|  |
| --- |
| **CPS633 Section 07 Fall2021** |
| Lab 04 Report |
| **MD5 Collision attack Lab**  **Name: Tusaif Azmat (group leader)**  **Student#: 500660278.**  **And**  **Name: Ankit Sodhi**  **Student#: 500958004**  **Group 04.** |

**CPS 633 - Lab 4 Report**

**MD5 Collision Attack Lab**

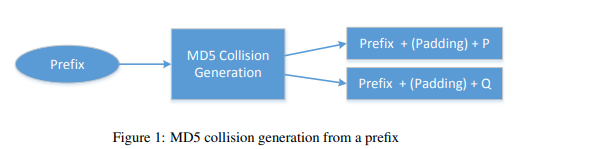
**2 Lab Tasks:**

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

In this task, we will 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. We can achieve this using the md5collgen program, which allows us to provide a prefix file with any arbitrary content.

The following command generates two output files, out1.bin and out2.bin, for a given a prefix file prefix.txt:

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



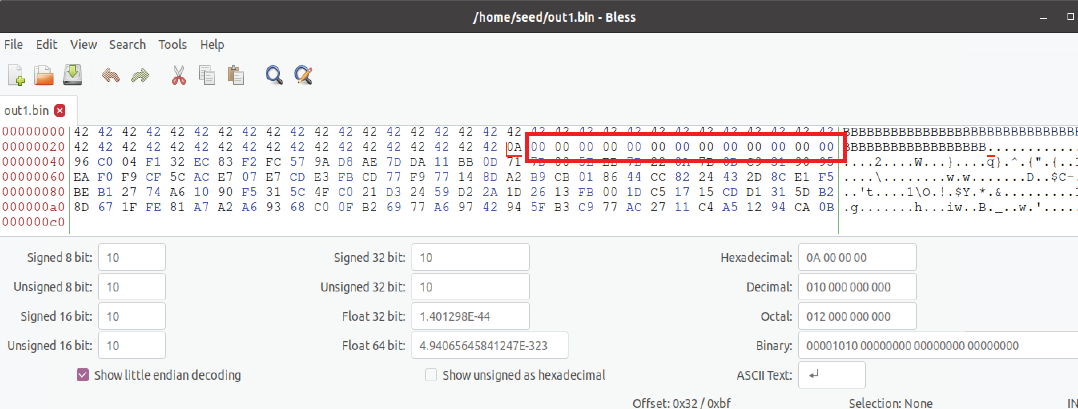
We can check whether the output files are distinct or not using the diff command. We can also use the md5sum command to check the MD5 hash of each output file. See the following commands.



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

**Answer:** let’s say we have 50 bytes and use for prefix.txt with that number which is less than 64. After running the commands we will see our output file is padded with extra zeros to make it 64 bytes.

As you could see in the image below:

















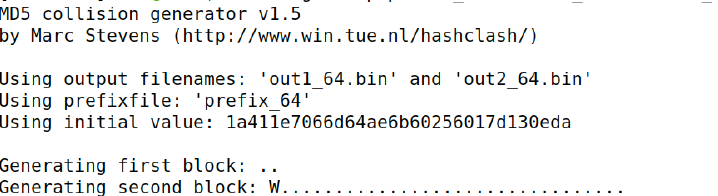


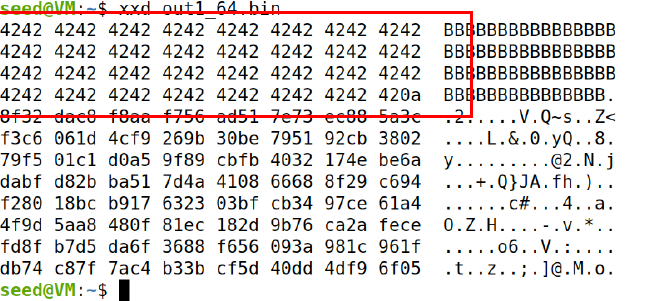
**– Question 2. Create a prefix file with exactly 64 bytes, and run the collision tool again, and see what happens.**

**Answer:** If we create a prefix file with exactly 64 bytes and run the collision tool again. We will see that it has not added any extra padding to it.

