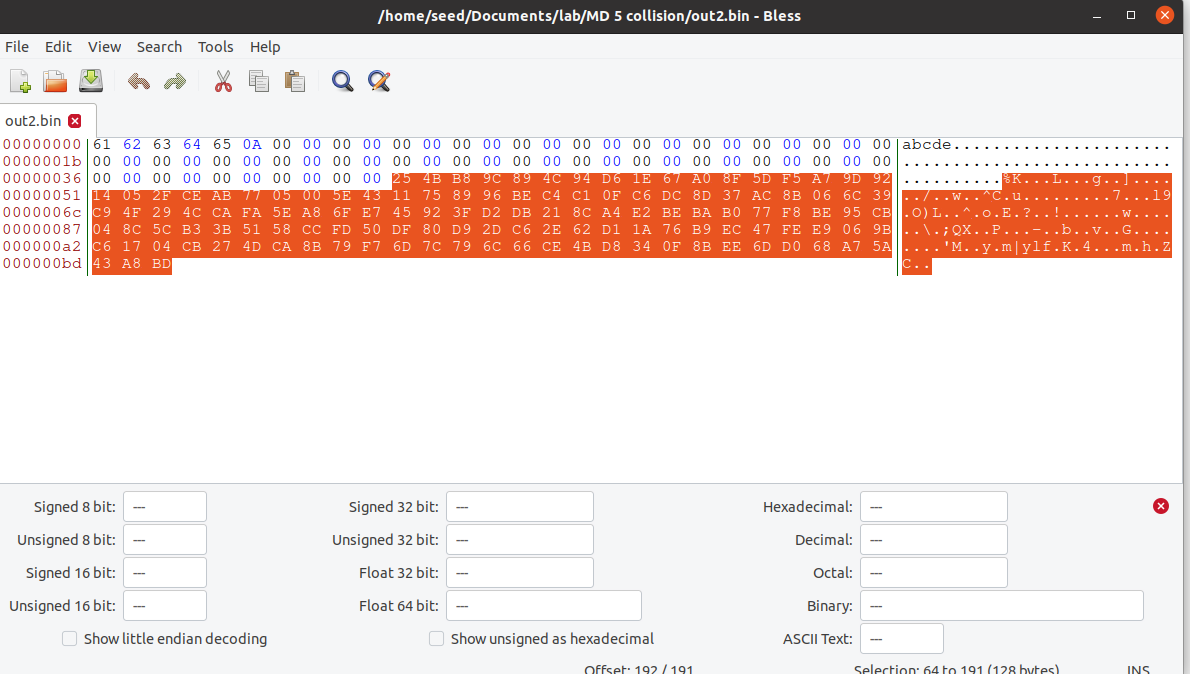
Here the highlighted part (64 to 191) shows 128 bytes which is considered as P value in out1.bin.



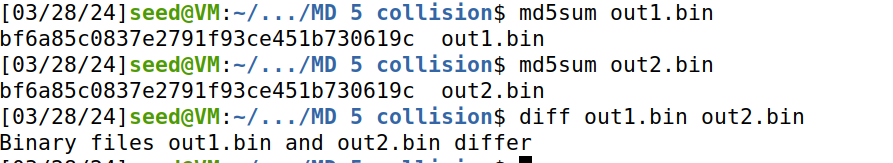
Similarly opening **out2.bin**, we can find only 5 bytes of data, the sixth byte is an empty character. "." after abcde indicates the extra space, then after the sixth byte till the last value it is padded with 0. In the selection we can see (0 to 63 bytes) which is highlighted.



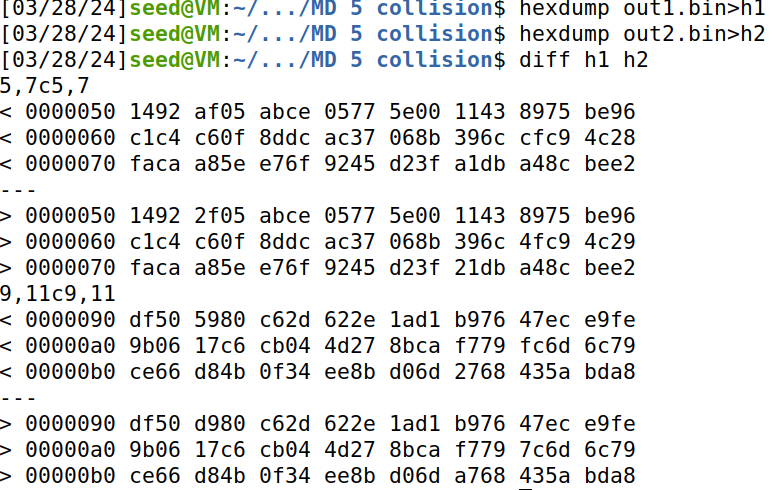
Here the highlighted part (64 to 191) shows 128 bytes which is considered as Q value in **out2.bin**.



We are now checking the md5sum of out1.bin and out2.bin, we will get the same md5 value, which shows that both the file values are the same. If we want to prove the content (out1.bin and out2.bin) are different,using diff out1.bin out2.bi and we get binary files that are "different".



