**SHIQI LIU**

**LAB REPORT**

**CS458**

**Introduction:**

This lab delves into the MD5 collision attack which makes use of its length extension property. In this lab, I compare two different files with the same MD5 Hash, and understand MD5 property deeply, then I created two different versions of the program, and make them behave differently. I found an interesting observations is a hash function is said to be secure if it is a one-way hash function and is collision-resistant. The one-way property ensures that given a hash value h, it is computationally infeasible to find an input m such that hash(m) = h.

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

In this task, we need to find out two different files with same MD5 Hash. First, I create a test file.

Command is –

1. echo TEST > prefix.txt

Then, I generated two files with the same md5 hash

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

Text

Description automatically generated

Then, I checked that the md5sum of these files are the same:

Text

Description automatically generated

Difference of these files are shown blow:

Text

Description automatically generated

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

Answer: It will be padded with zeros.

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

Answer: It’s almost the same as previous experiments, no zero padding is observed.

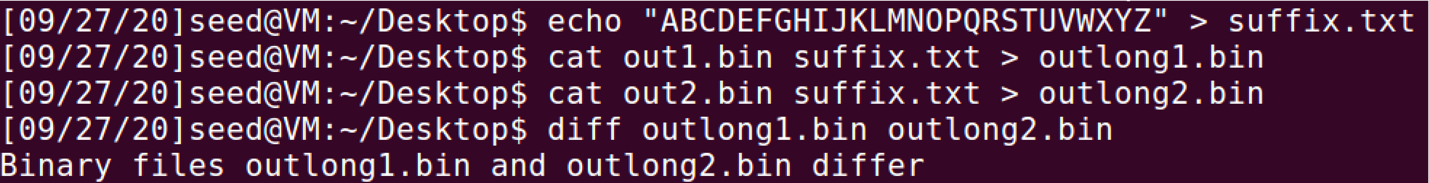
***Question 3. Are the data (128 bytes) generated by md5collgen completely different for the two output files? Please identify all the bytes that are different*.**

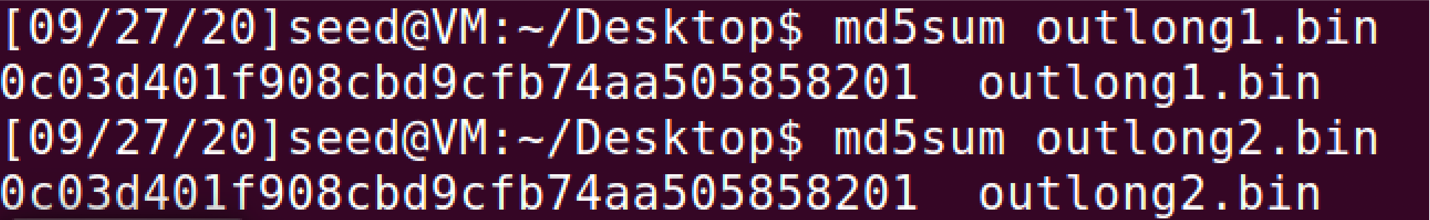
Answer: No, most bytes are the same, but there still are few bytes are different.

