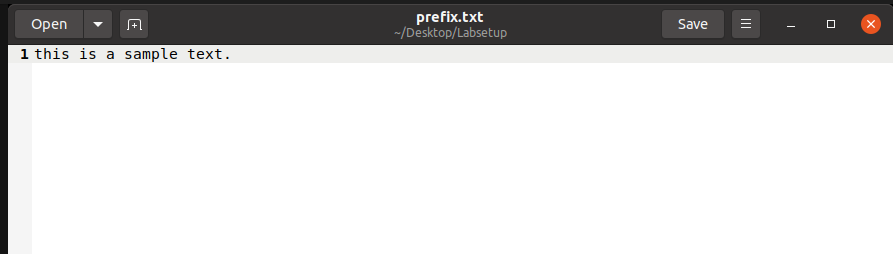
**ACS 54500 Cryptography and Network Security – Lab 9**

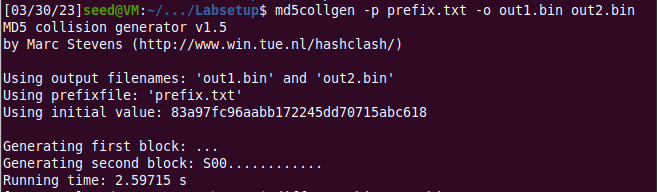
**Task 1:**

In this task, we need to generate two different files with the same MD5 hash values. The beginning parts of these two files need to be the same, i.e., they share the same prefix. This can be achieved using md5colgen.

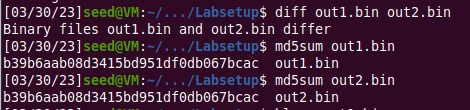
First, we need to create a sample prefix text file. This is what I chose to put in mine -



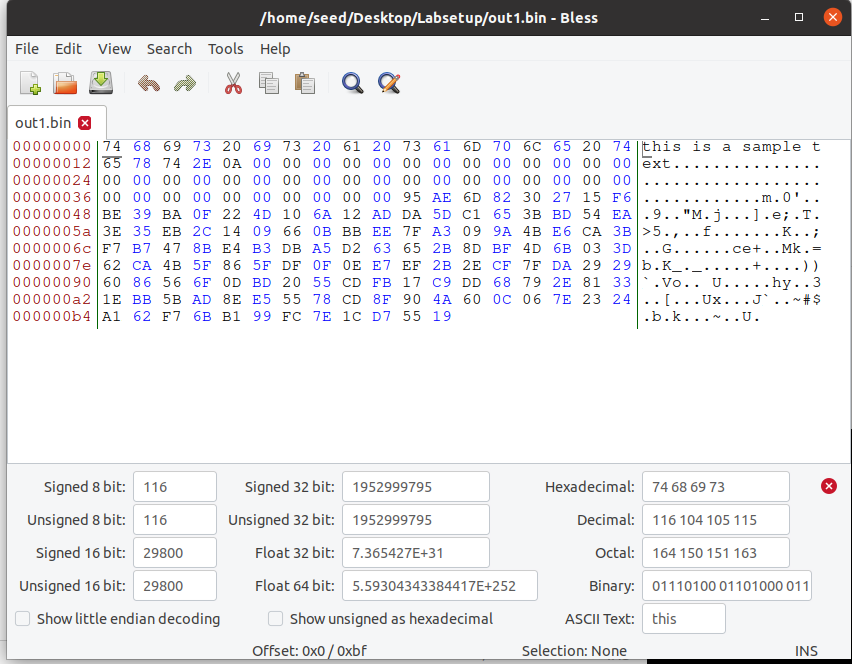
Now, we run the following command to generate two output files, out1.bin and out2.bin, for a given a prefix file prefix.txt –



