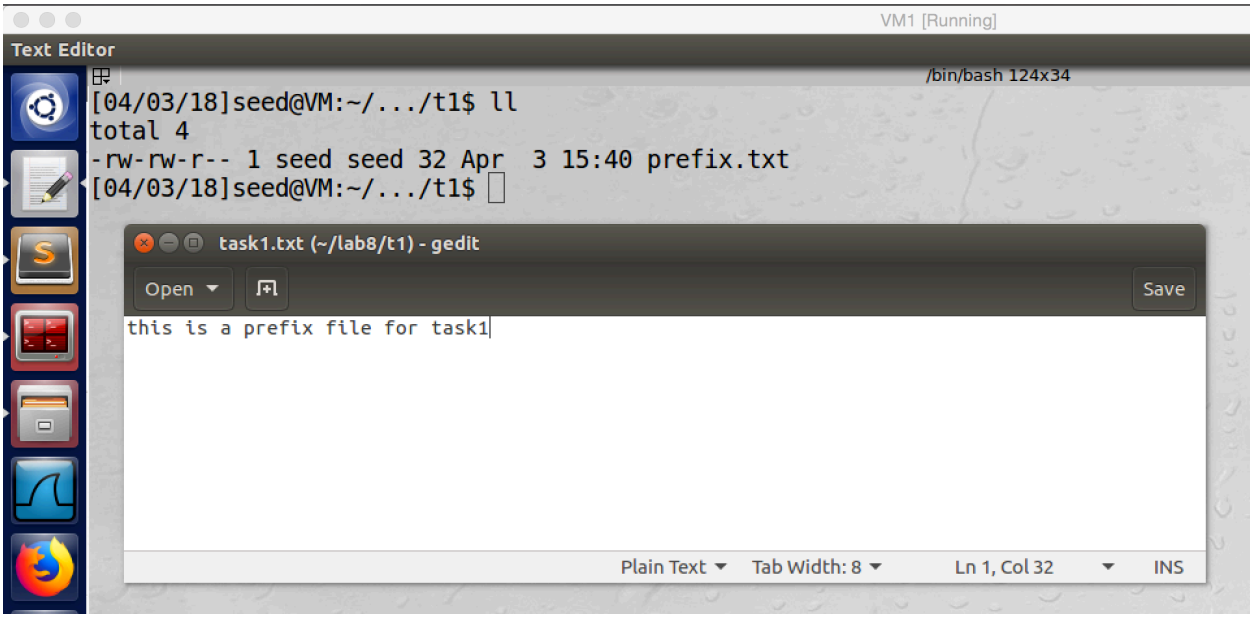
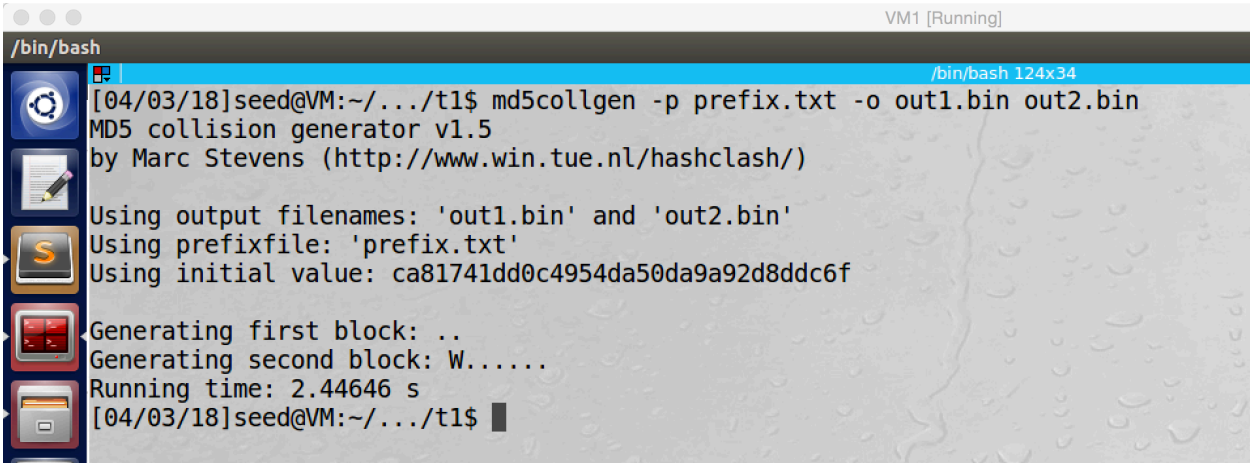
CSE644 Lab8
Yishi Lu
4/6/2018

In this lab, I use one VM which is VM1 with IP address 10.0.2.20.

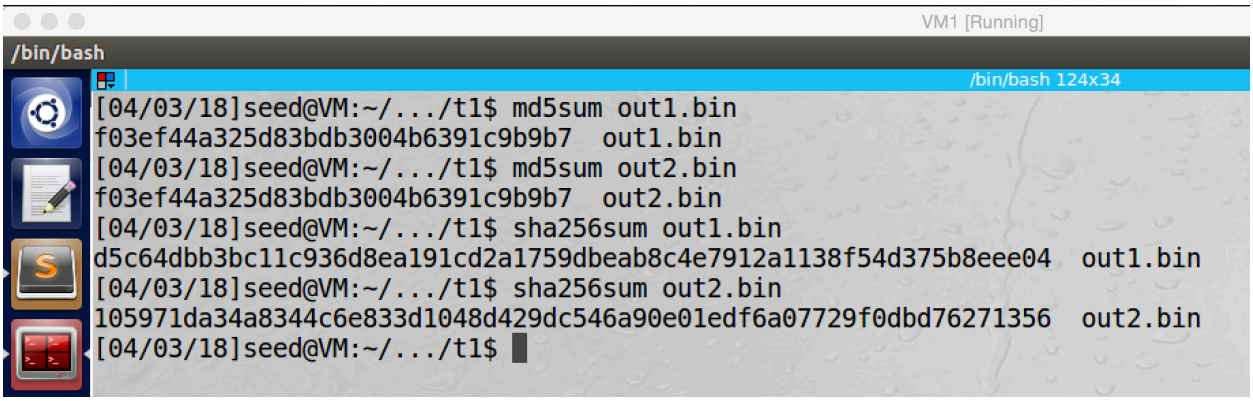
Task1: Generating Two Different Files with the Same MD5 Hash



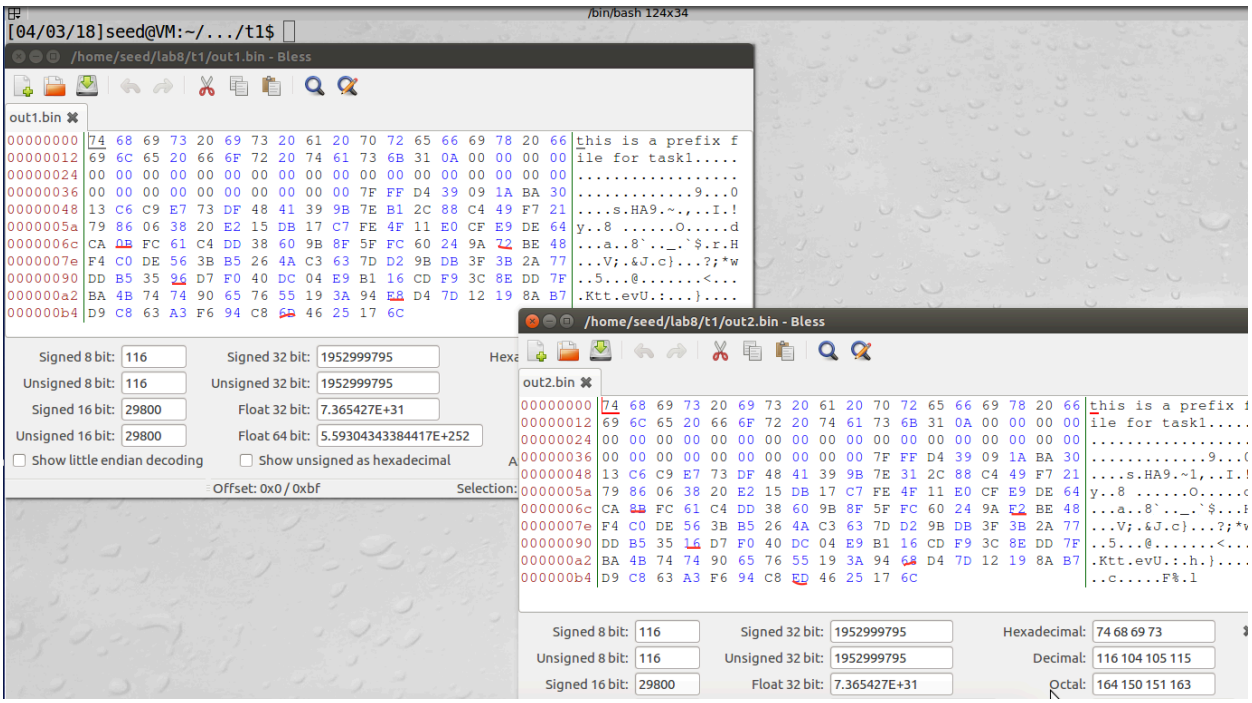
screenshot1. Creating a 32bytes prefix file



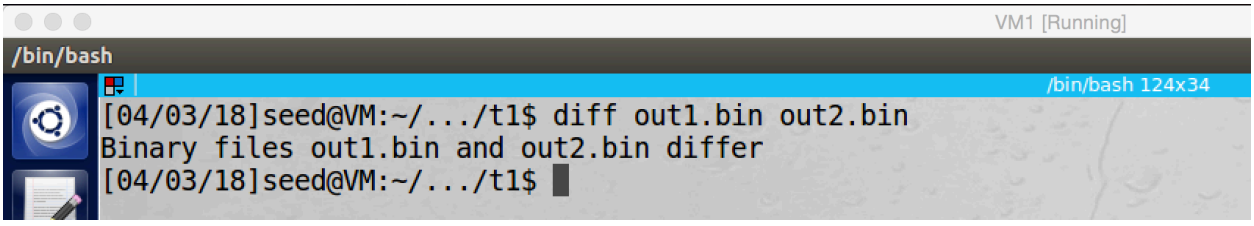
screenshot2. Using mad5collgen to create two output files out1.bin and out2.bin with same prefix file prefix.txt



screenshot3. We use command md5sum to produce MD5 hash value for out1.bin and out2.bin, they have exactly same hash value. We also use command sha256sum to produce SHA256 hash value for them, and they have different SHA256 hash value, which proves they are different files. So we successfully generate two different files with same MD5 hash value.