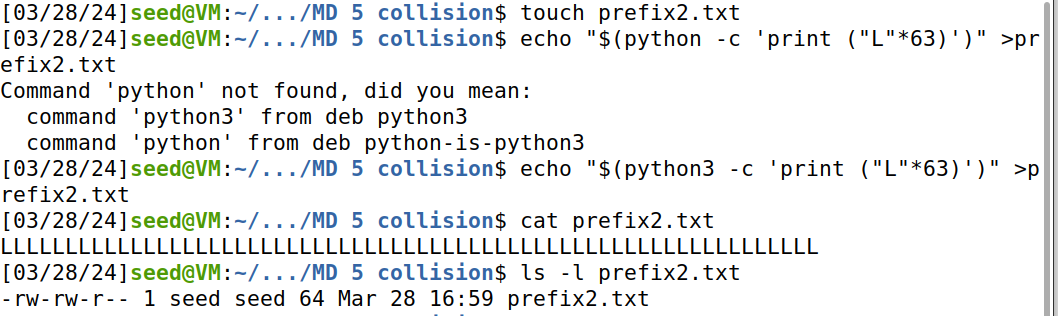
We are finding out the difference using the command "hexdump" to change it into the hexadecimal. Now if we diff h1 h2, we can get the difference.



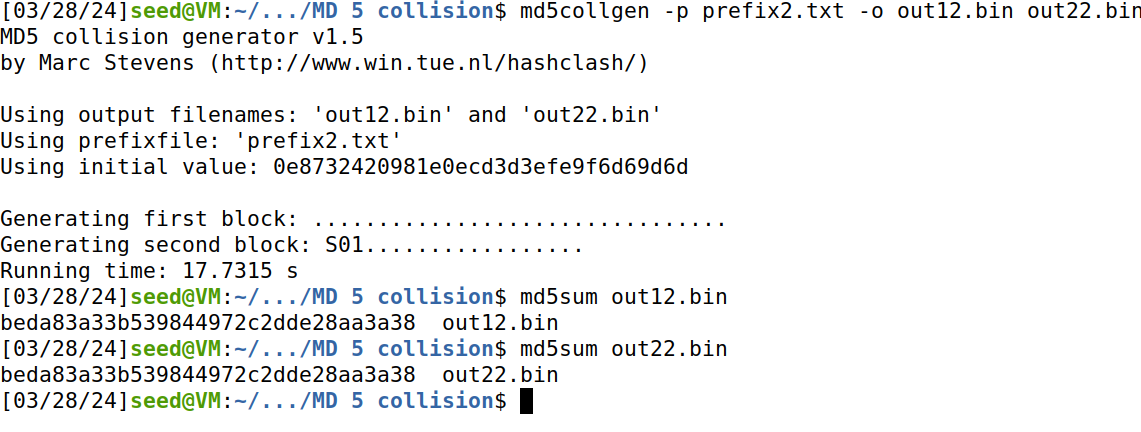
**Here the length of the prefix file is only 5 bytes that is ”abcde”, which is not the multiple of 64. So the remaining data is padded with 0 to make it multiple of 64 byte.**

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

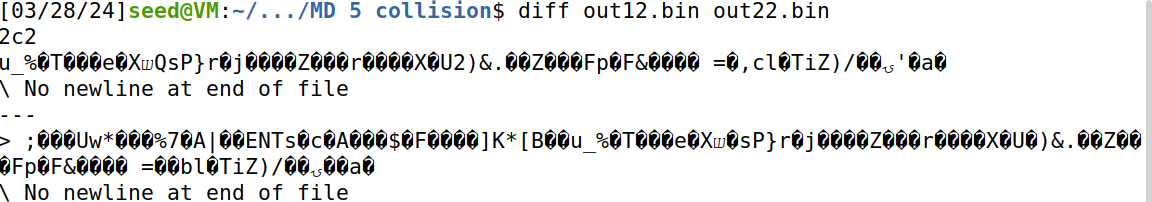
We are creating prefix2 for this part, we are now write 64 byte using echo "$(python3 -c 'print ("L"\*63)')" >prefix2.txt. Now we can see inside prefix2.txt to find out 64 L and we can check the size using ls -l prefix2.txt which is 64.



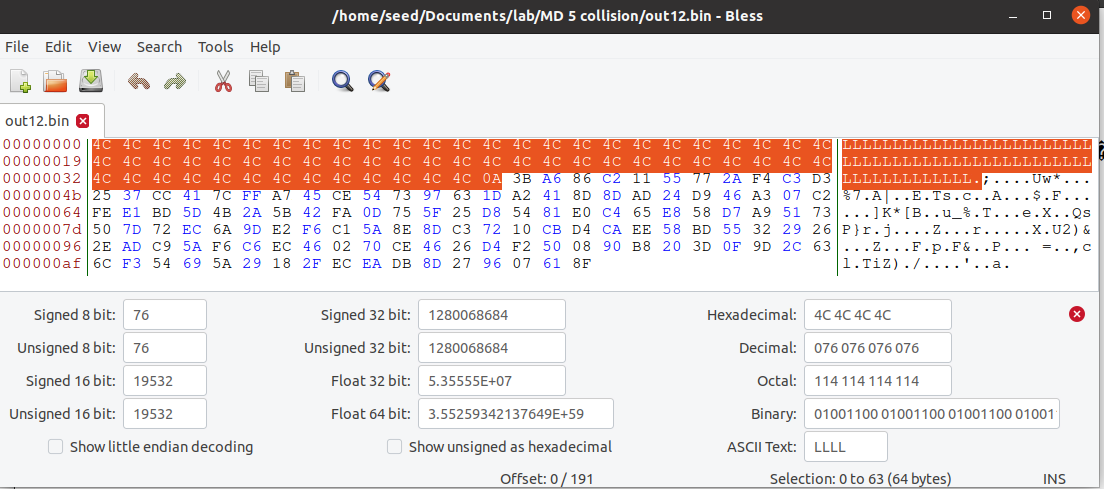
Then we are generating the 2 binary files out12.bin and out22.bin using md5 collagen. While checking it with md5sum , we can see that 2 will look similar.



Now we are checking the difference in out12.bin and out22.bin, we find the difference.