See screen shot below:







**– 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.**

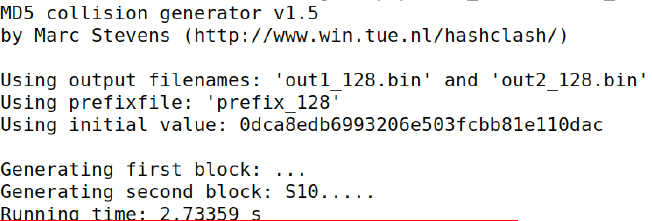
**Answer:** we ran the following commands and find that there are few differences. We suspect this may cause a collision error, but we shall see.

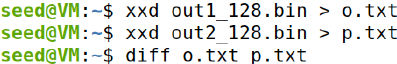
As you could see below:

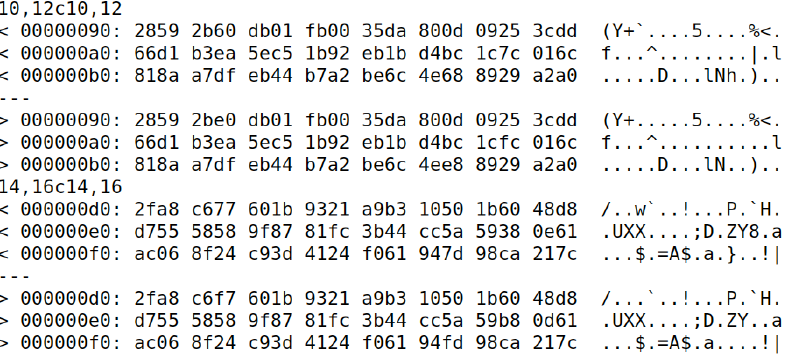










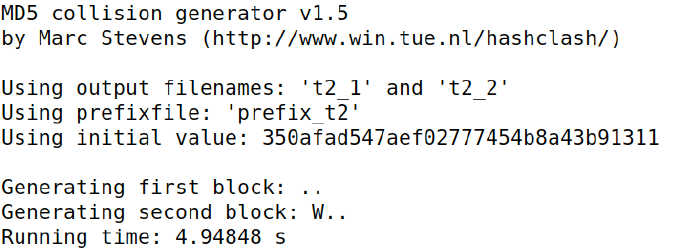


**2.2 Task 2: Understanding MD5’s Property**

To achieve this task of understanding MD5’s property, we create a prefix\_t2 with “cpssixthreet” and suffix\_t2 “63306330C” respectively. By doing md5collgen on prefix\_t2 to create t2\_1 and t2\_2 to test that the two different files have same hash values. We tested the two strings created above as pairs. We tested string t2\_1with suffix\_t2 together and t2\_2 with suffix\_t2 and created two more files t2\_1\_done and t2\_2\_done. We tested if they were same or different but should have the same hash values. It proved to the property of MD5’s.

As we can observe our test below:







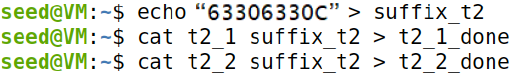




















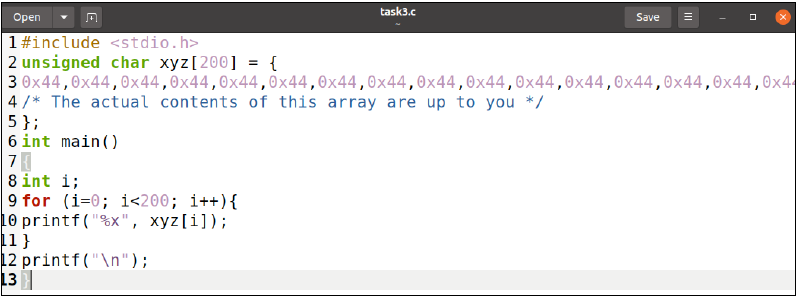




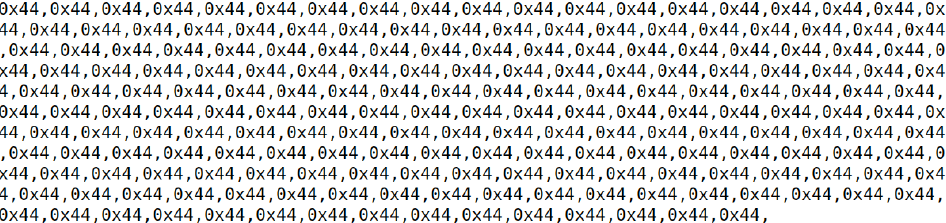
**2.3 Task 3: Generating Two Executable Files with the Same MD5 Hash**

For this task we created the two executable files with same MD5 hash to perform our findings.