**Task 2: Understanding MD5’s Property**

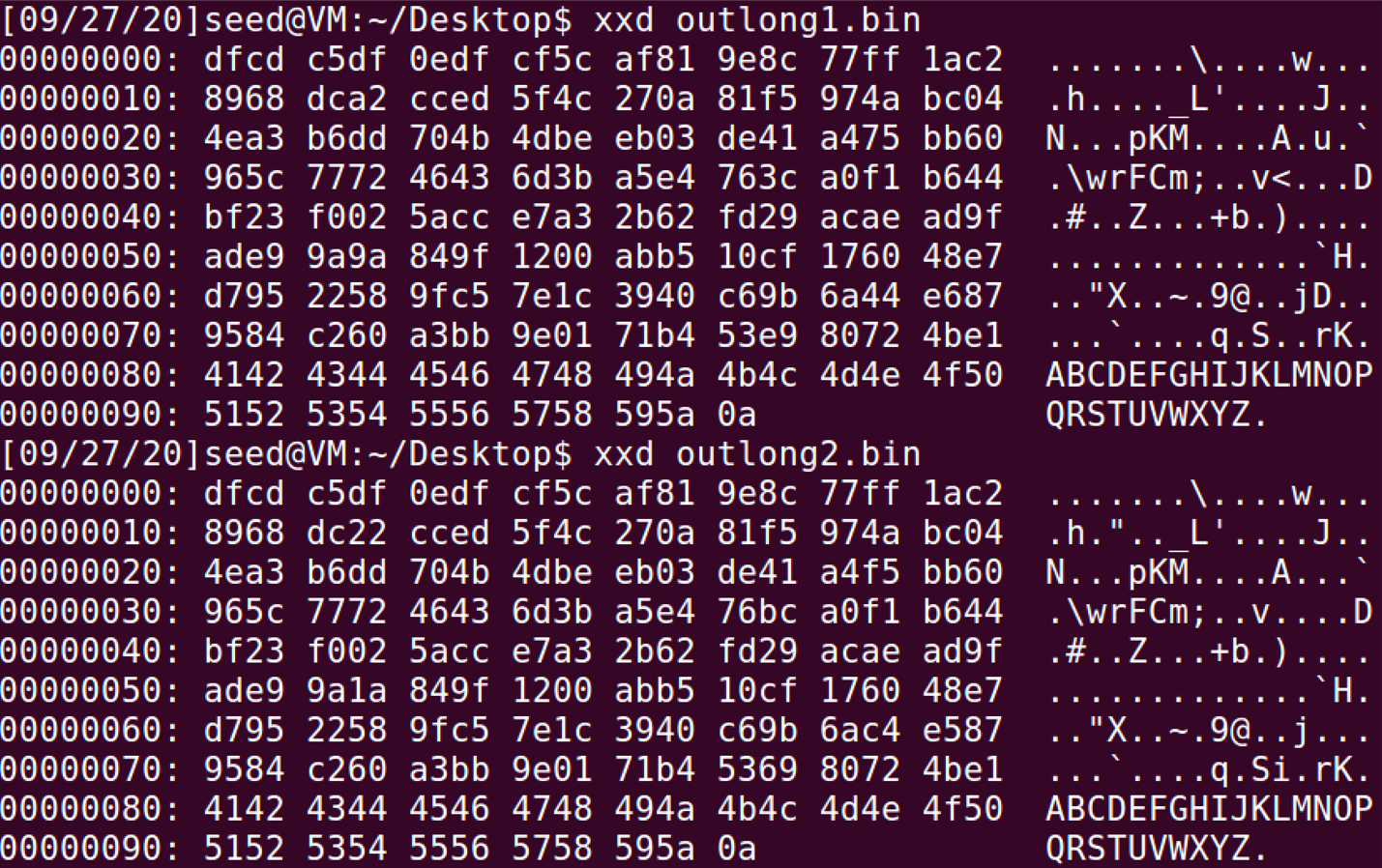
At high level, MD5 divides its data into blocks of 64 bytes and then computes the hash iteratively on these blocks. The core of MD5 is a compression function which produces a 128 bit IHV or intermediate hash value. The input for the first iteration i.e., IHV0 is fixed. Based on the working of the MD5 algorithm, we can derive a property which is - Given two inputs M and N, if MD5(M) = MD5(N), then for any input T, MD5(M || T) = MD5(N || T). Therefore, adding a particular suffix to any two distinct messages having the same MD5 hash, gives two new longer messages for by concatenation of the original and the suffix messages, both of which also have the same MD5 hash.

Therefore, in this task we reuse the outputs from task 1. We have generated two sets of files, there are two files in each set with the same md5sum. We randomly choose file A from the first set and append same content to it to genetate file C. If the properity of MD5 holds, then the MD5 of C remains the same.



using md5sum command: 

using “xxd” command:

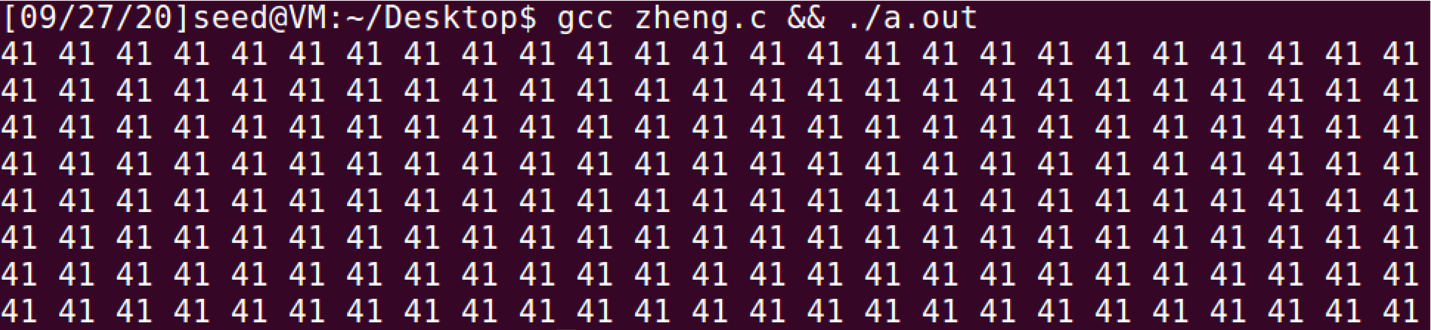


**Task 3: Generating Two Executable Files with the Same MD5 Hash**

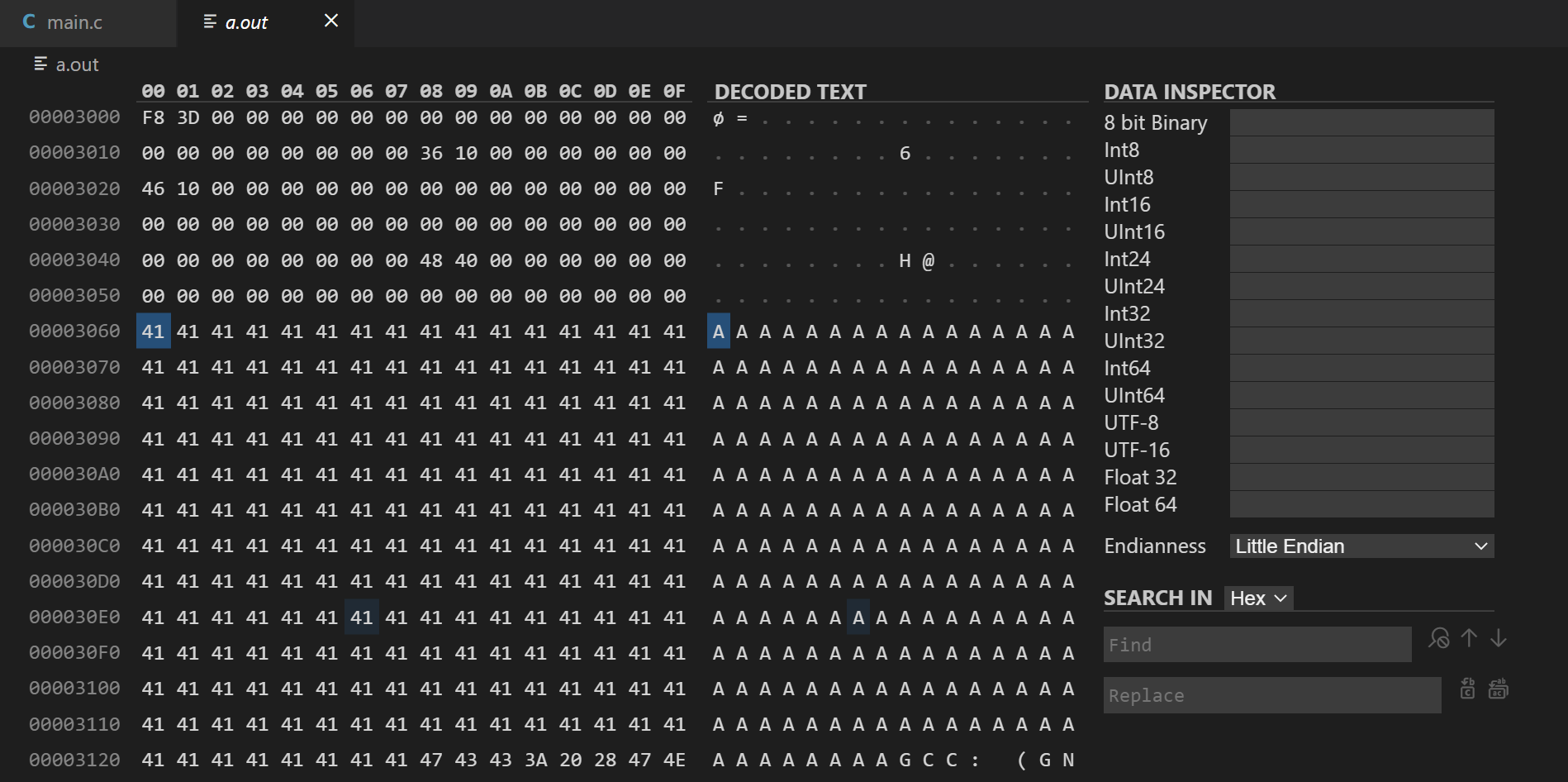
Given a code in C, create two different versions of this code such that the difference in them lies in the array contents, but the hash values of their executables are the same. The code is -