Now we open out12.bin using bless. Highlighted space with empty space OA is 64 bytes. **so, when we create a prefix file with 64 bytes, there will be no padding which shows if the data is less than 64 bytes, then it is padded with 0s.** Unhighlighted bytes after the highlighted one are called P bytes.



**Question 3. Are the data (128 bytes) generated by md5 collagen completely different for the two output files? Please identify all the bytes that are different.**

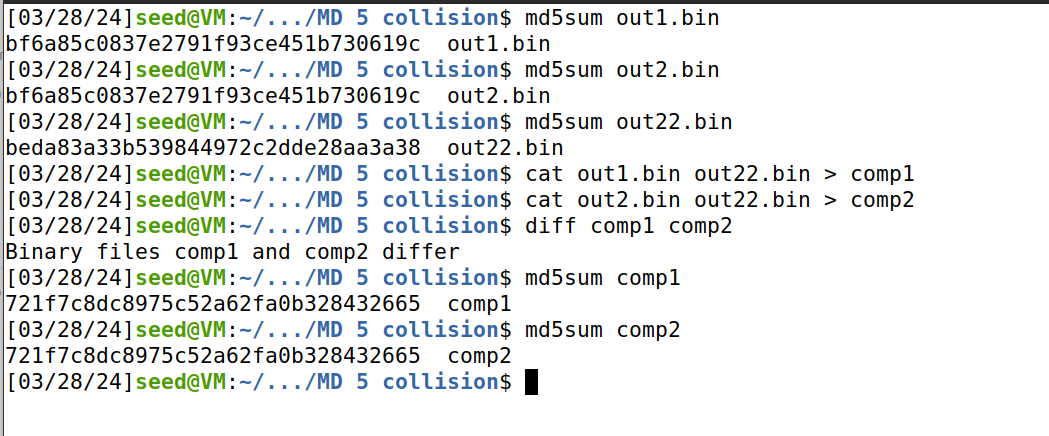
Now we will do hexdump to create 2 hexadecimal files, we can see what is the difference in bytes. We can find change in bytes and round it.

Not all bytes are different only there 14 bytes changes, most are similar



**Task 2 (Understanding MD5’s Property) – 10pts**

**We are going to demonstrate the MD5 property, using this experiment.**

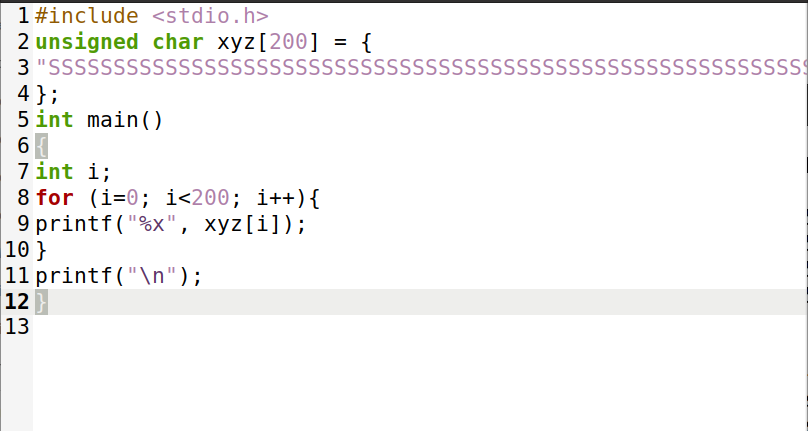


Now, we do md5sum for out1.bin and out2.bin,and it is the same. so we consider out1.bin as m and out2.bin as n. Now we take the md5sum of out22.bin. We are going to join out1.bin with out22.bin using **cat out1.bin out22.bin > comp1** .Similarly, join out2.bin with out22.bin using **cat out2.bin out22.bin > comp 2**.Then we check difference between comp1 and comp2, which differs.When we check md5sum of comp1 and comp2, it is similar.

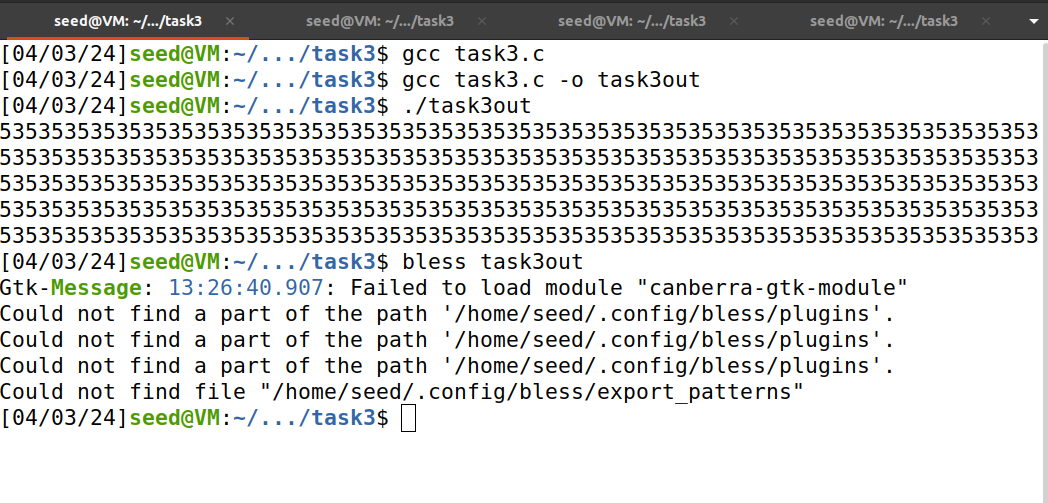
**Task 3 (Generating Two Executable Files with the Same MD5 Hash) – 30pts**

We have changed the code provided and we give the letter "S"200 times, which will print the hexadecimal value of "S".