Screenshot4. If we open these two output files by hex editor, we can see the contents of these two files are similar. However, if we look at them more carefully, we find some bytes are different (marked by red underlines)

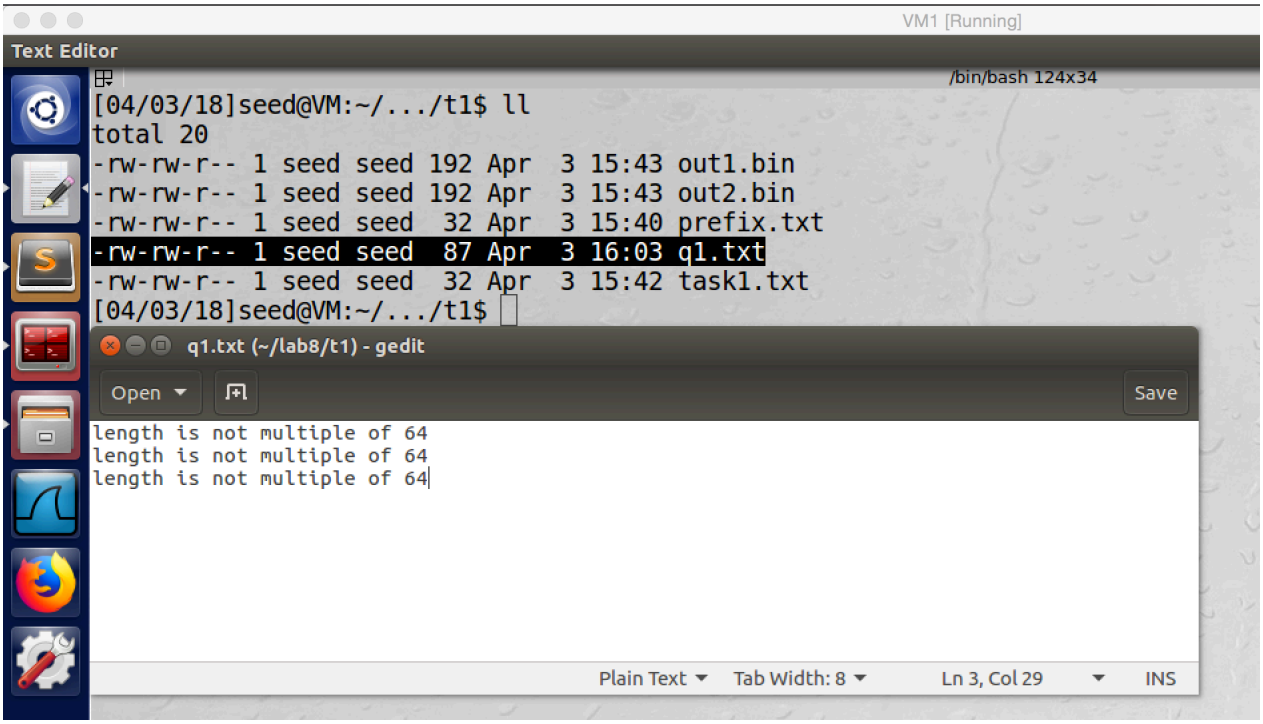


screenshot5. I use diff command to check their difference, they are totally different files

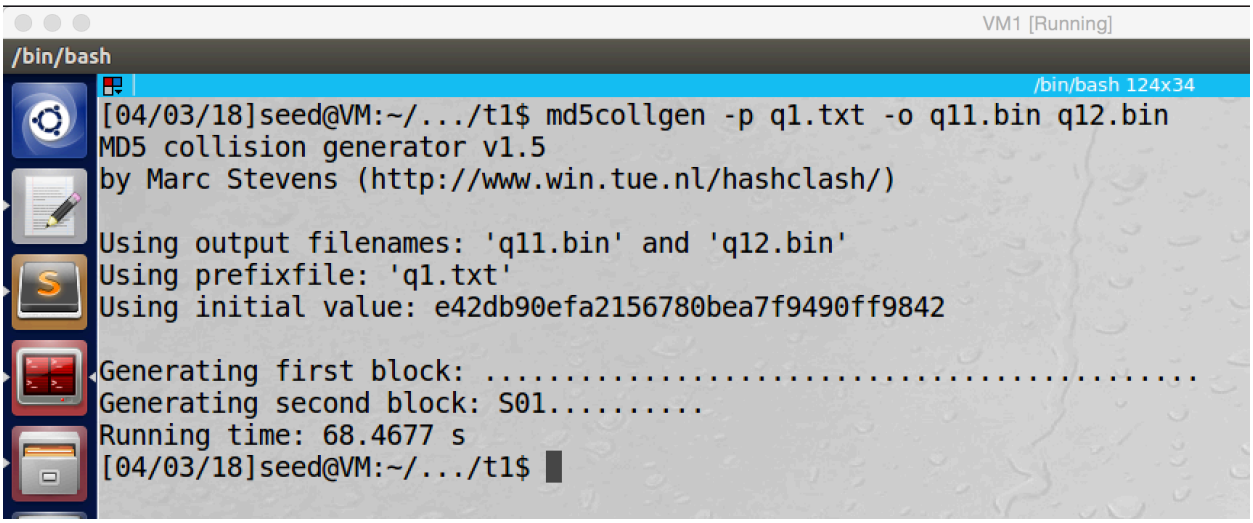
Observation and Explanation:

Firstly, we created a prefix file, and we save it as prefix.txt (screenshot1). And then we run command m5collgen command with the prefix file, this command creates two files, out1.bin and out2.bin, these two files should have same MD5 hash value (screenshot2). Afterwards, we need to verify these two file has same hash value, and they are different files. For first purpose, we run md5sum, which will generate MD5 hash value for input file. As screenshot3 shows, the MD5 hash value of out1.bin and out2.bin are same. For second purpose, we run sha256sum command, this command will generate SHA256 hash value. As screensshot3 shows, they have different SHA256 hash value, which means they have different contents. When we use hex editor to open them, we see there are a lot of consecutive 0 bytes, and the contents look like the same; however, if we check them carefully, we find some bytes are different (screenshot4). We also run diff command to check their difference, the result shows they are different file (screenshot5).

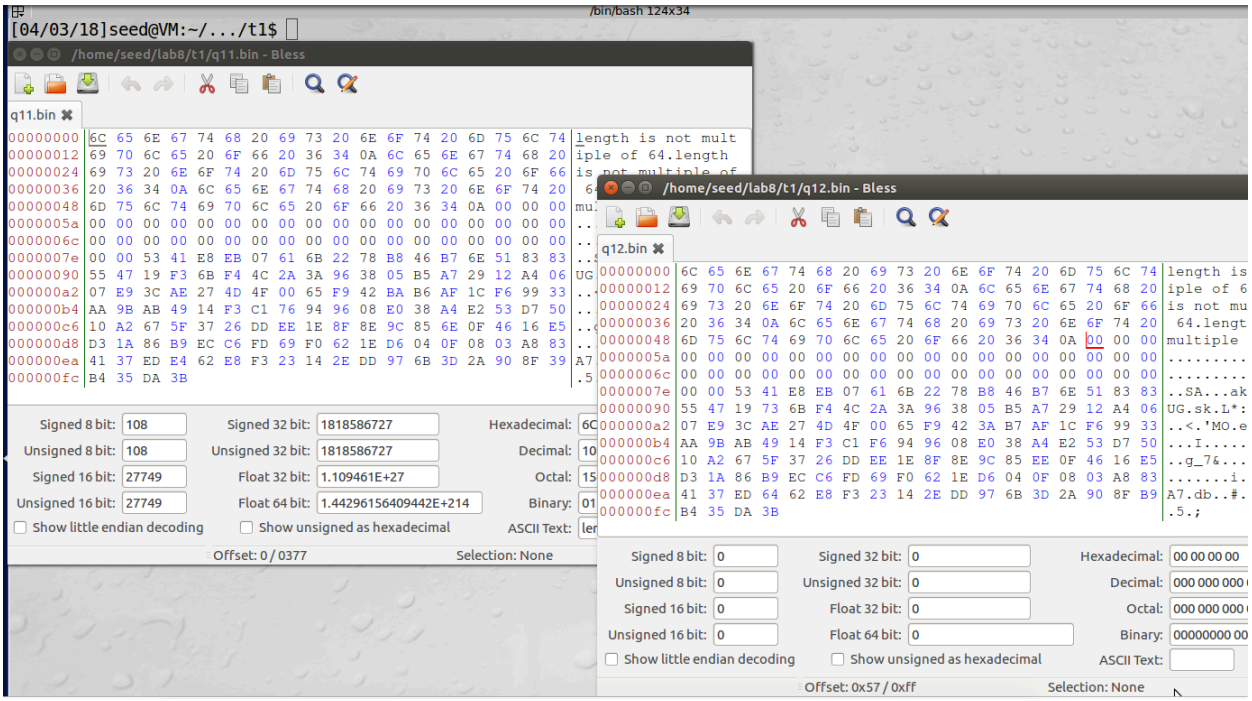
Question1:



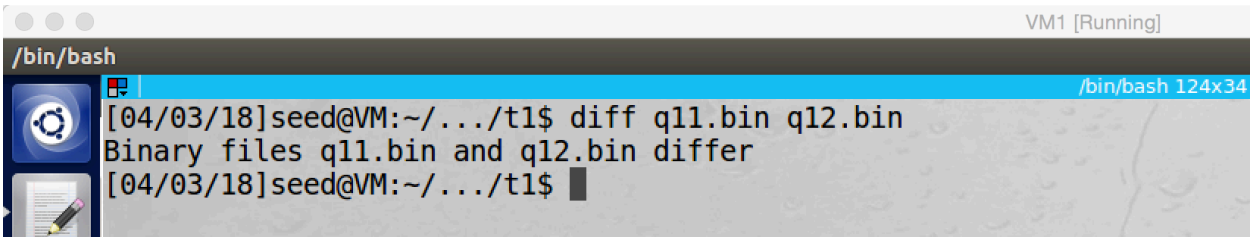
I create an 87bytes file, so its length is not multiple of 64, and it called q1.txt



Running md5collgen with the 87bytes file, creates two output files q11.bin and q22.bin



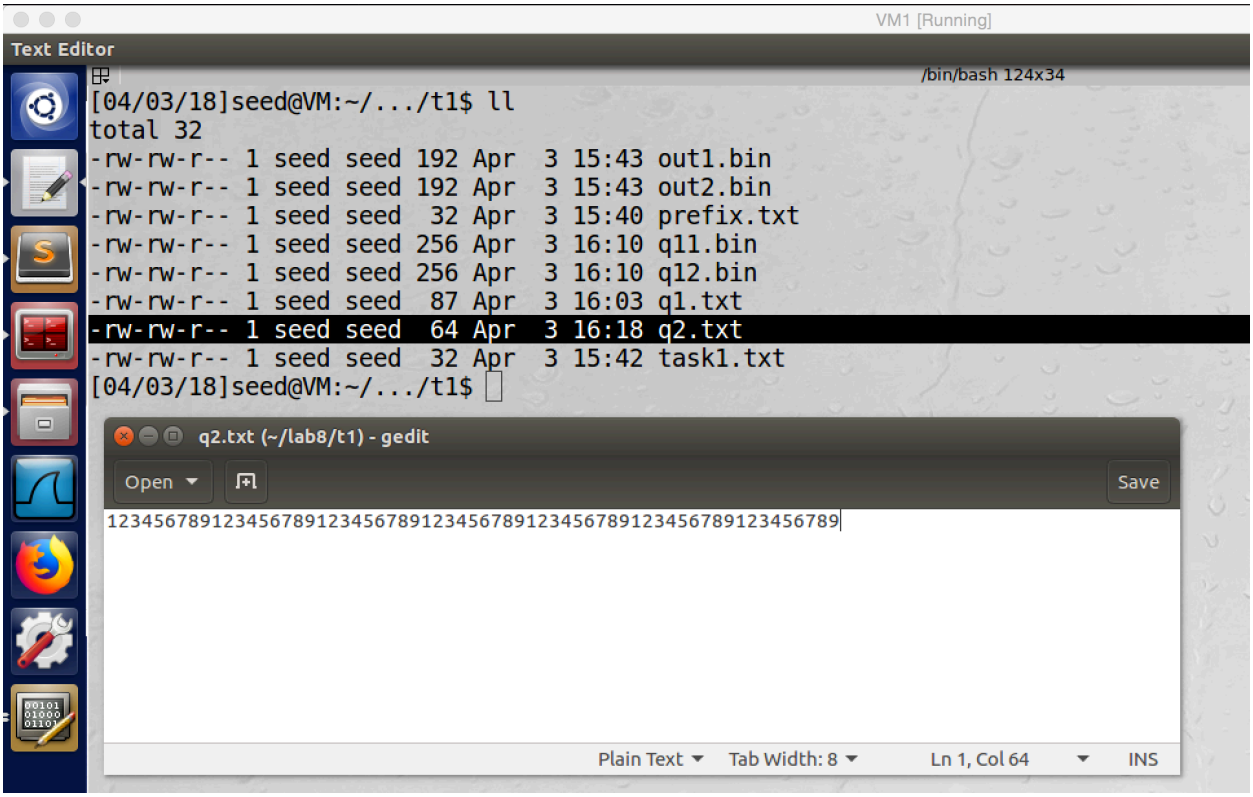
I open these two output files by hex editor, a lot of consecutive 0 bytes are in them.