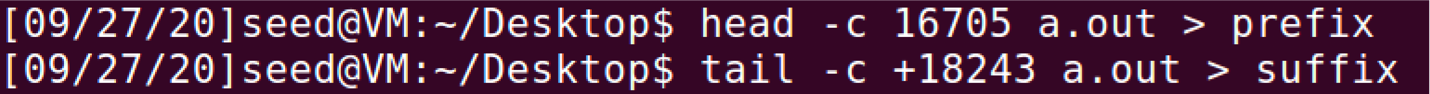
1. #include <stdio.h>
2. unsigned **char** xyz[200] = {
3. "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
4. };
5. **int** main() {
6. **int** i;
7. **for** (i = 0; i < 200; i++) {
8. **if**(i%25 == 0 && i > 0){ printf("\n");
9. }
10. printf("%x ", xyz[i]); }
11. printf("\n"); }

the output of this program is:

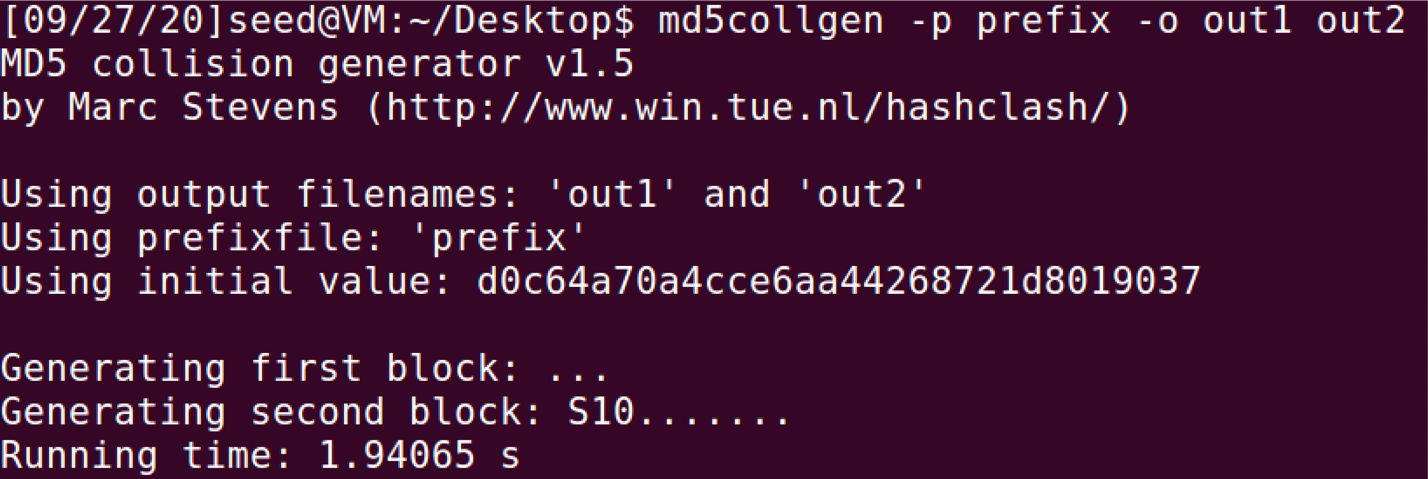


To find the position of the array, I used the hex plugin with VS code, it is easy to identify the position of a long secquence of 'A'.