**Code:**

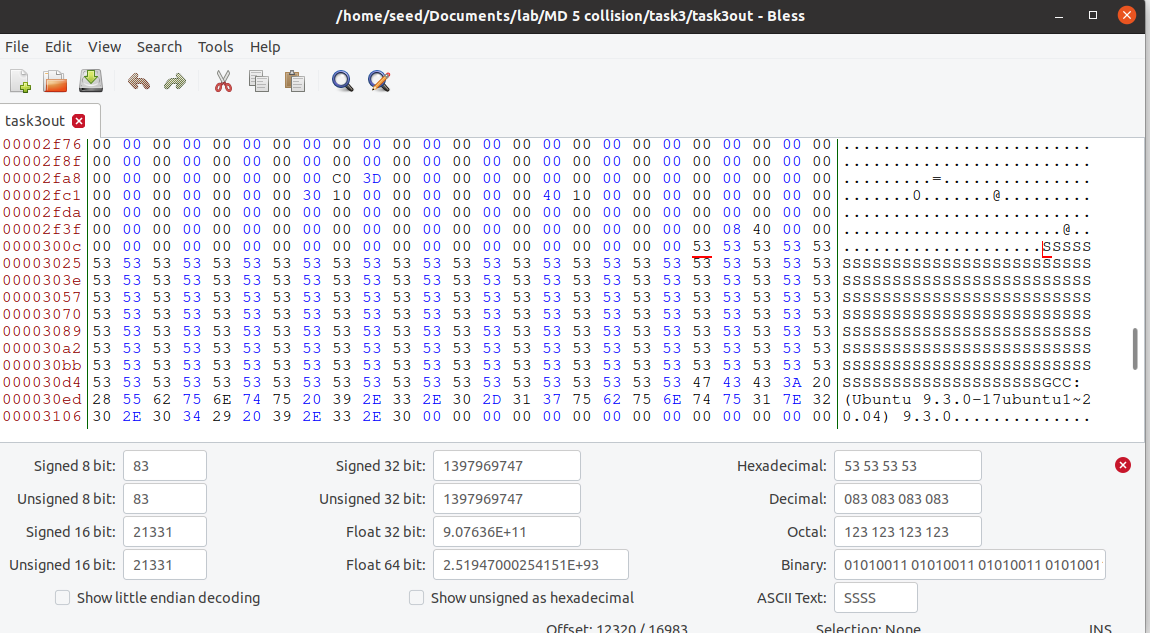
****

Once we run that code, we will get a hexadecimal value of 200"S" like "5353..." for 200 "S".



**Bless Output:**

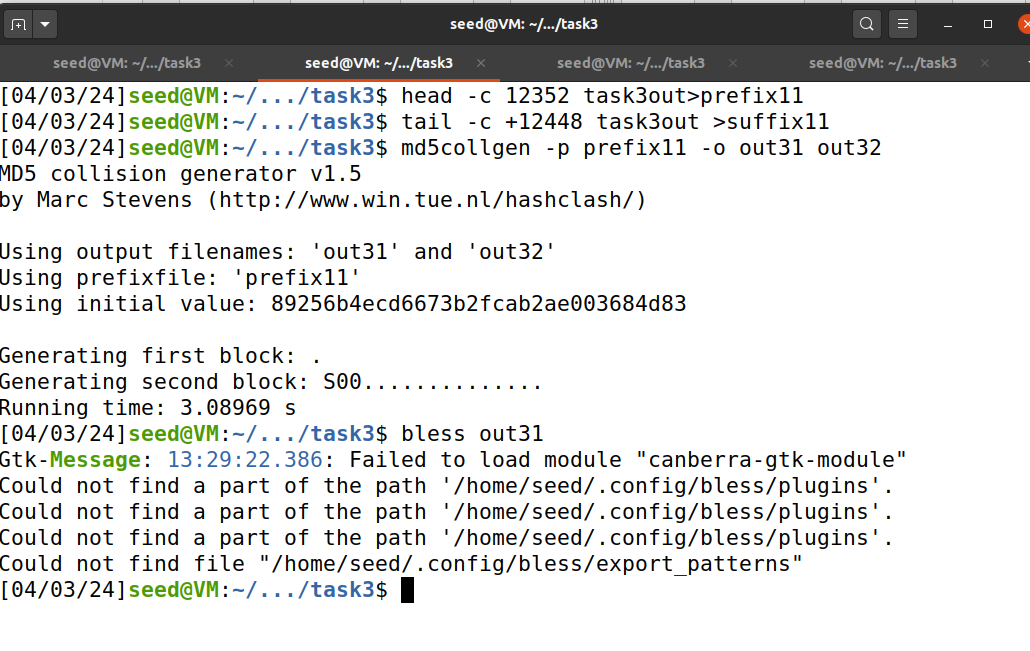
We open bless and locate where S s found