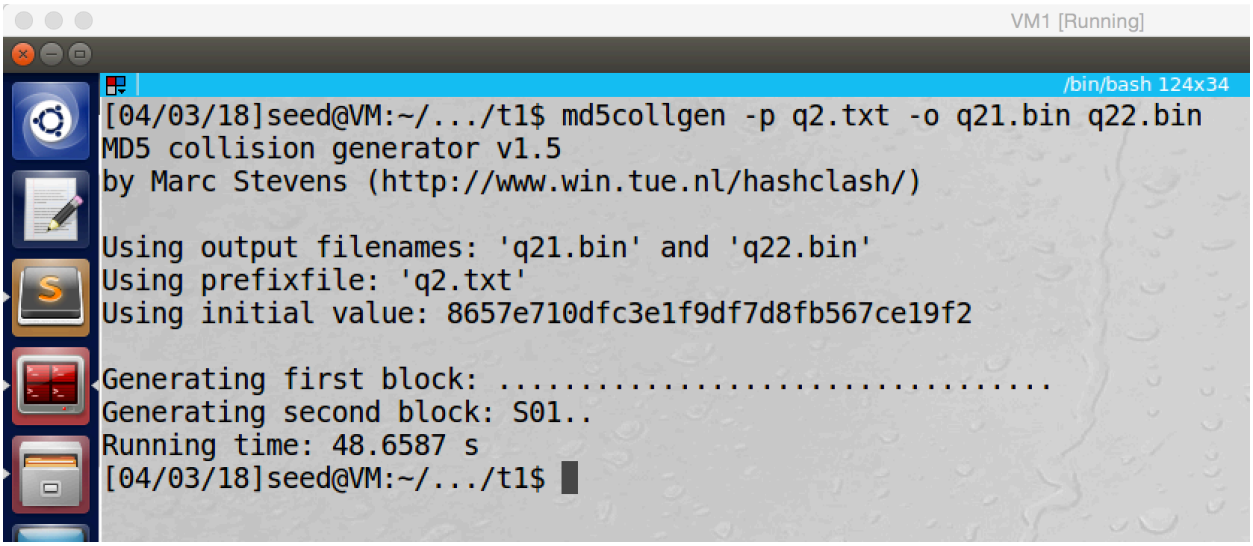
When I run diff command to check these two files, they are different file

In this question, we use 87bytes prefix file, and in task1 we use 32bytes prefix. Both of them are not multiple of 64. Moreover, the size of output files in this question are 256bytes. And the size of output files in task1 are 128bytes. So we can conclude, if the prefix file is not multiple of 64, then 0bytes will be used to fill up output files to be multiple of 64.

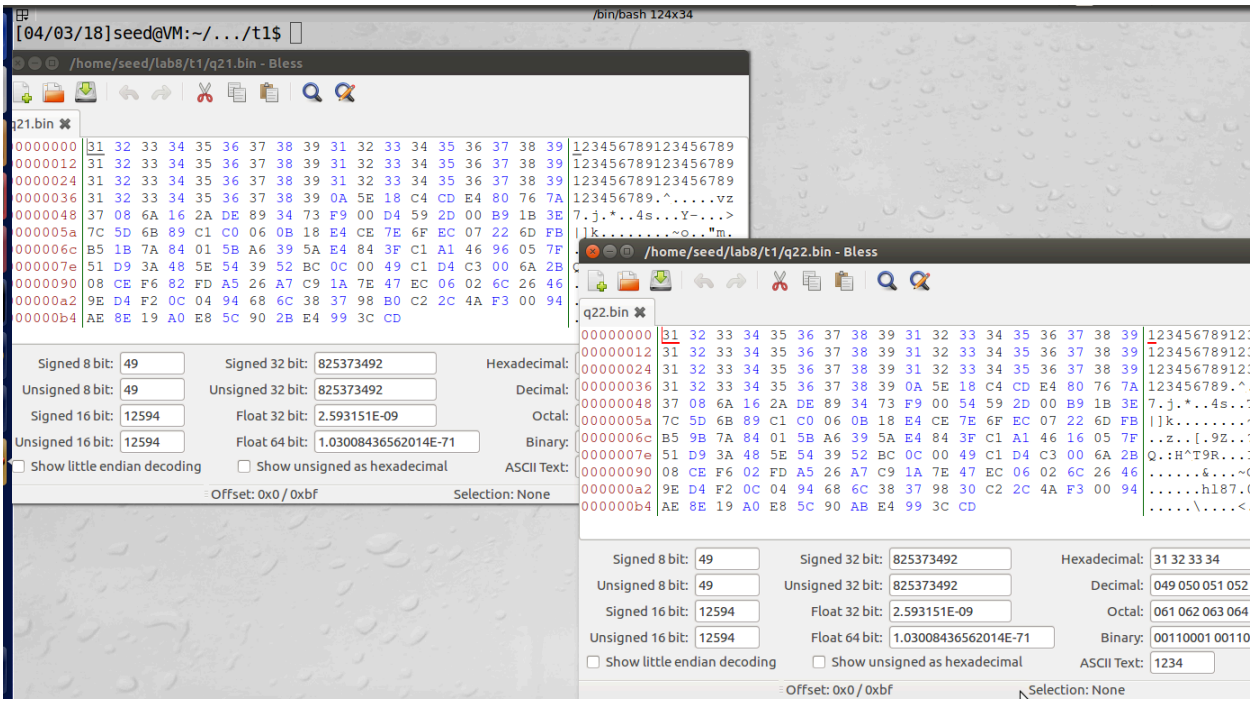
Question2:



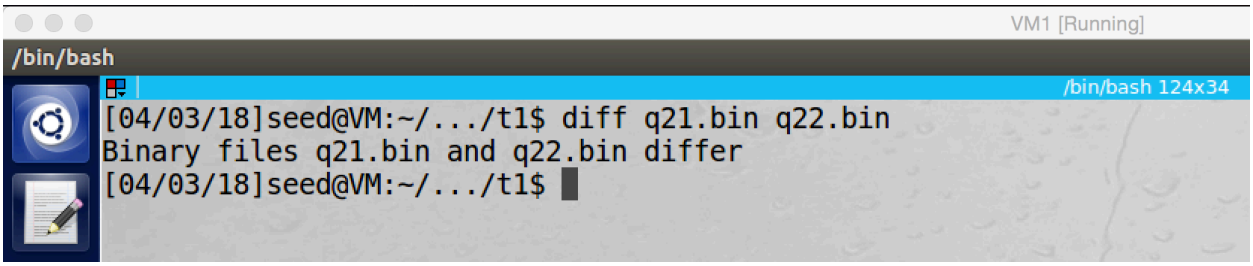
I create a 64byte prefix file: q2.txt



We use the 64bytes file as prefix to run md5collgen and creates two output files q21.bin an q22.bin



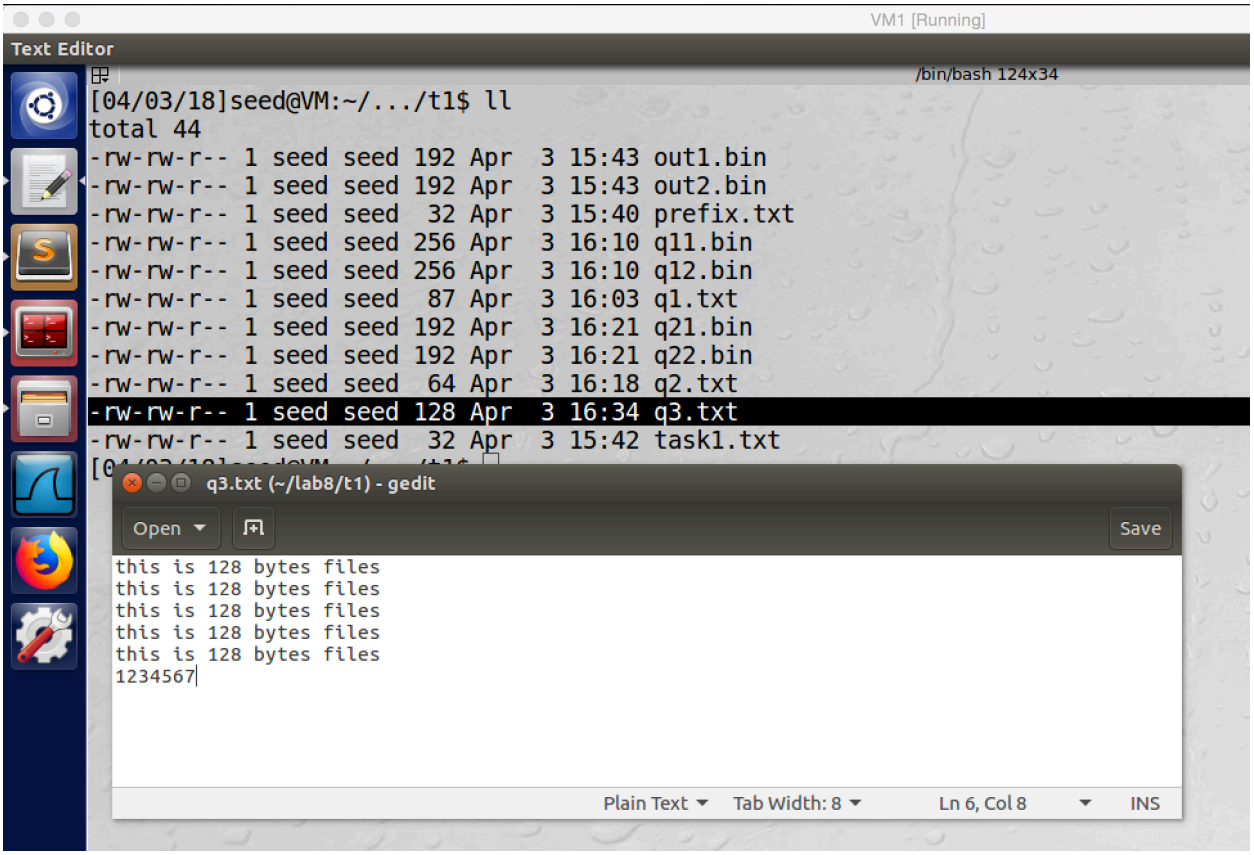
I open one output file with hex editor, there is no consecutive 0 bytes anymore.