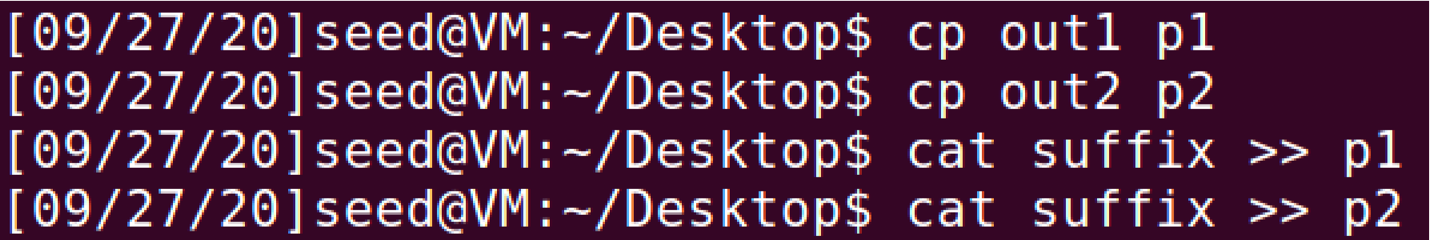


Then, creating prefix and suffix files: 

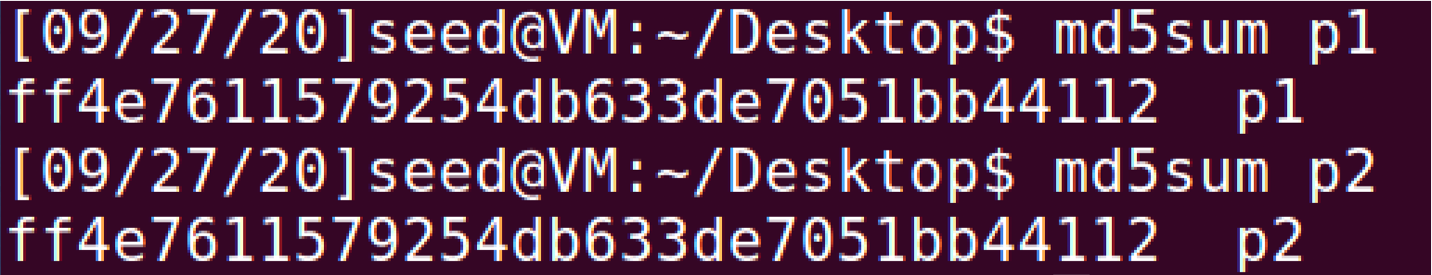
Output files:



Then, assemble these two new programs:



Checking the md5sum od these two programs:



Finally checking the outputs are the same.

**Task 4 : Making two programs behave differently**

In this task, we need to design a program to make two programs that share the same MD5 hash behave differently.

Therefore, write a program as follows.

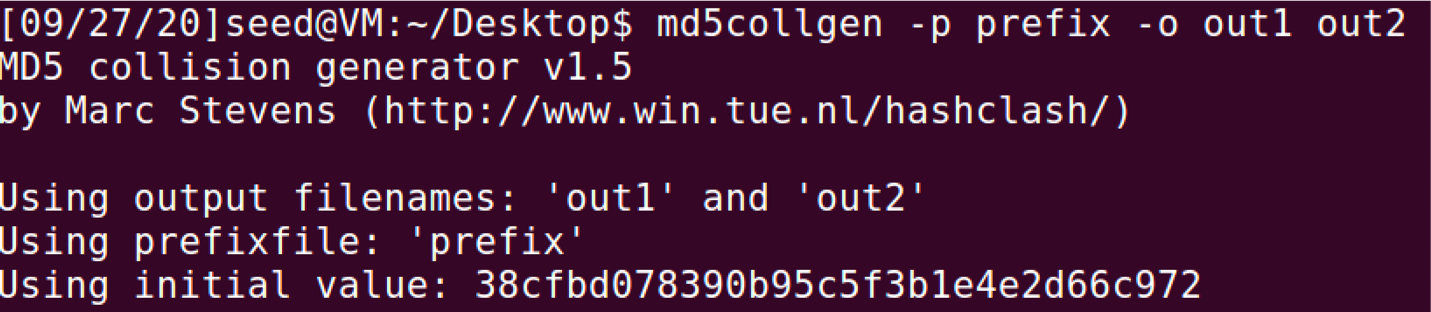
1. #include <stdio.h>
2. unsigned **char** A[128] = {
3. "BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"};
4. unsigned **char** B[128] = { "BAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" "AAAAAAAAAAAAAAAAAAAAAAAAAAAA"};
5. **int** main() {
6. **for** (**int** i = 0; i < 128; i += 1) {
7. **if** (A[i] != B[i]) {
8. printf("Malicious Code Executed!\n");
9. **return** 0; }
10. }
11. printf("Every Thing Seems Good!\n"); **return** 0; )