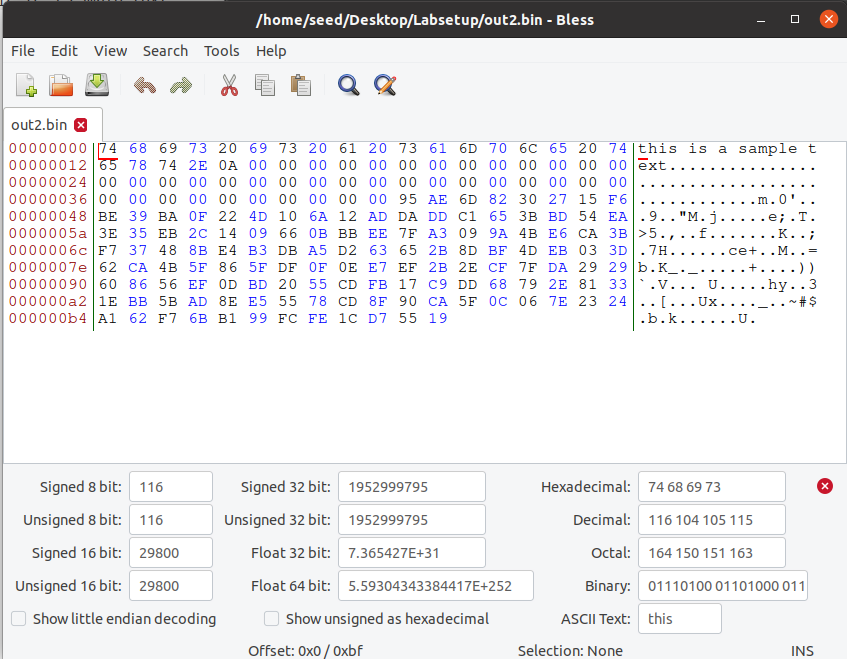
The diff command checks whether the output files are distinct or not. So, the two files are distinct. We can also use the md5sum command to check the MD5 hash of each output file.



Now we run bless on each output file -



out1.bin



out2.bin

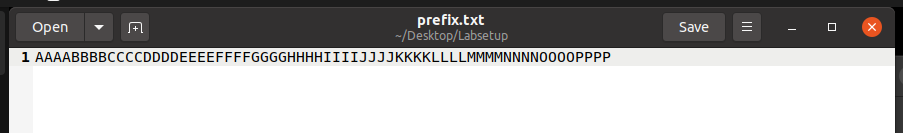
We can tell that a few offset locations are different when comparing both files, i.e. for example, the offset locations of 0123, 0155, 0156 etc. are different, even when they have the same MD5 hash value.

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

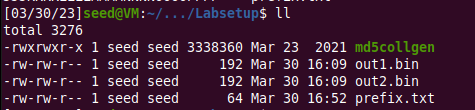
If the length of the prefix file is not a multiple of 64, then md5collgen adds in padding to make the length a multiple of 64.

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

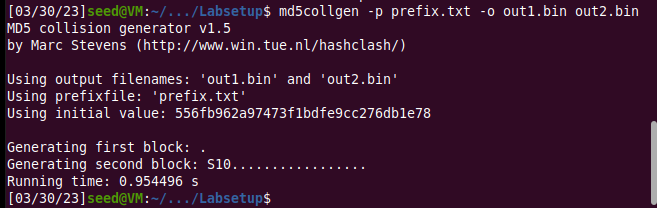
In this question, we use echo to change the contents of the prefix file to make the length 64 bytes.



Here is a screenshot showing that the length is 64 bytes –

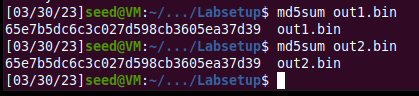


Now we run the same commands again to generate two output files for this prefix text file, and check if they differ.