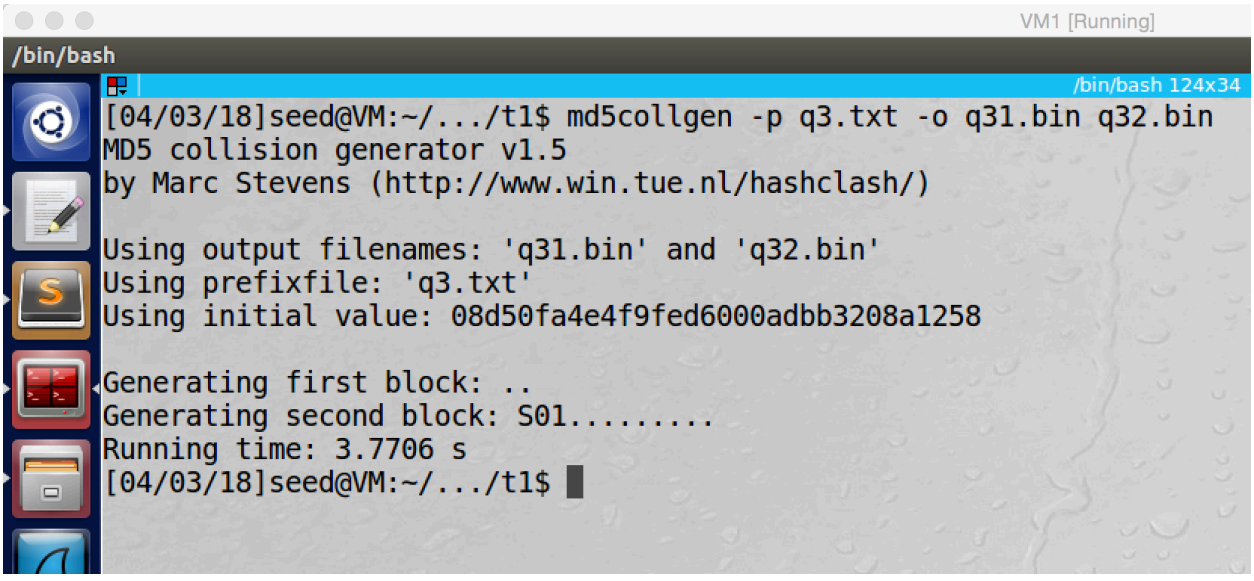
We use diff command to check, these two output files have different contents

In conclusion, if the prefix file is multiple of 64, there is no need to have consecutive 0 bytes in the output files to fill up the multiple of 64.

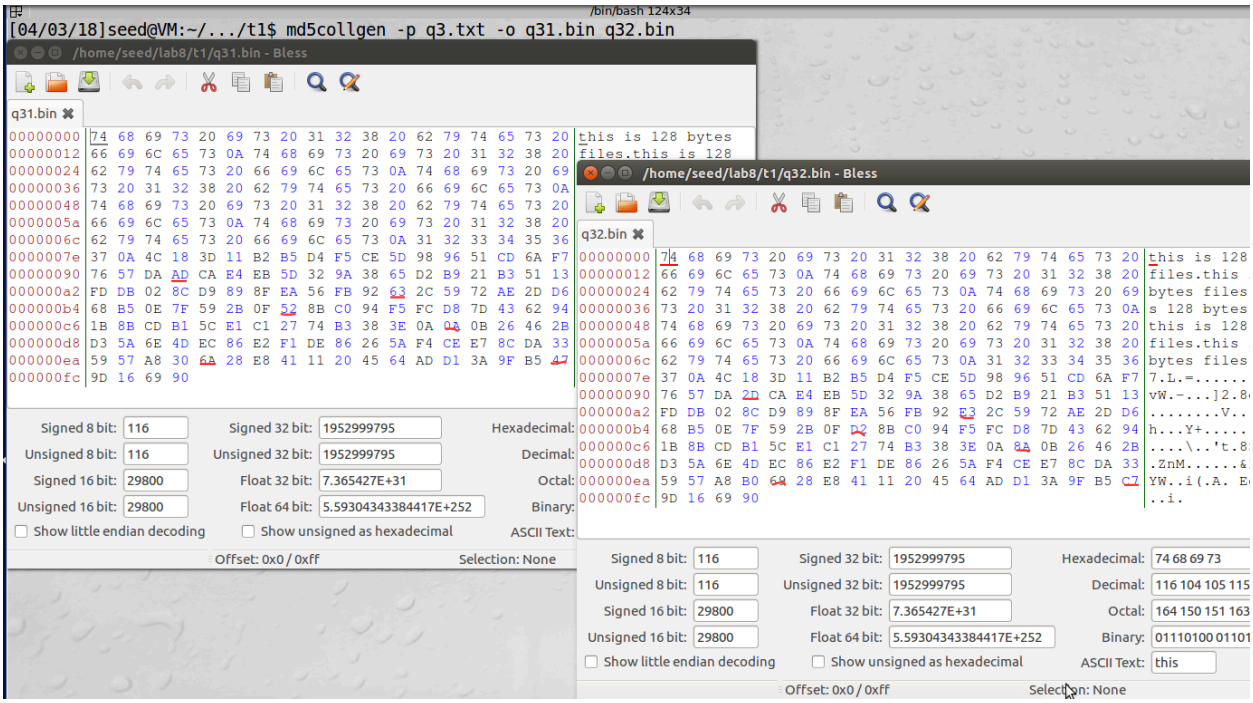
Question3:



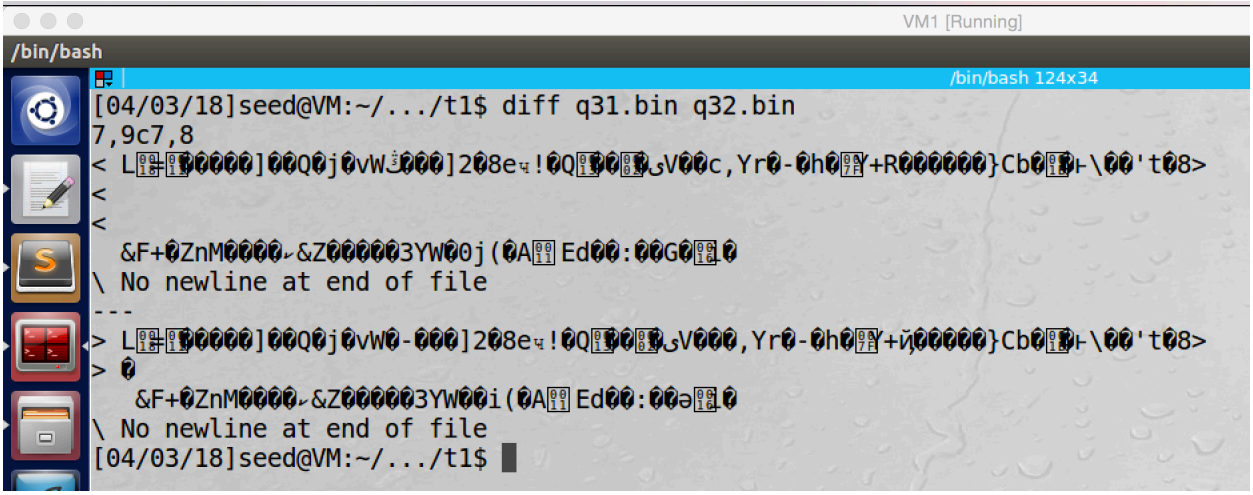
Created a 128bytes file, which called q3.txt



Generating two output files by md5collgen



I used hex editor to open these two output files, and I marked all different bytes by red underlines



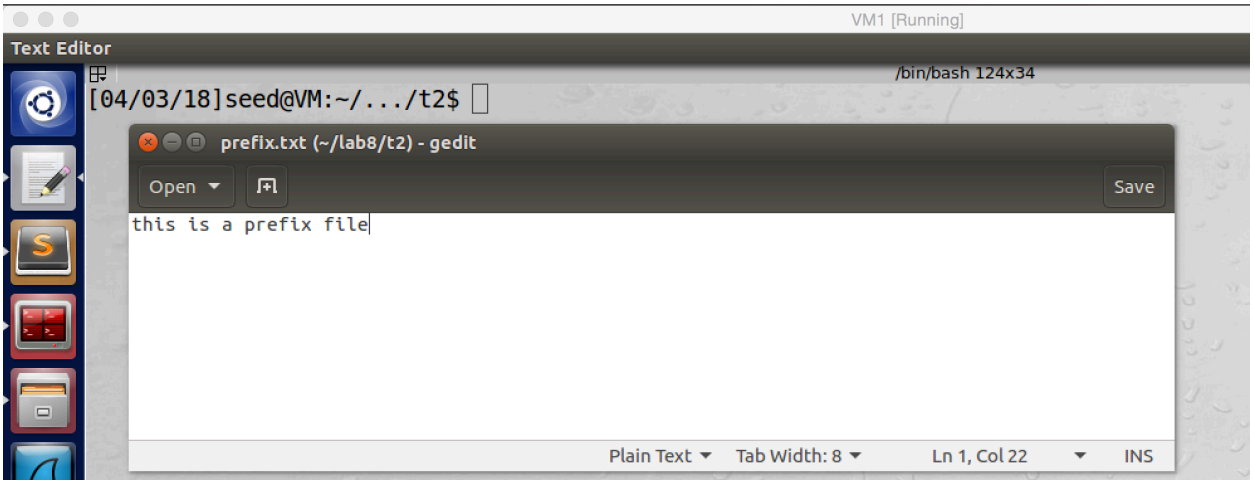
If we run diff command, these two files' contents are different, but some corrupted characters are printed

In conclusion, not all bytes generated by md5collgen are different for the two output files, but some bytes are different, I have marked them in the above screenshot by red underlines.

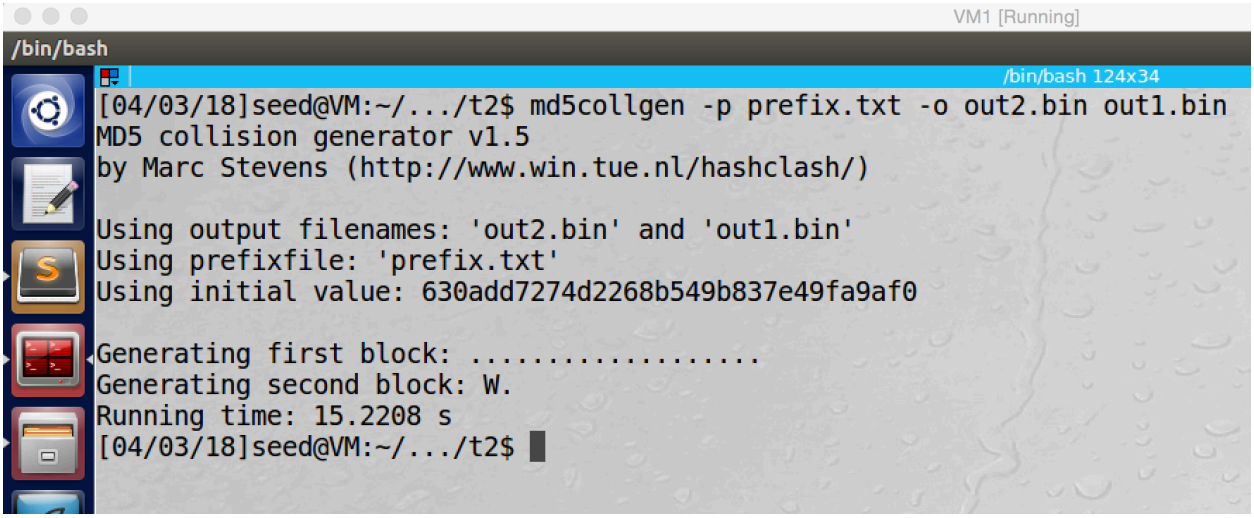
Task2: Understanding MD5's Property

In this task, we will use md5collgen to create two different files, but these two files have same MD5 hash value. And then we need to append same suffix file to them. If these two