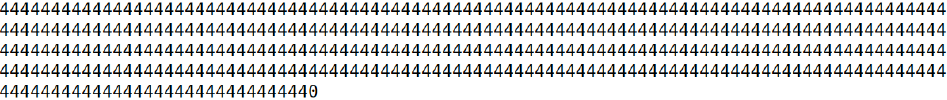
To achieve our task, first we created a file name task3.c with array of char 200 and save the output in D.txt file.



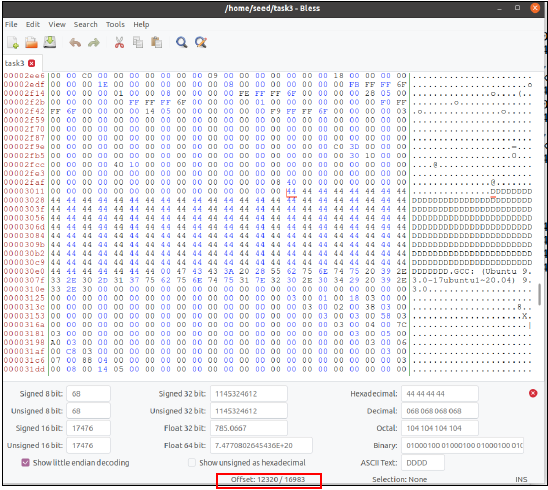










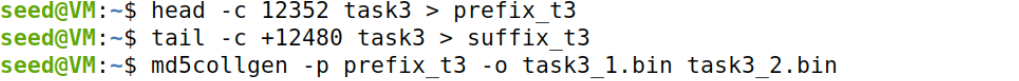


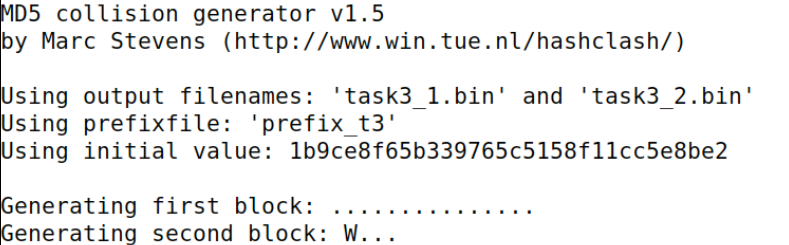
From the above file task.3 bless you see that the offset for the D is 12320. According to the tile the prefix length must be an integer multiple of 64, we get after calculating 12320 mod 64 = 32, and add it to the offset 12320+32=12352 and makes the prefix\_t3 of 128 D that followed by suffix and we get suffix’s offset that is 12352+128=12480. We get our p and q values here.

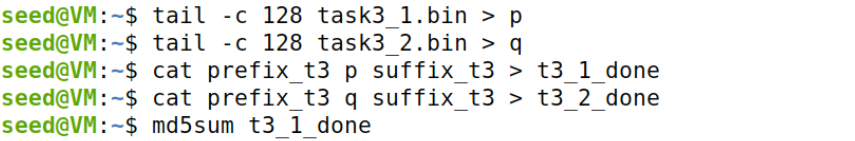
When we finish prefix\_t3 and suffix\_t3, we use prefix\_t3 to generate task3\_1.bin and task3\_2.bin. After generation of the two files we take separately 128 bytes from the end of each file to get as p and q values.

We only need to put prefix\_t3 p|q suffix\_t3 string together to get the two file. Then we verify the MD5 has value and execute it with chmod +x <file> that will help us to confirm the differences among them.

Check below for the screen shots of the process:





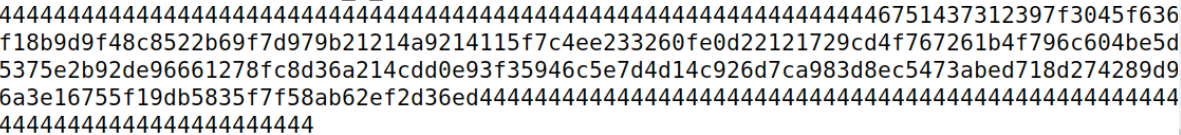




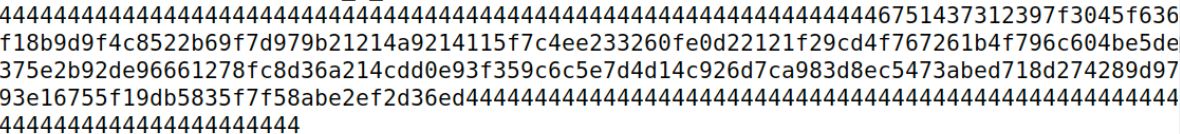








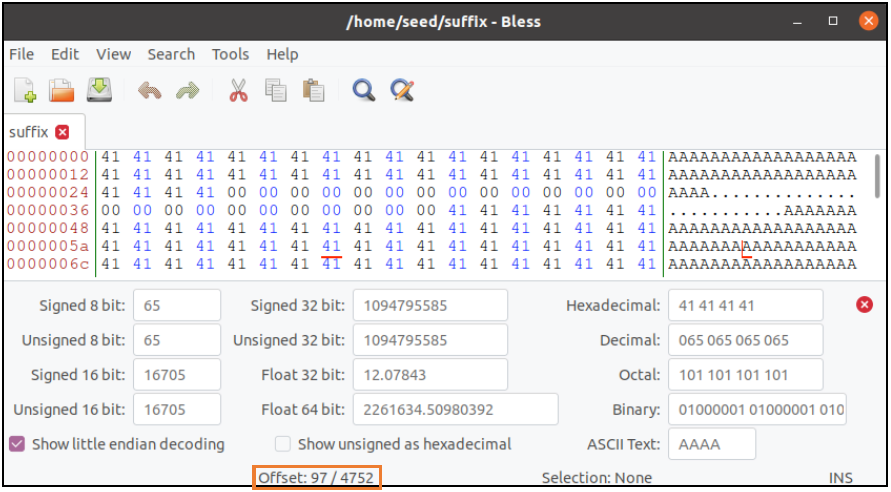
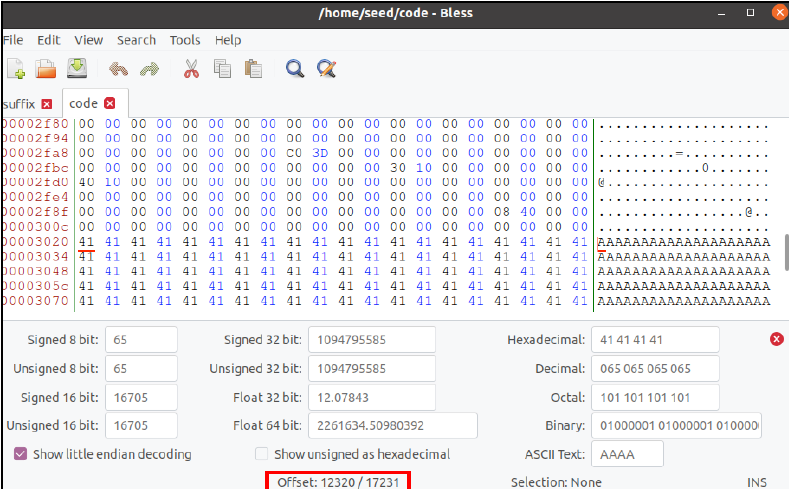
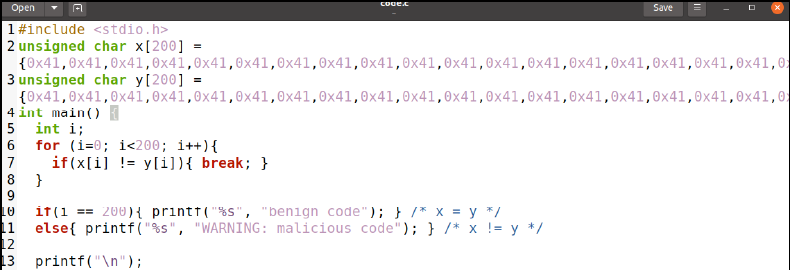




**2.4 Task 4: Making the Two Programs Behave Differently**

In our approach, we create two arrays X and Y. We compare the contents of these two arrays; if they are the same, the benign code is executed; otherwise, the malicious code is executed.

We create code.c program and make our judgement from there. If an array X and Y are same then could will begin execution otherwise it’s a malicious code.