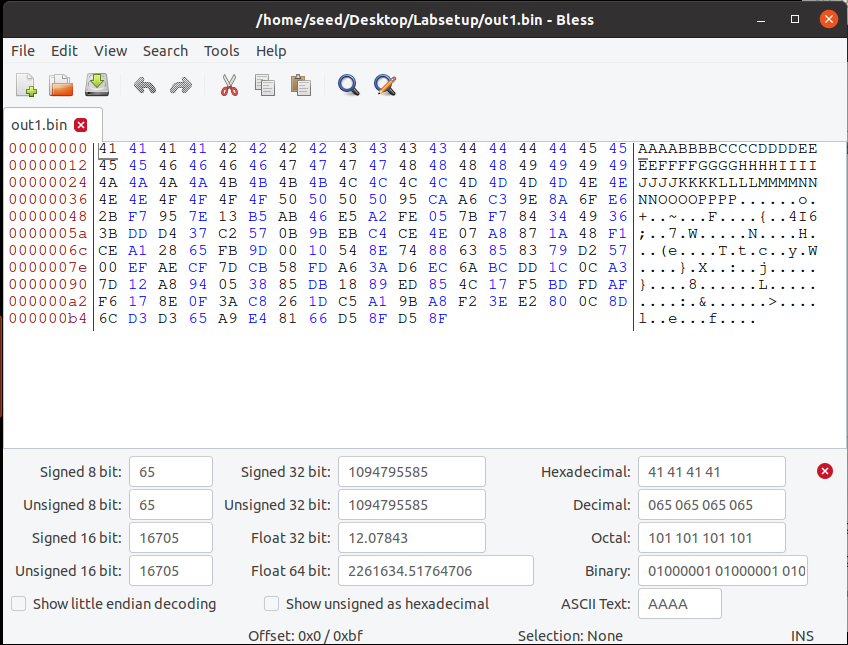




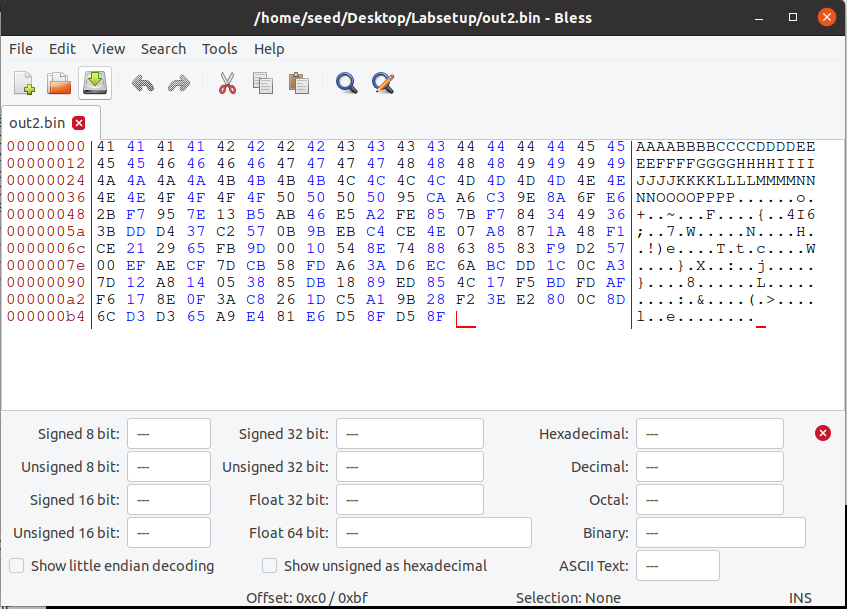
So, the two files are different. Then we generate the md5 hash value for both using md5collgen.



Now, we use bless to view both the binary files –



out1.bin



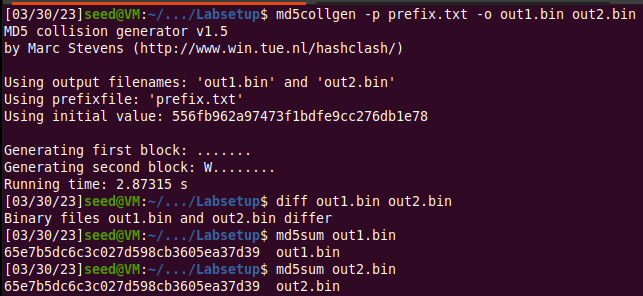
out2.bin

There are quite a few bytes that are different. The offset locations that are different are: 0155, 0156, 0173, 0255, 0273.

**Task – 2:**

In this task, we generate two output files for a prefix, and then add a common suffix to both. This would demonstrate the property that adding the same suffix T to two input files with the same hash will result in two outputs that have the same hash value.

We will reuse the same output.bin files from the first task.



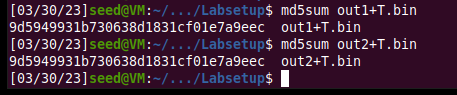
Now we use echo to generate a new suffix file called T.txt –



After this, the cat command is used to concatenate T.txt to the end of both out1.bin and out2.bin to get 2 new output files.



Generating the md5 sum hash values of both files –



This proves that they are indeed the same. Thus, the property has been successfully demonstrated.