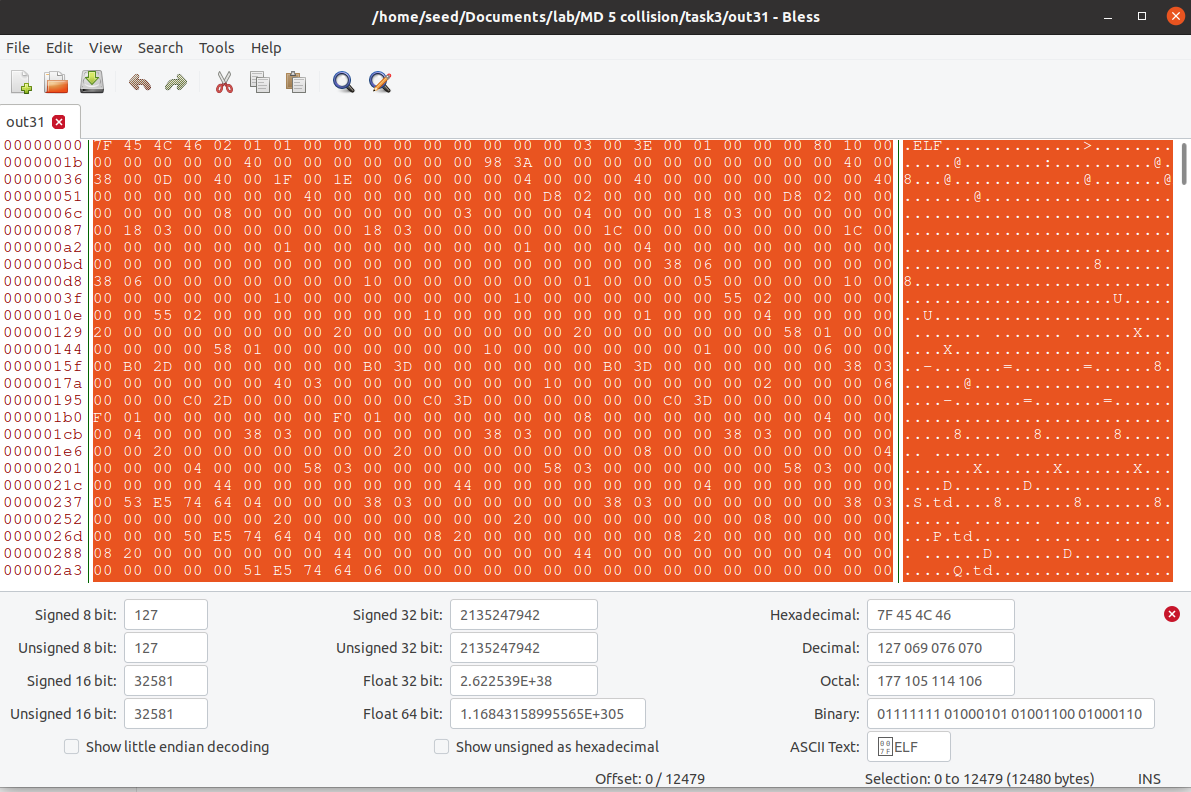
**INFO**: bytes before SSSS.. are Prefixes ,bytes after SSSS.. are suffixes. And SSSS.. are 128 bytes.

Prefix is from 0 to 12352(nearest number multiple of 64) and suffix starts from 12448 till end.After 12352 bytes, count 128 bytes(12448 bytes).After that it is suffix.

We are going to create prefixes and suffixes using the head and tail method. Prefix is from 0 to 12352 and saved as **prefix11** and suffix starts from 12448 till end is saved as **suffix11** .Now, we are going to do MD5 collision generator on prefix11. We are creating 2 files one is out31 and other is out32 using md5 collgen.

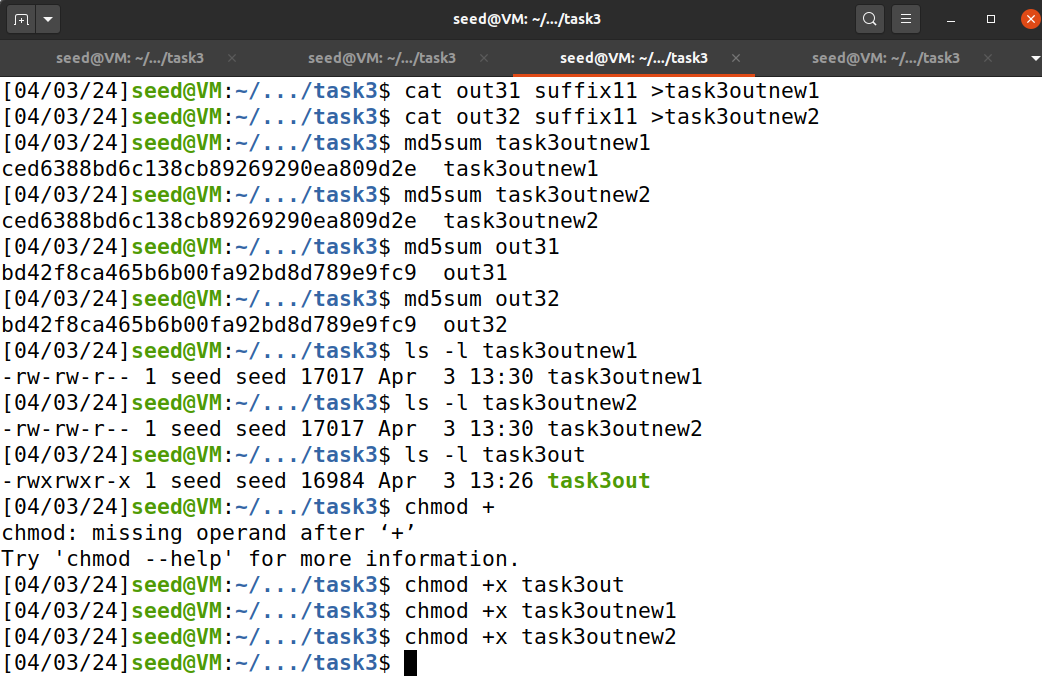
****

When we open the bless, we can find the size of bytes( that is 12480 bytes)