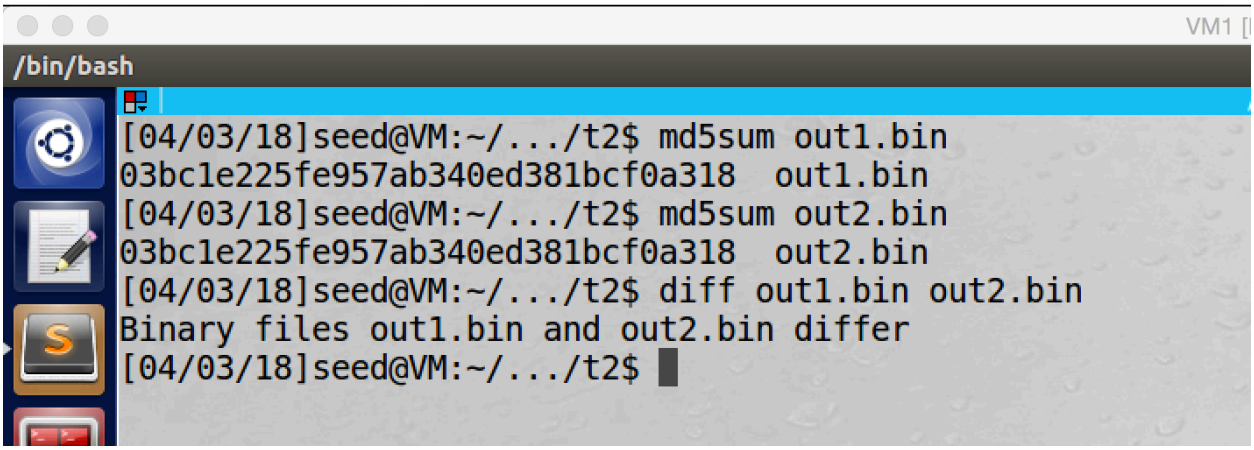
appended files have same MD5 hash value, then we demonstrate that MD5 holds the length extension property.



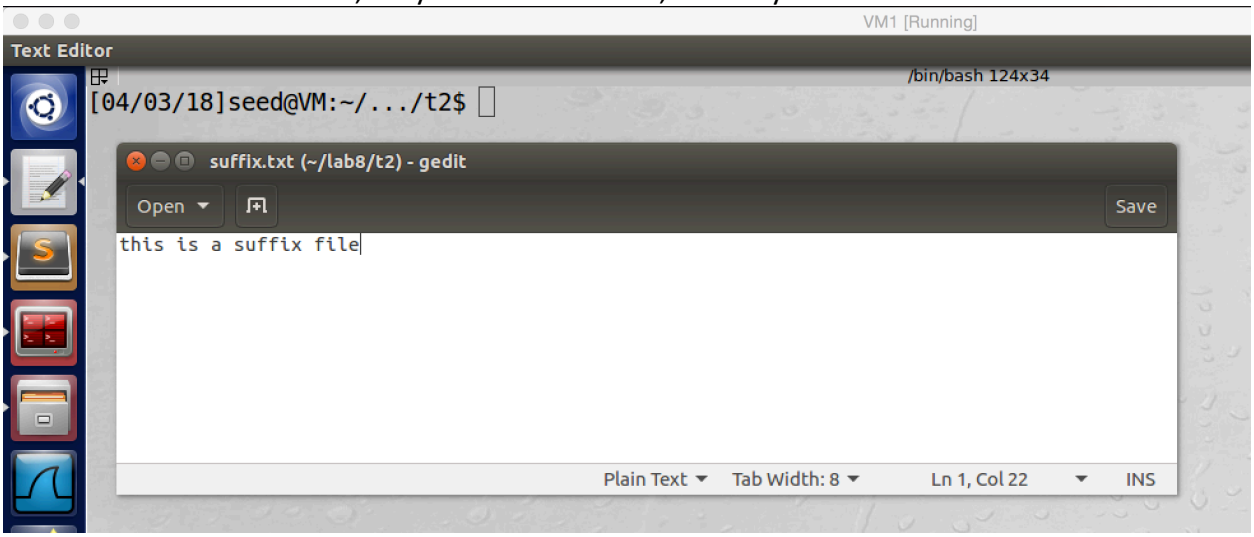
screenshot1. We create a prefix file



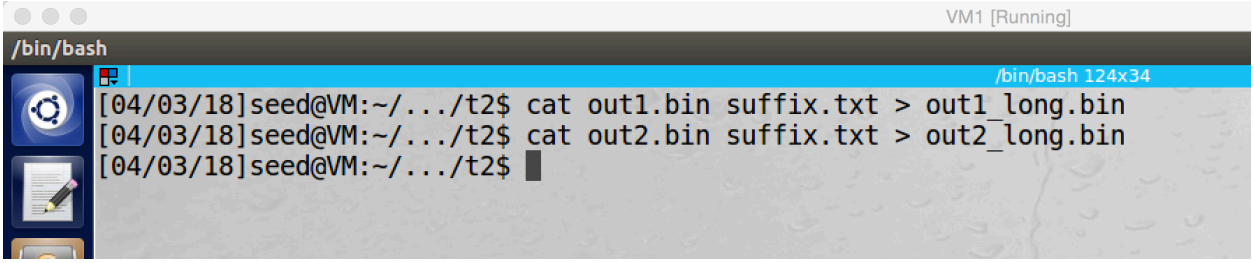
screenshot2. Using md5collgen to generate two different files (out1.bin and out2.bin) with same MD5 hash value



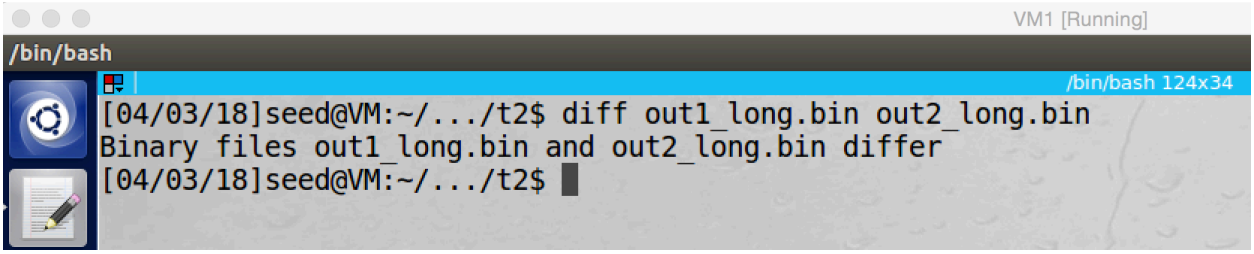
screenshot3. We test them, they are different files, but they have same MD5 hash value



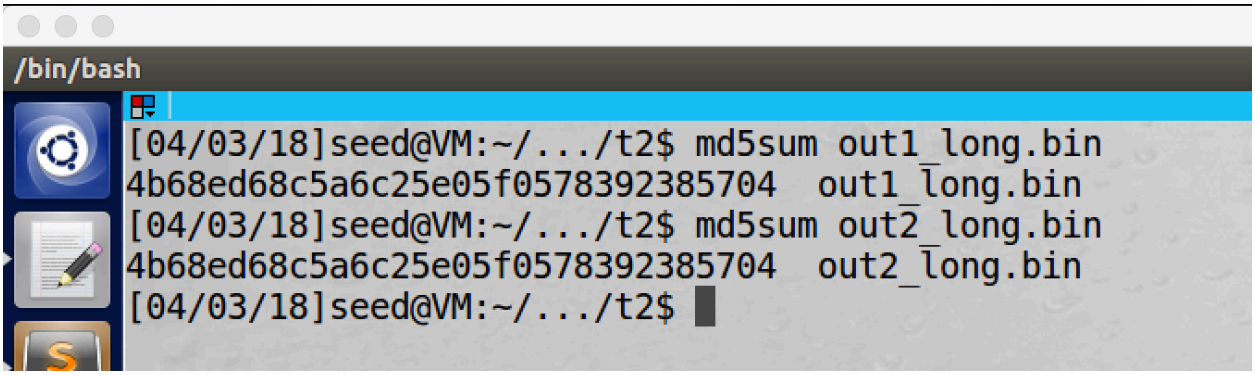
screenshot4. Creating a suffix file



screenshot5. Appending the suffix file to out1.bin and out2.bin. Now we get two new file, out1_long.bin and out2_long.bin



screenshot6. Running diff command to show these two files have different contents



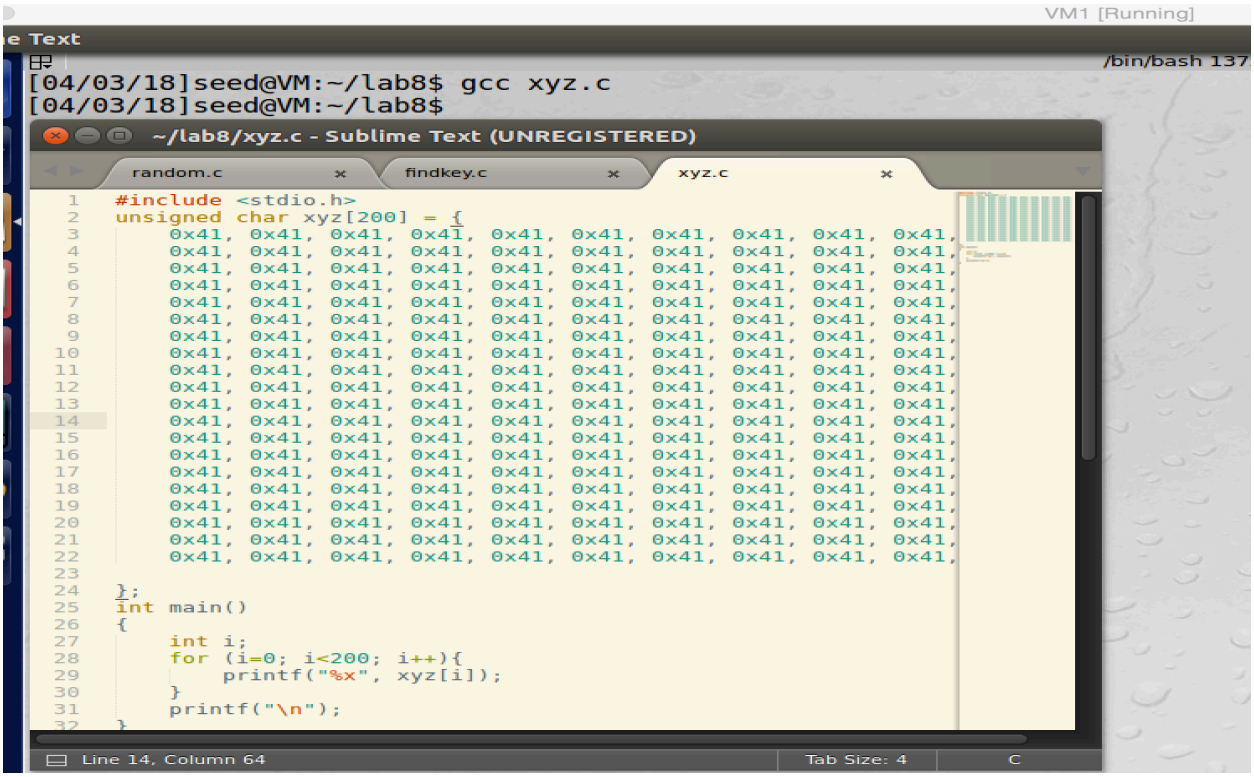
screenshot7. Running md5sum, these two appended files have same MD5 hash value, so MD5 holds the length extension property