If contents in array A and B are the same, the program will print

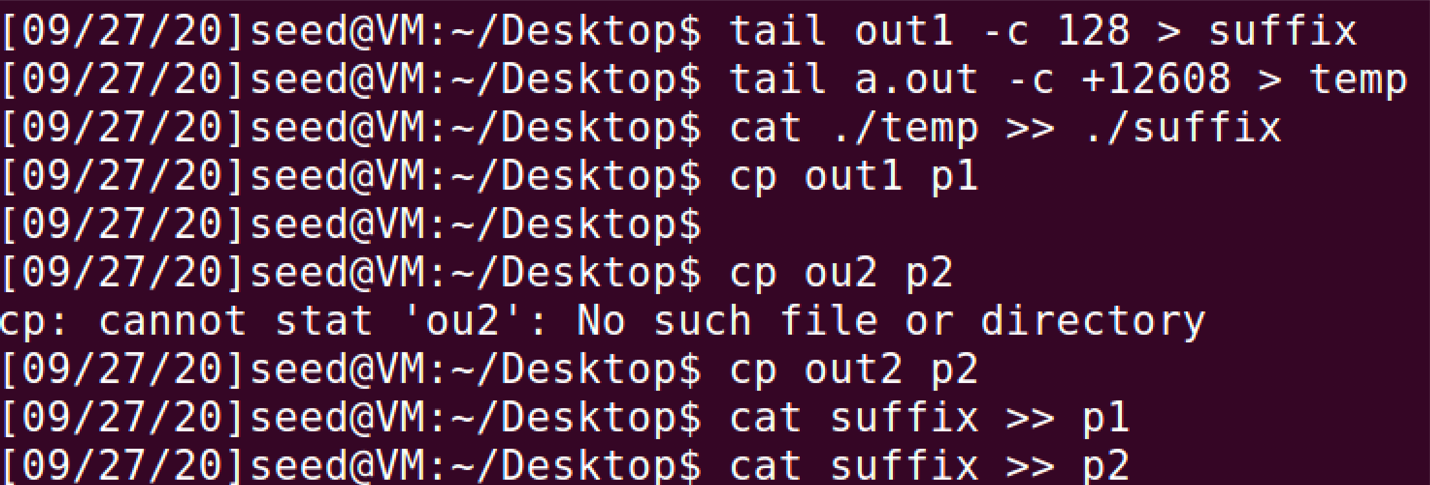
Every Thing Seems Good!, otherwise the program will print Malicious Code Executed. Similiar to previous task, I used the VS-code to find address of A and B



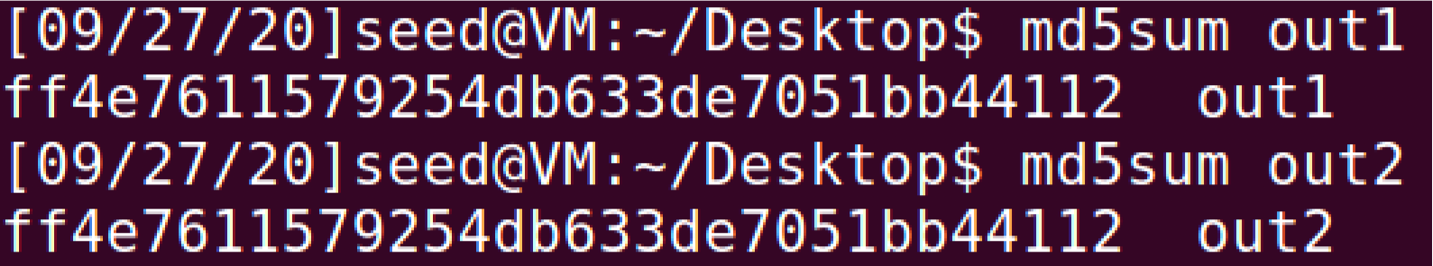
then we generate two prefix with the same MD5



then we copy the A array from output1 to B array and generate the suffix



Then we check the MD5 value and the output of these two programs:



Therefore, the same MD5 but behaves differently.

The full script that carry out this experiment are listed below:

1. gcc task4.c
2. ./a.out
3. head -c 12352 a.out > prefix
4. ./md5collgen -p prefix -o out1 out2
5. tail out1 -c 128 > suffix
6. tail a.out -c +12608 > temp
7. cat ./temp >> ./suffix
8. cp out1 p1
9. cp out2 p2
10. cat suffix >> p1
11. cat suffix >> p2
12. md5sum p1
13. md5sum p2
14. ./p1
15. ./p2