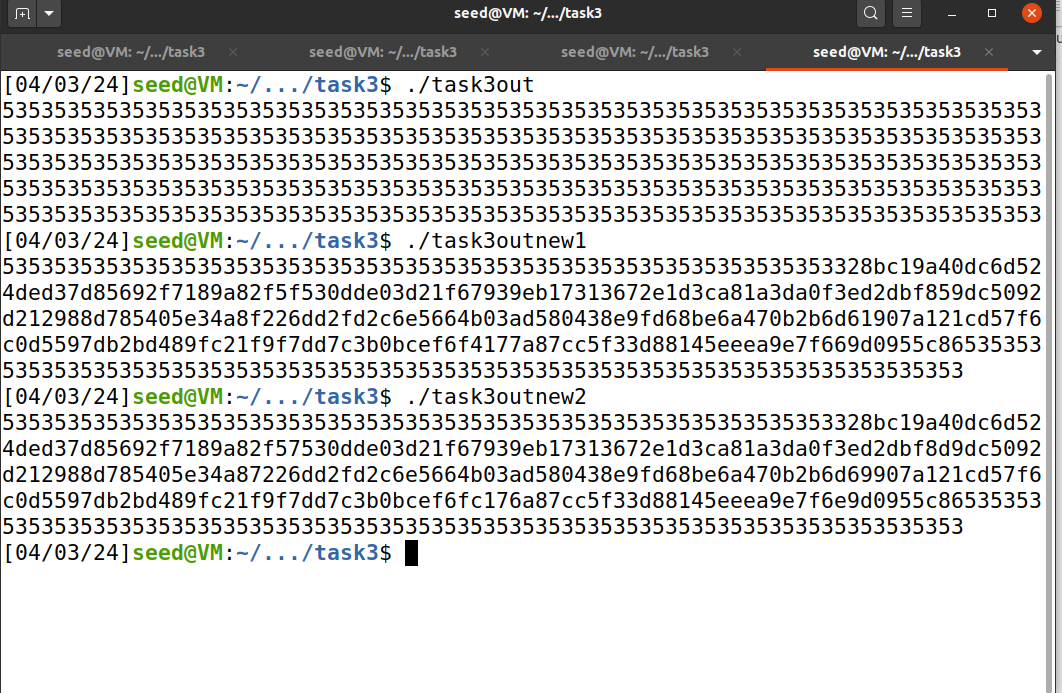
Now, we completely replace from 0 to 12480 bytes as out31, the suffix is kept the same. We create temp1 with out31 and suffix11.

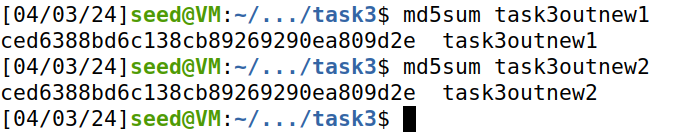
We are creating **task3outnew1** by concatenating out31 and suffix11 and creating **task3outnew2** by concatenating out32 and suffix11. md5sum of task3outnew1 and **task3outnew2** are the same. Similarly, md5sum of out31 and out32 are the same.



When I run **task3out**, it will run and give "5353..".When I run **task3outnew1**, it will run and print content of value of P upto 128 bytes and remaining give "5353.." which is hexadecimal value of "SS..".When I run **task3outnew2**, it will run and print content of value of Q upto 128 bytes and remaining give "5353.." which is hexadecimal value of "SS..".

These output files task3out, **task3outnew1** and **task3outnew2** are giving different outputs.





We achieved the target of creating 2 new executable files which output with different content. However, the MD5 values are the same completed task3 successfully.

**Task 4 (Making the Two Programs Behave Differently)– 40pts**

We should make 2 programs in the binary mode that should behave differently but should have the same MD5 hash value.

**Code:**

#include <stdio.h>

#define LENGTH 400

unsigned char X[LENGTH]= {

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

};

unsigned char Y[LENGTH]= {

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"

};

int main()

{

int i = 0;

for (i =0; i< LENGTH; i++){

if (X[i] != Y[i]) break;

}

if (i==LENGTH){

printf("%s\n", "Executing benign code... ");

}

else {

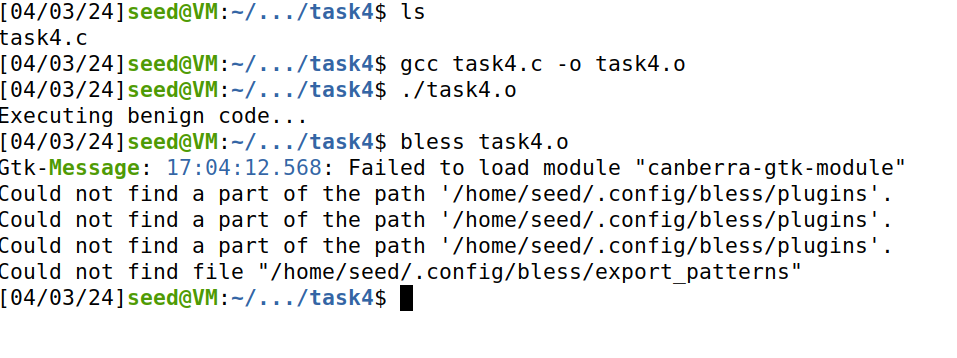
printf("%s\n", "Executing malicious code... ");

}

return 0;