Observation and Explanation:

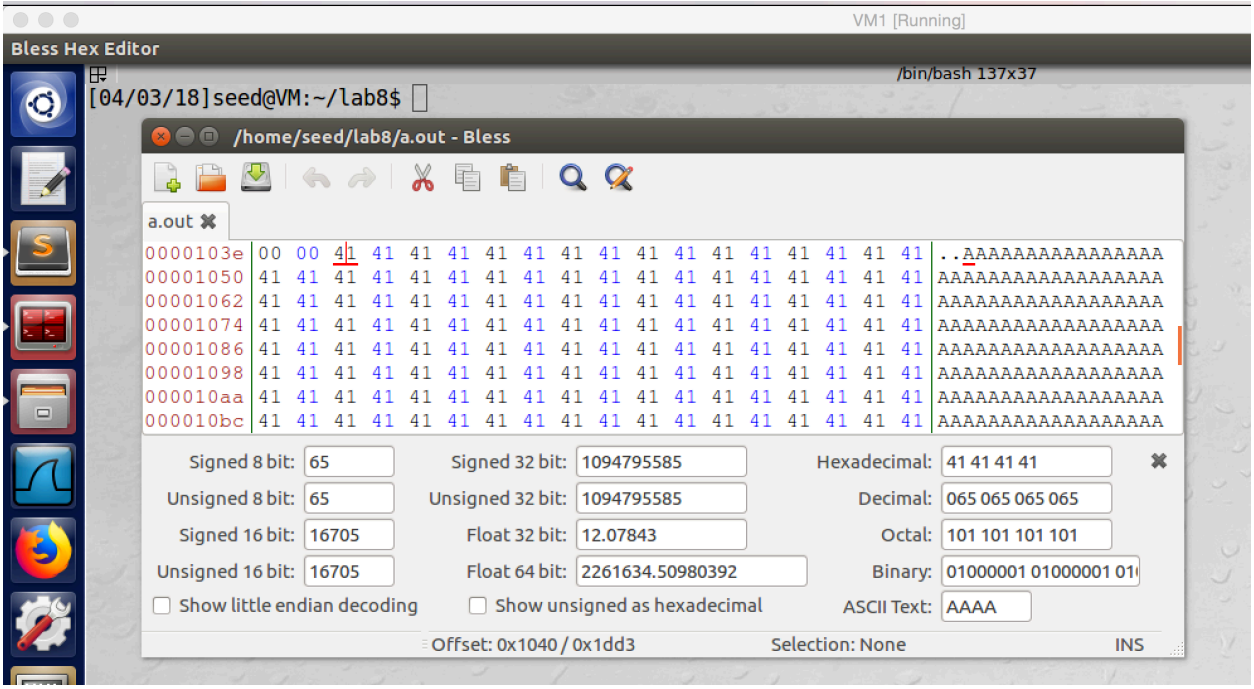
In this task, we want to show that MD5 holds the following property, if MD5(M) = MD5(N), then MD5(M || T) = MD5(N || T) should be true for any T, and this property called length extension.

We first create a prefix file (screenshot1), and then we feed this prefix file to the command md5collgen to generate two output files, out1.bin and out2.bin (screenshot2). As screenshot3 shows, they are different files, but they have same MD5 hash value. And then we create a suffix file (screenshot4). And we append suffix.txt to out1.bin and out2.bin, and we got two new files out1_long.bin and out2_long.bin(screesnshot5). And then we check if these two new files are different or not by using diff command, and the answer is they are different (screenshot6). Afterwards, we run md5sum to get MD5 hash value for both of them. As the screenshot7 shows, these two new files have same MD5 hash value. So we demonstrated that MD5 holds the length extension property.

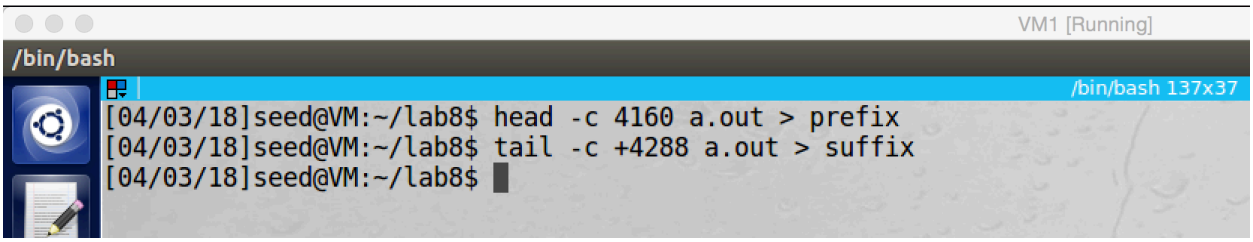
Task3: Generating Two Executables Files with the Same MD5 Hash



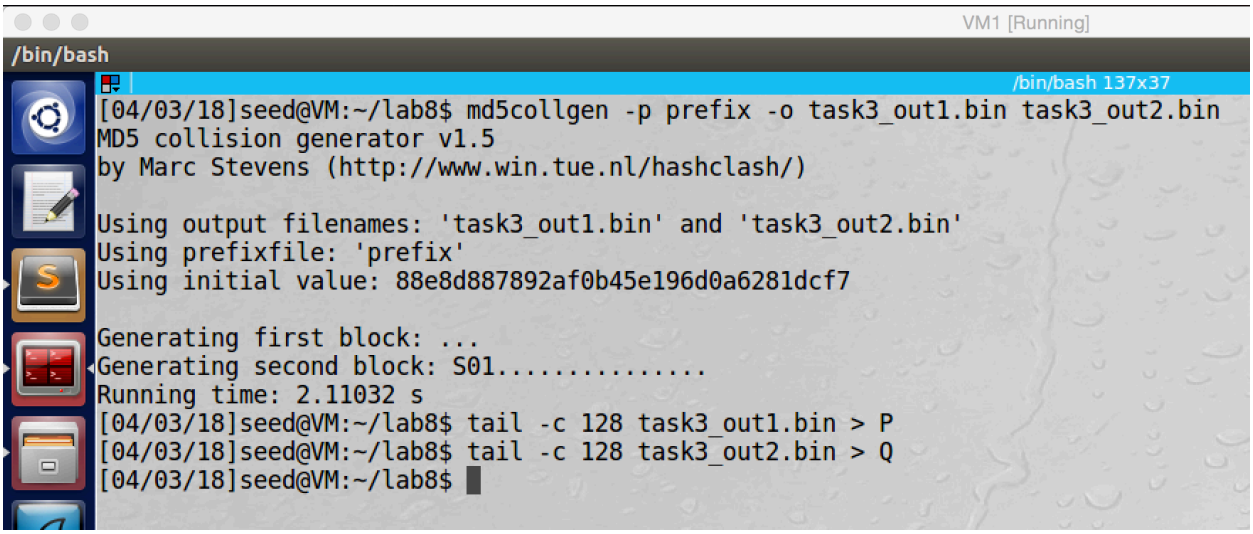
screentshot1. Creating a C program, xyz.c, and compiling it, so we get executable file a.out



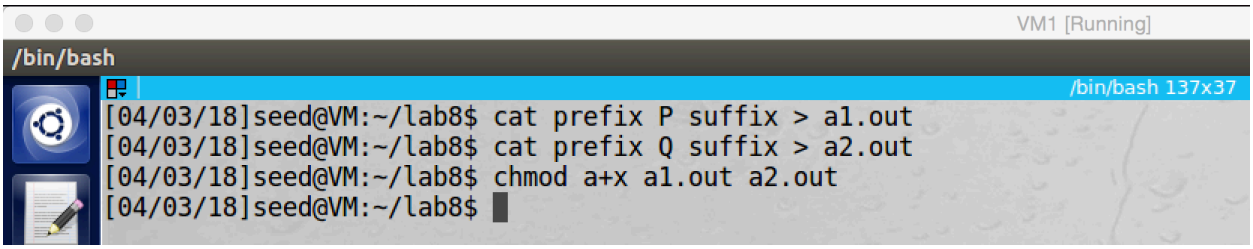
screenshot2. After compiling the C program, we open the executable file a.out by hex editor bless, and we can find the array, it starts at offset 4160, which is multiple of 64



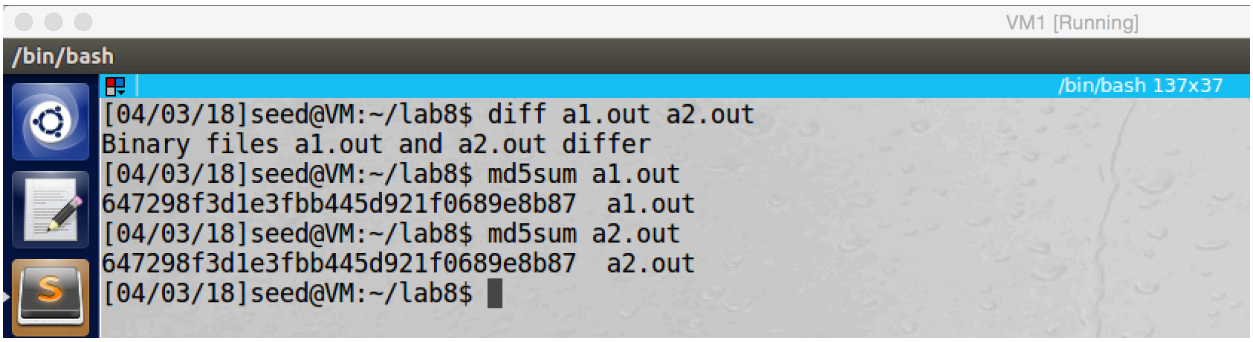
screenshot3. We want to skip 128bytes of the array in the program, so we store the head (first 4160bytes of the file) into prefix. We store the tail (content after skipping 128bytes of the array) into suffix



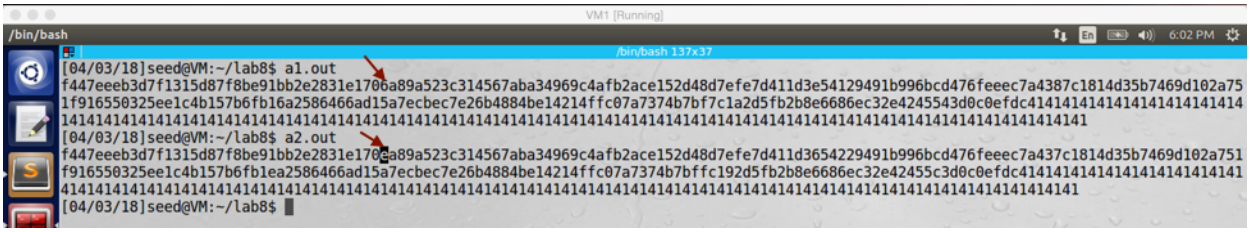
screenshot4. We run md5collgen with the prefix which we got before to generate two files, task3_out1.bin and task3_out2.bin. And then we take the last 128 bytes from these two files and store them in P and Q.



screenshot5. We concatenate prefix P/Q suffix all these parts together to get two new files a1.out and a2.out. Moreover, the third line is used to make these two new files executable.



screenshot6. We check these two files' content is different by using command diff. And then we also run md5sum to get their MD5 hash value, and they have same hash value.



screenshot7. These two files (a1.out and a2.out) are also executable. After we run them, we can see the outputs are almost same, but there are still some bytes are different. For example, I marked one on the screenshot, the "**e**" and the "**6**".