**Task – 3:**

In this task, the goal is to create two different versions of the given C program, such that the contents of their xyz arrays are different, but the hash values of the executables are the same.Here is the program –

#include <stdio.h>

unsigned char xyz[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

};

int main()

{

  int i;

  for (i=0; i<200; i++){

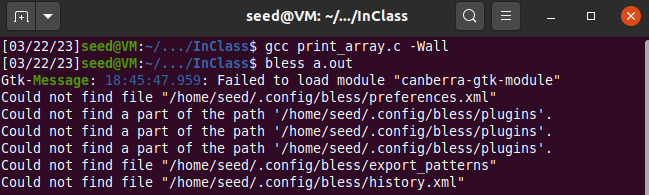
    printf("%x", xyz[i]);

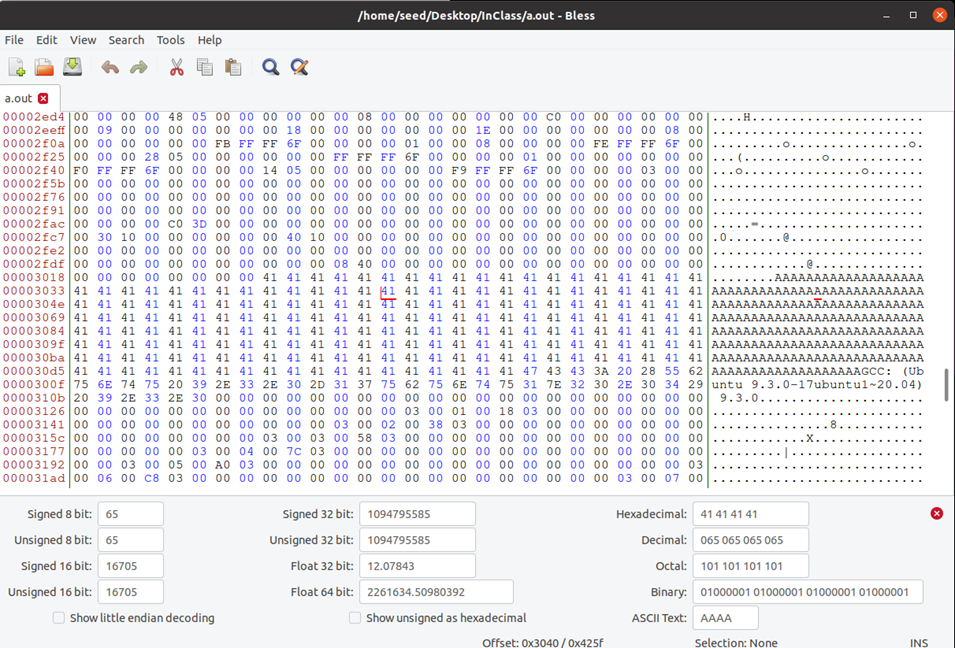
  }

  printf("\n");

}

Compiling print\_array.c into a.out using GCC and opening this using Bless:

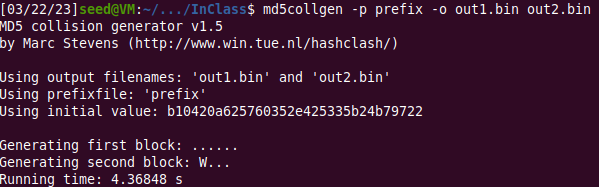




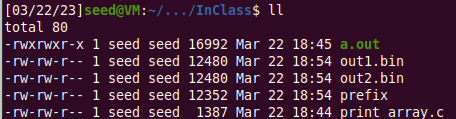
From the above screenshot, we find the necessary offset value to be 0x3040. Now we run the head command using the offset value to generate the prefix -.



Using md5collgen to generate 2 files with the same md5 hash value –



Now, we run the ll command to look at the files. The binary files have a size of 12480 bytes and the prefix has a size of 12352 bytes, so we start the suffix from 12481.



Generating the suffix using the offset value –



Generating P and Q -



Now we concatenate the prefix and suffix with P and then Q -



Giving permissions to both a1.out and a2.out -



Now we check to see if they are the same or not -

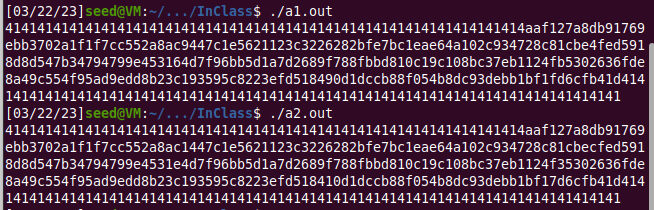


This shows that they both differ.

Using md5sum to calculate the md5 hash value from both these two files -



Displaying the two files -



Thus, we have successfully created two different versions of the given C program, such that the contents of their xyz arrays are different, but the hash values of the executables are the same.

**Task – 4:**

In this task, we have to showcase the attack described and generate two versions of the same program array with the same hash values. We have to create two programs that share the same MD5 hash. However, one program will always execute benign instructions, while the other program will execute malicious instructions.