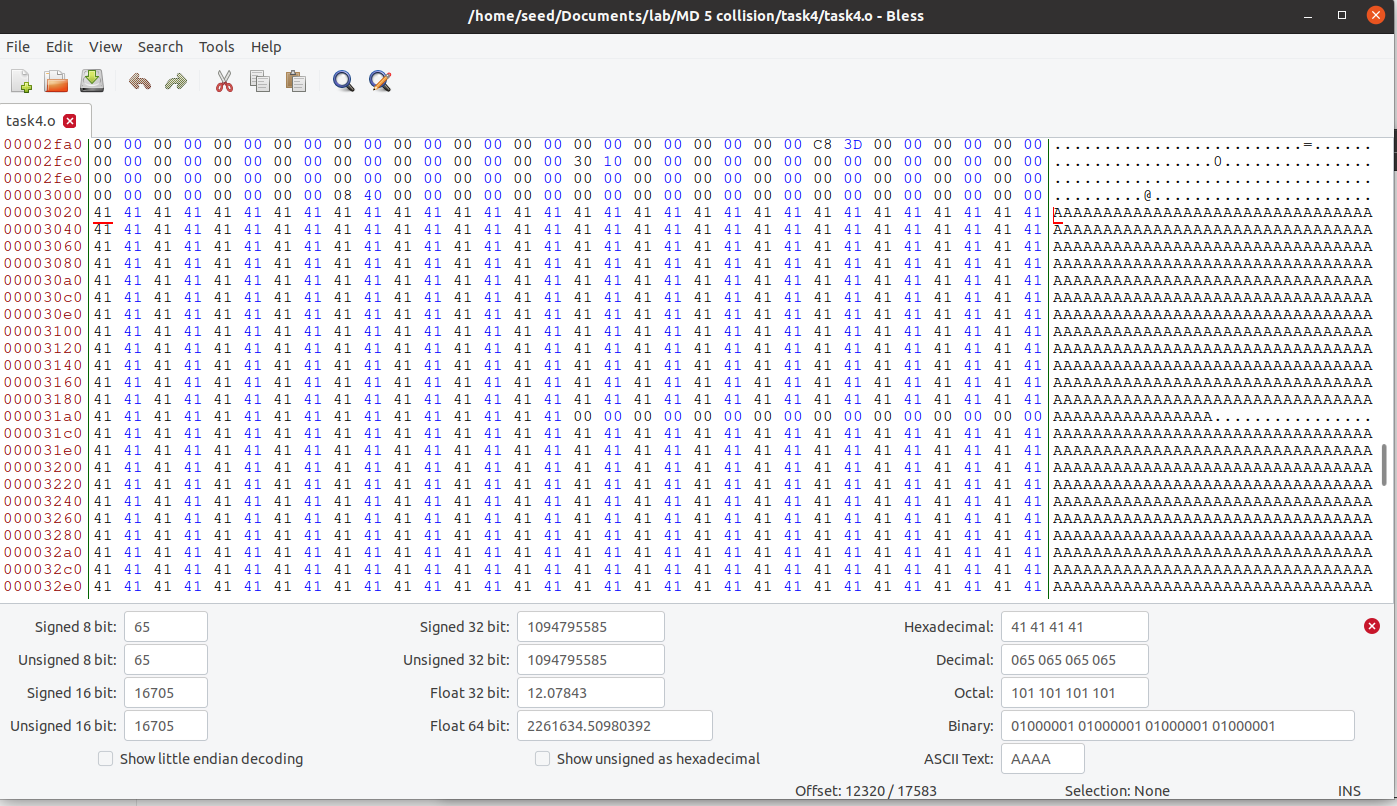
}

We are running a file using the command **gcc task4.c -o task4.o** , then we run task4.o, it will print (Executing benign code …) because my X and Y values are the same. Now we open object file task4.o using bless



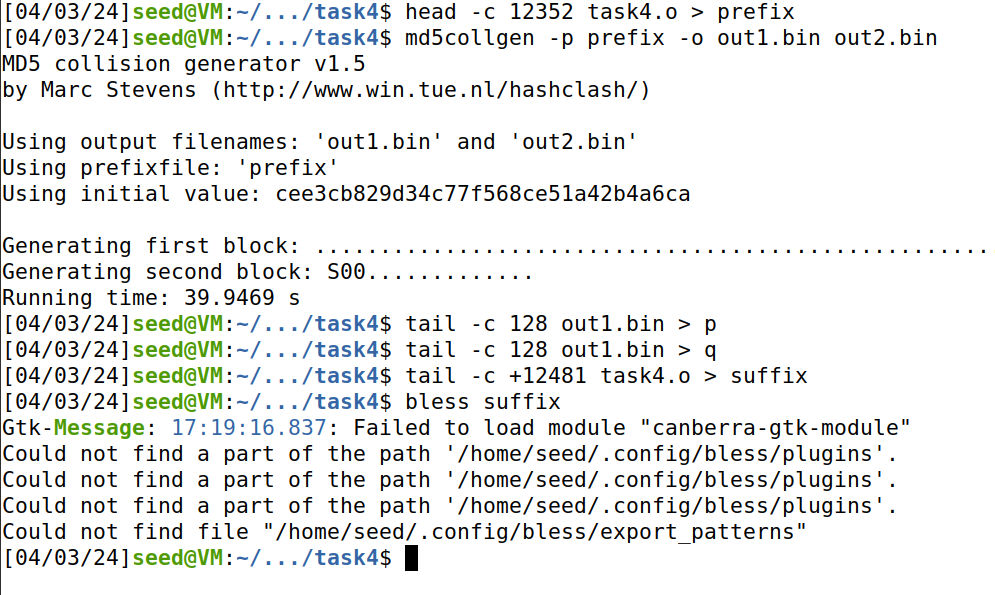
From this we note where x and y arrays are starting. AA.. are starting at 12320 decimal.