Observation and Explanation:

In this task, we want to generate two different program with same MD5 hash value. In our case, we will write program, such program will print content of an array. Our goal is to have two similar programs, but their array contents are different; in other words, they can print different things. But their MD5 hash value are same.

Firstly, we create such C program: xyz.c, and then we compile it to get a.out (screenshot1). The program can print the content of its array xyz, and this array is initialized to be 200's 0x41. 0x41 is the ASCII value for A, so we can easily find the position of the array when we open a.out by hex editor. As screenshot2 shows, we open the file a.out by hex editor, and we located the array's position which is 4160. Then we run head and tail command to get the prefix and suffix, because the array starts at 4160, so the first 4160 bytes of a.out are the prefix; and because we want to replace 128bytes of the array, so the bytes after 4288 (4160 + 128) are the suffix (screenshot3). After we got the prefix, we can use the md5collgen to generate two file with same MD5 hash value. And then we take the last 128byes from these two files and save them into P and Q (screenshot4). Now we have got prefix, P, Q, and suffix, we need to concatenate them to get two new files, a1.out and a2.out(screenshot5). These two files should be different contents, but they have same MD5 hash value. This is because the length extension property, which we demonstrated in the task2. MD5 (prefix || P) and MD5 (prefix || Q) have same hash value, so by the property, MD5 (prefix || P || suffix) and MD5 (prefix || Q || suffix) should have same MD5 hash value with any suffix. We verify this in screenshot 5 and 6. In screenshot5, we first run diff command to see if these two files are different, and the result shows a1.out and a2.out are different files. And then we run md5sum to get MD5 hash value of

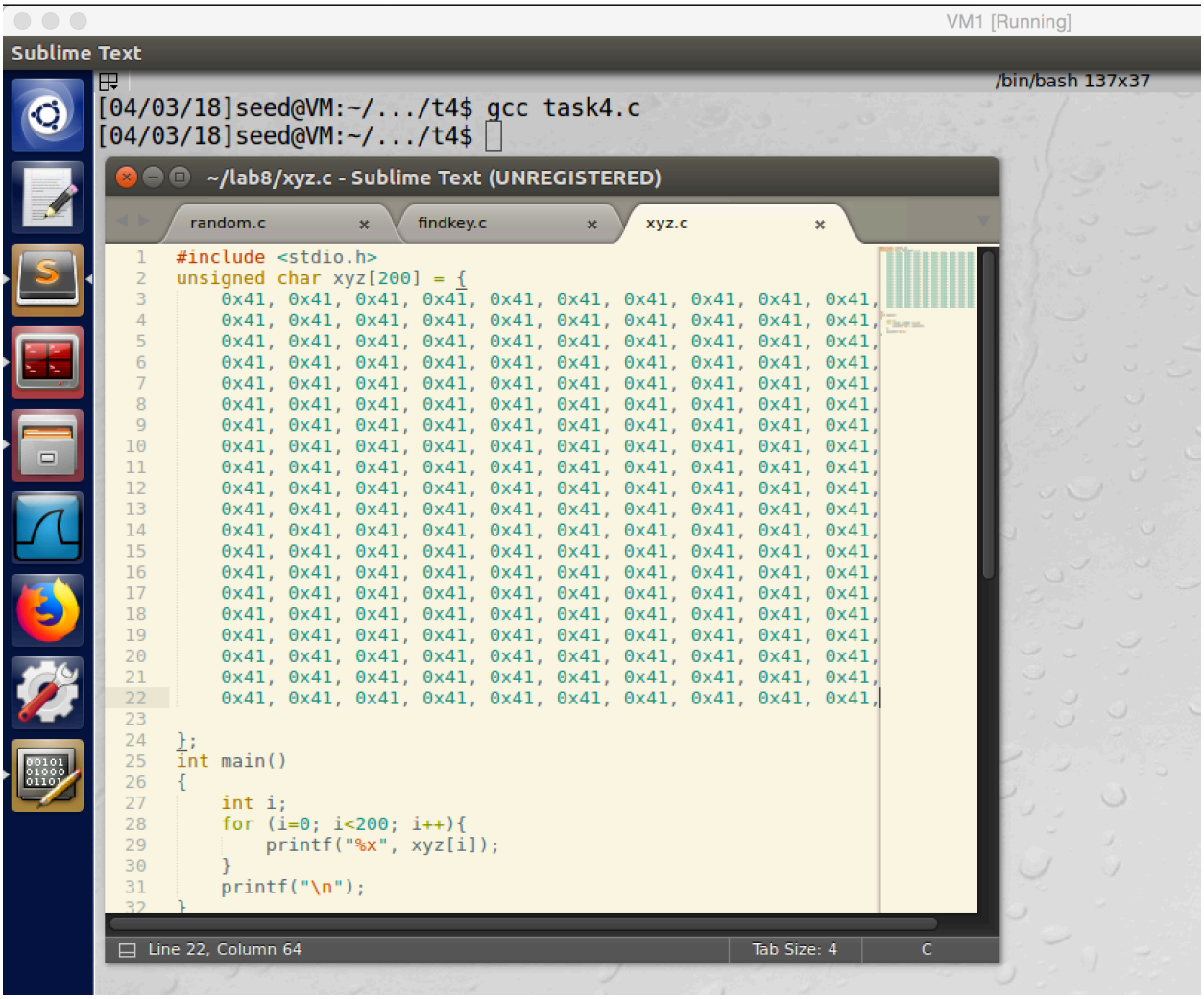
them, and they have same MD5 hash value. Finally, we also run these two programs. Because we just changed their array, the program should be executable. As the screenshot6 shows, they print the content of their array. Because we only changed few bits of their array, so the result seems similar. However, if we examine the result carefully, we can find some bytes are different; for example, one byte is marked on screenshot6

The C program xyz.c which is used in this task

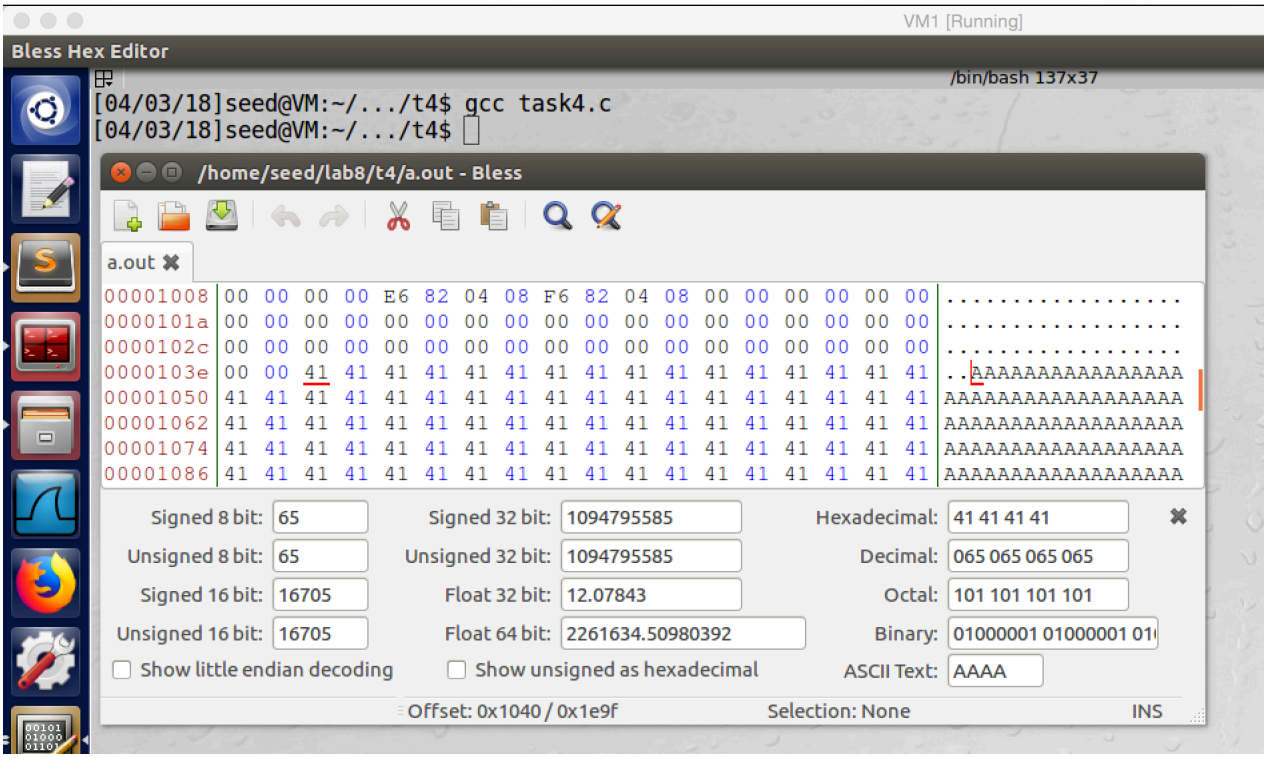
```
#include <stdio.h>
unsigned char xyz[200] = { //there are two 200 0x41.
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,

};
int main()
{
        int i;
        // print the array contents
        for (i=0; i<200; i++){
                printf("%x", xyz[i]);
        }
        printf("\n");
}
```

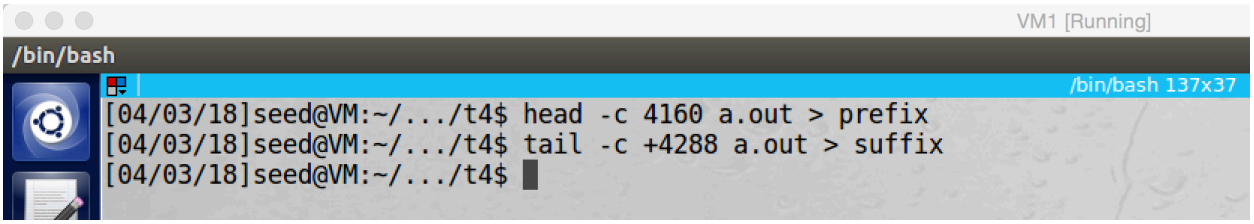
Task4: Making the Two Programs Behave Differently