Here is how the program looks like -

#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... ");

  }

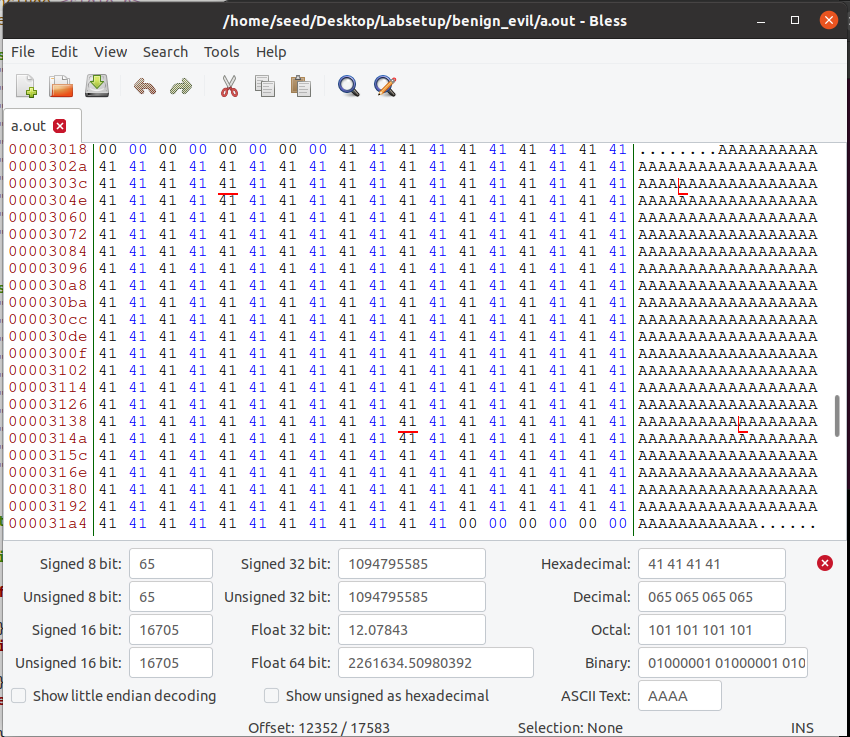
  return 0;

}

Now we execute this with GCC.



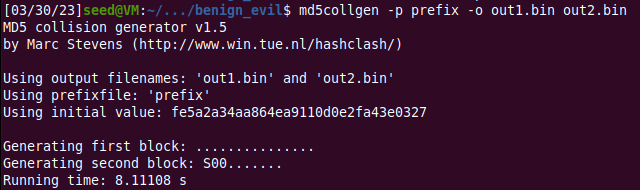
The next step is to look at the generated binary file with Bless -



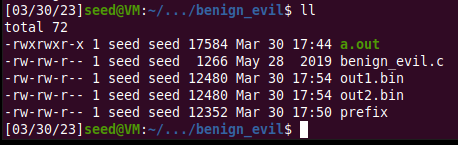
In the first array, we need to find a position that makes a partition before a multiple of 64 – I chose 12352 (193 \* 64 = 12352). This offset is shown in the above screenshot. So, the first 12352 bytes of a.out are copied into a file called prefix.



We use md5collgen to generate two output files called out1.bin and out2.bin, which would be prefix + P and prefix + Q -



Running the ll command to look at the files -



The binary files have a size of 12480 bytes and the prefix has a size of 12352 bytes, so we start the suffix from 12481.



We need to overwrite the next 128 bytes of the array with P in the first file and with Q in the second file. So, we use tail to do this (12352 + 128 = 12480).



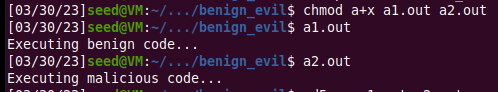
The suffix array has suffix\_1, P/Q and suffix\_2. So, we overwrite the first 288 bytes with suffix\_1 and everything but the first 417 bytes with suffix\_2 -



The next step is to concatenate the files that we have found together into 2 output files. The first file has prefix + P + suffix\_1 + P + suffix\_2 and the second file has prefix + P + suffix\_1 + Q + suffix\_2.



We give both files the necessary permissions and execute them. As we can see from the below screenshot, one executes benign code, while the other executes malicious code.



Finally, we check the md5 hash sum value of both files. They are the same -



Thus, we have successfully demonstrated the attack described.