Bless Output of **task4.o:**

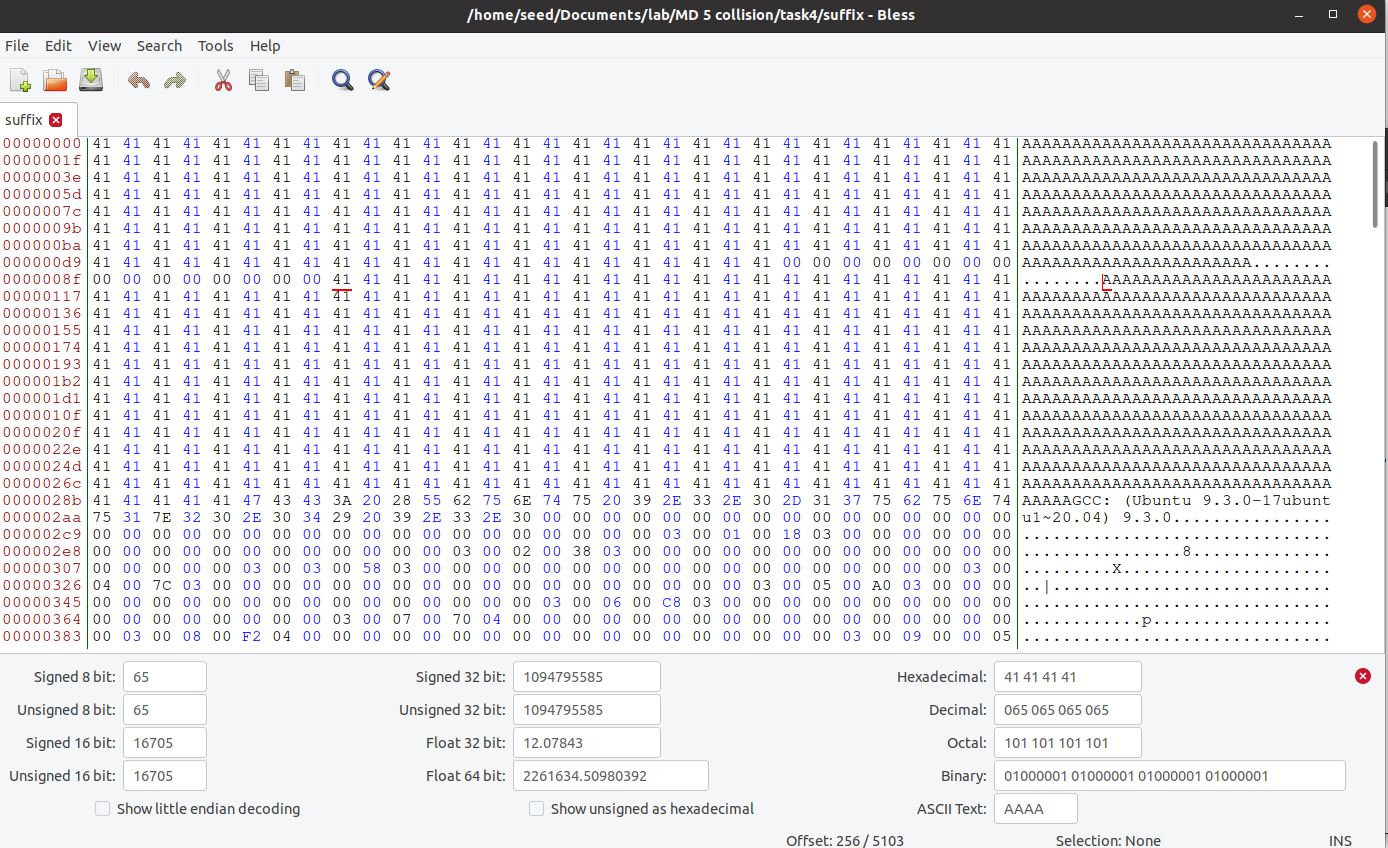


Now we are creating prefix which is head using command **head -c 12352 task4.o > prefix**. Then we are using **md5collgen -p prefix -o out1.bin out2.bin**, which generates MD5 collisions using prefix and produces 2 output files out1.bin and out2.bin.

The command **tail -c 128 out1.bin >p** extracts 128 bytes from the file out1.bin and saves to p file. Similarly, **tail -c 128 out1.bin >q** extracts last 128 bytes from out1.bin and saves into file q.The command **tail -c 12481 task4.o >suffix** , extracting suffix from task4.o starting from byte 12481 and save to file suffix.



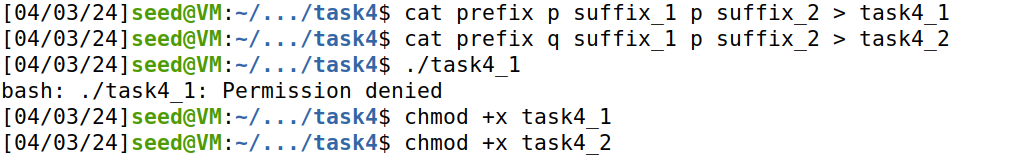
The below one is the bless output of suffix:



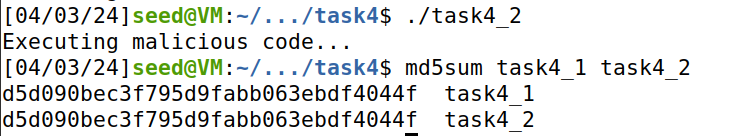
We open the bless to find out the starting of y is starting. then we create suffix\_1 using **head -c 256 suffix >suffix\_1**. Then we create suffix\_2 using **tail -c +384 suffix >suffix\_2.(384 =256+128**).



Now we are concatenating files prefix, p, suffix\_1, p and suffix\_2 to task4\_1. Similarly, we are concatenating prefix, q, suffix\_1, p and suffix\_2 to task4\_2. Then we are running task4\_1, we get output **Executing benign code.**. Similarly, while we are running task4\_2, we get **Executing malicious code**.







When we check the md5sum of task4\_1 and task4\_2 are the same, but show different output while running it(which behave differently).