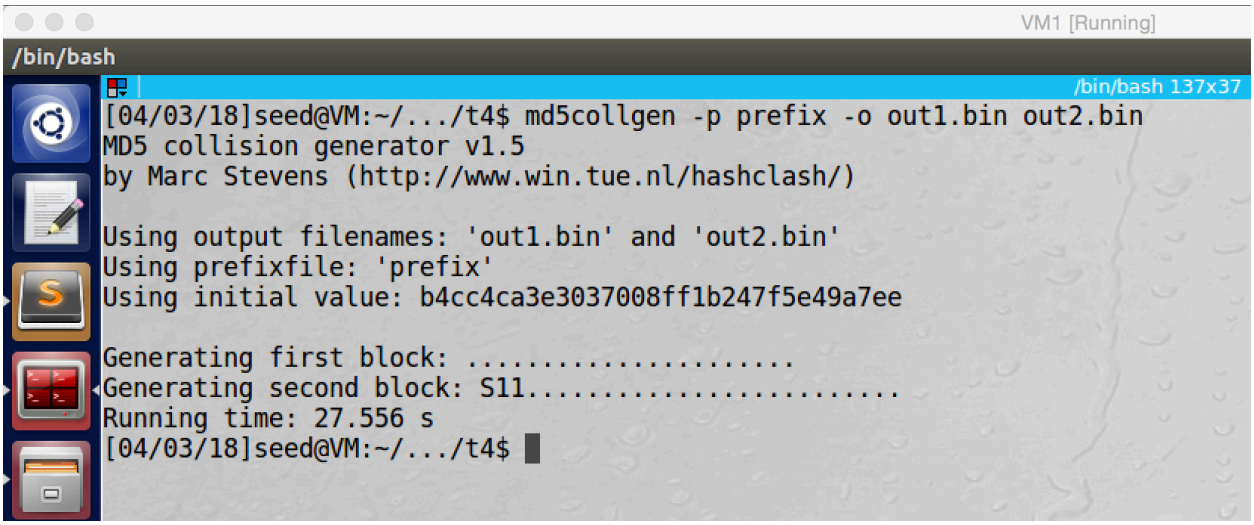
screenshot1. Creating C program task4.c; after compiling it, we got a.out



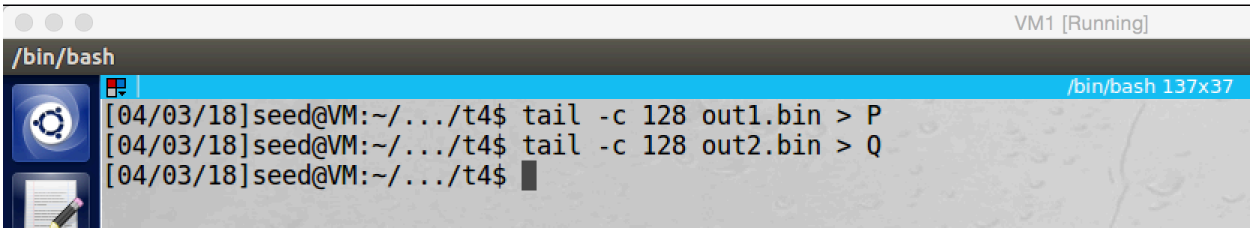
screenshot2. We open the executable file a.out by hex editor, and we see that the array is started at 4160.



screenshot3. We divided a.out into two parts, prefix and suffix. The first 4160 bytes are saved in to prefix, and the contents from 4288 to the end are saved into suffix. The middle 128bytes of array1 is skipped

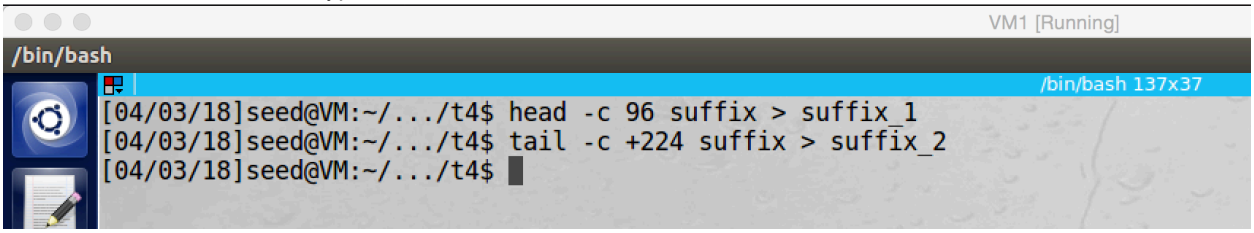


screenshot4. Using md5collgen with prefix.txt to generate two output files out1.bin and out2.bin, they are different files with same MD5 hash value

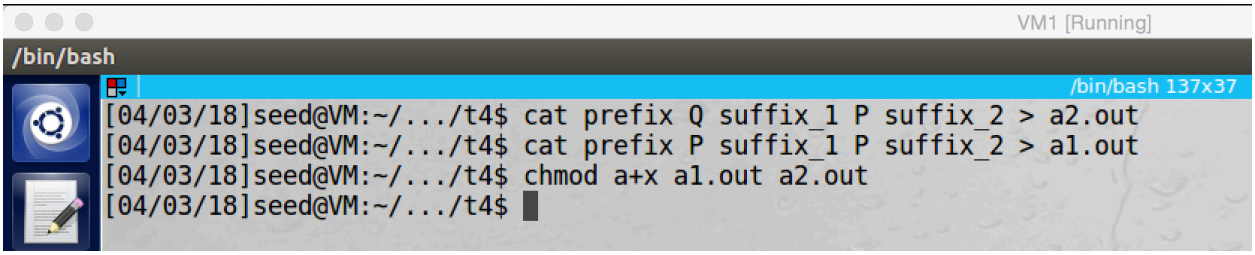


screenshot5. Saving the last 128bytes of out1.bin and out2.bin into P and Q

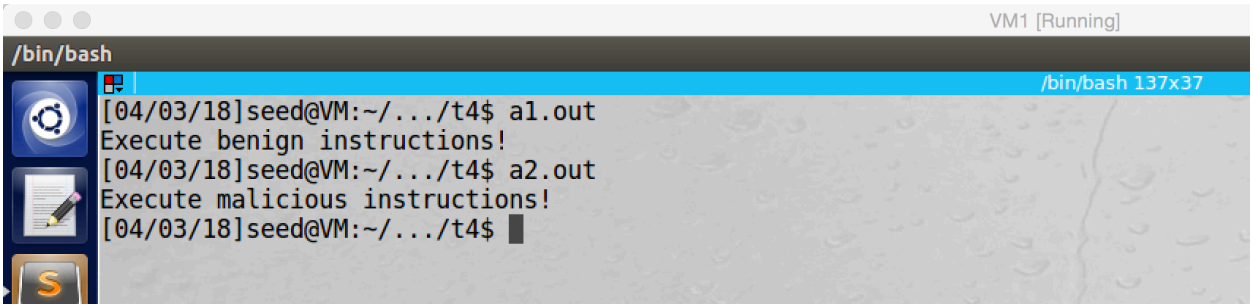
Because we want create two programs, first program has two same arrays (executes benign codes) and second program has two different arrays (executes malicious codes). So two arrays in the second program should be different. Therefore, we need to modify the suffix (suffix contains the second array).



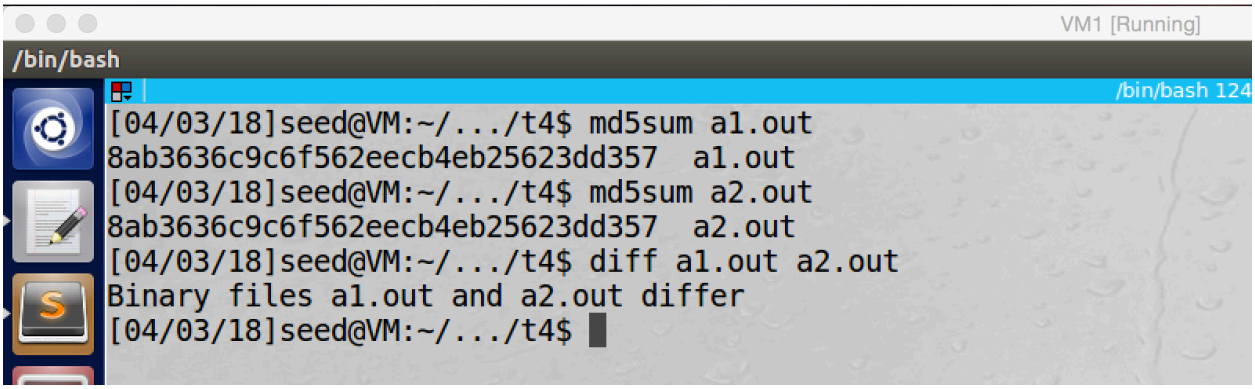
screenshot6. We divide the suffix into 2 parts. First part is the first 96 bytes, and the second part is from 224 to the end (128bytes are removed, which will be replace by P)



screenshot7. We concatenate all pieces together. For a1.out (prefix || P || suffix1 || P || suffix2), because we put P in both array1 and array2, the contents of these two array are same; so it should execute benign instruction. For a2.out (prefix || Q || suffix1 || P || suffix2), because we put Q in the first array and P in the second array, the contents of these two array are different; so it should execute malicious instruction.



screenshot8. After we concatenate all pieces together, we can execute a1.out and a2.out. a1.out executes the “benign instructions”, and a2.out executes the “malicious instructions”



screenshot9. We run the md5sum command to generate MD5 hash value for a1.out and a2.out. Both of them have exactly same MD5 hash value, our attack is successful.

Observation and Explanation:

We have a program, the program has two array, array1 and array2. if the content of array1 and array2 are same, the if statement will be true, so benign codes will be executed. Otherwise, the if statement is false, so malicious code in the else clause will be executed. In this task, we want to create two similar program. One has two same array, so benign code will be executed. The other one has two different array, so malicious code will be executed. The most important thing is that these two programs have exactly same MD5 hash value. To do this task, we should use the length extension property of MD5, which we demonstrated in the task2.

If P and Q are generated by md5collgen with same prefix, then they have same MD5 hash value. Therefore, MD5 (prefix || P) = MD5 (prefix || Q). With the length extension property, MD5 (prefix || P || suffix) = MD5 (prefix || Q || suffix) should be true for any suffix. If we divide the suffix into two part and insert P or Q in the middle, which is MD5 (prefix || P || suffix1 || P ||suffix2) and MD5 (prefix || Q || suffix1 || P || suffix2), the MD5 hash value of them should be same as well. This is the mechanism which we used in this task.

First we write and compile the C program, and we got a.out (screenshot1). And then we use hex editor to open it, so we find that the first array is started at 4160 (screenshot2). And then we divide the file into two part, prefix and suffix. The middle 128byes will be replaced by P or Q (screenshot3). Then we run md5collgen with prefix to generate out1.bin and out2.bin, they have same MD5 hash value (screenshot4). And then we take the last 128bytes of out1.bin and out2.bin as P and Q (screenshot5). Afterwards, we also divide the suffix into two parts, first 96bytes, and bytes from 224 to the end. The middle 128bytes of suffix will be replaced by P (screesnhot6). Finally, we put all pieces together to form two new files, a1.out and a2.out (screenshot7). For a1.out (prefix || P || suffix1 || P || suffix2), because we put P in both array1 and array2, the contents of these two array are same; so it should execute benign instruction. For a2.out (prefix || Q || suffix1 || P || suffix2), because we put Q in the first array and P in the second array, the contents of these two array are different; so it should execute malicious instruction. Now we need to test them. By running a1.out, benign codes are executed. By running a2.out, malicious codes are executed (screenshot8). We also run md5sum to generate their MD5 hash value, and they have exactly same MD5 hash value; otherwise, we also run diff to verify they are different files, the answer is they are different (screenshot9). So our attack succeeds.

C program task4.c which is used in this program

```
#include <stdio.h>
#define LENGTH 200

unsigned char a[] = { //array1 contains 200 0x41 initially
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
};

unsigned char b[] = { // array2 contains 200 0x41 initially
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
```

```
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
        0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41, 0x41,
};

int main()
{
        int i = 0;
        //check if array1 and array2 are same
        for(i = 0; i<LENGTH; i++){
                if(a[i] != b[i]) break;
        }

        if(i == LENGTH)
                // if they are same, then run benign code
                printf("%s\n", "Execute benign instructions!");
        else
                //if they are different, then run malicious code
                printf("%s\n", "Execute malicious instructions!");

        return 0;
}
```