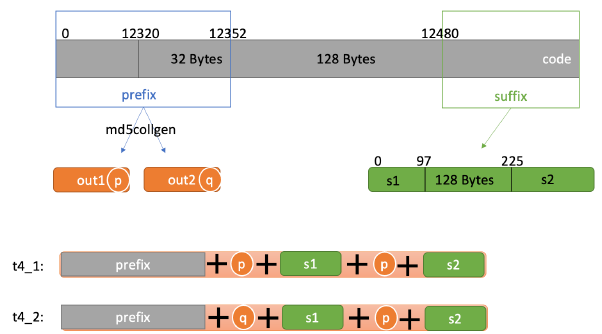


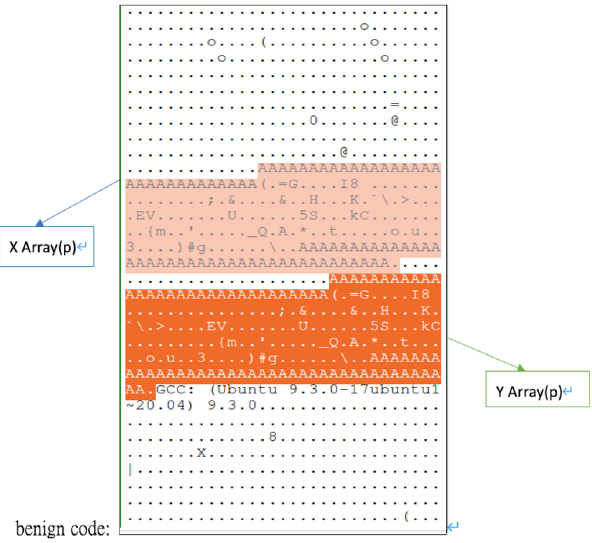
Our Goal is to do as follows:

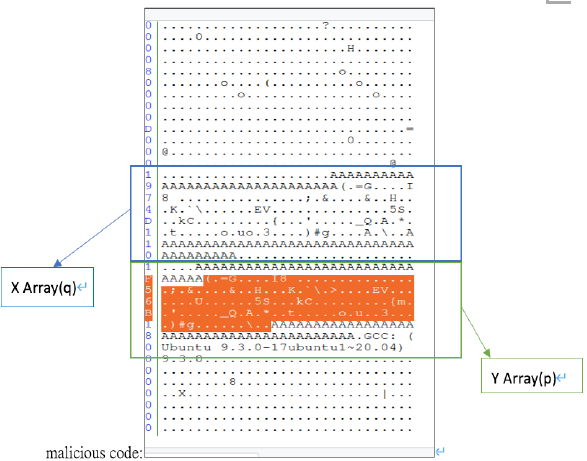
| 12352 | p | 64 | p | 4512 | > task4\_1

| 12352 | q | 64 | p | 4512 | > task4\_2

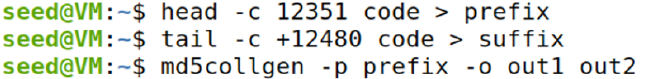
As can been seen below:

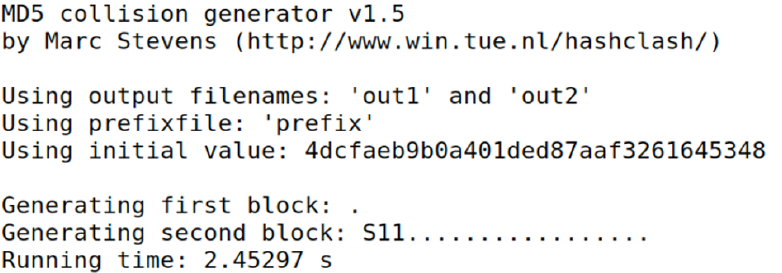


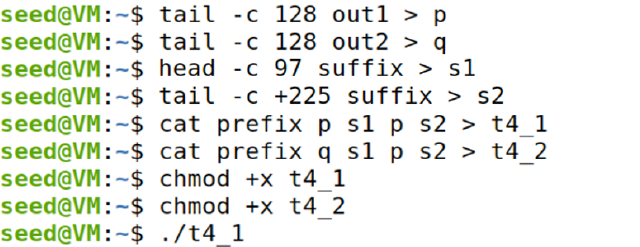




This is how we conclude the task 4:













Hence, the “WARNING: malicious code” proves that the two programs behaved differently